FRÉDÉRIC ESPIAU

CRÉEZ DES APPLICATIONS POUR ANDROID

LE DÉVELOPPEMENT POUR APPAREILS MOBILES ANDROID À LA PORTÉE DE TOUS

www.siteduzero.com

DANS LA MÊME COLLECTION

CONCEVEZ VOTRE SITE WEB AVEC PHP ET MYSQL

MATHIEU NEBRA
ISBN : 978-2-9535278-1-0

RÉDIGEZ DES DOCUMENTS DE QUALITÉ AVEC LATEX

NOËL-ARNAUD MAGUIS
ISBN : 978-2-9535278-4-1

PROGRAMMEZ AVEC LE LANGAGE C++

M.NEBRA ET M.SCHALLER
ISBN : 978-2-9535278-5-8

RÉDIGEZ FACILEMENT DES DOCUMENTS AVEC WORD

MICHEL MARTIN
ISBN : 978-2-9535278-7-2

APPRENEZ À PROGRAMMER EN PYTHON

VINCENT LE GOFF
ISBN : 979-10-90085-03-9

RÉALISEZ VOTRE SITE WEB AVEC HTML5 ET CSS3

MATHIEU NEBRA
ISBN : 978-2-9535278-8-9

DÉBUTEZ DANS LA 3D AVEC BLENDER

ANTOINE VEYRAT
ISBN : 978-2-9535278-9-6

APPRENEZ À PROGRAMMER EN C • 2e ÉDITION

MATHIEU NEBRA
ISBN : 979-10-90085-00-8

APPRENEZ À DÉVELOPPER EN C#

NICOLAS HILAIRE
ISBN : 978-2-9535278-6-5

CRÉEZ DES APPLICATIONS POUR IPHONE, IPAD ET IPOD TOUCH

MICHEL MARTIN
ISBN : 979-10-90085-06-0

ADMINISTREZ VOS BASES DE DONNÉES AVEC MYSQL

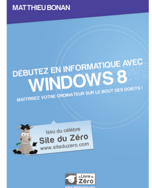
CHANTAL GRIBAUMONT
ISBN : 979-10-90085-10-7

REPRENEZ LE CONTRÔLE À L'AIDE DE LINUX • 2e ÉDITION

MATHIEU NEBRA
ISBN : 979-10-90085-23-7

SIMPLIFIEZ VOS DÉVELOPPEMENTS JAVASCRIPT AVEC JQUERY

MICHEL MARTIN
ISBN : 979-10-90085-25-1



DÉBUTEZ EN INFORMATIQUE AVEC WINDOWS 8

MATTHIEU BONAN
ISBN : 979-10-90085-26-8

APPRENEZ À PROGRAMMER EN JAVA

CYRILLE HERBY
ISBN : 979-10-90085-07-7

PROGRAMMEZ EN ORIENTÉ OBJET EN PHP

VICTOR THUILLIER
ISBN : 979-10-90085-36-7

CRÉEZ DES APPLICATIONS POUR WINDOWS 8 EN HTML ET JAVASCRIPT

XAVIER BOUBERT
ISBN : 979-10-90085-33-6

Retrouvez aussi tous nos titres en version eBook !

FRÉDÉRIC ESPIAU

CRÉEZ DES APPLICATIONS POUR ANDROID

LE DÉVELOPPEMENT POUR APPAREILS MOBILES ANDROID À LA PORTÉE DE TOUS

www.siteduzero.com

Sauf mention contraire, le contenu de cet ouvrage est publié sous la licence :
Creative Commons BY-NC-SA 2.0
La copie de cet ouvrage est autorisée sous réserve du respect des conditions de la licence
Texte complet de la licence disponible sur : http://creativecommons.org/licenses/by-nc-sa/2.0/fr/

Simple IT 2012 - ISBN : 979-10-90085-37-4

Avant-propos

Connaissez -vous le nombre de téléphones portables dans le monde ? En octobre 2012, les opérateurs déclaraient qu'il y avait plus de 6 milliards d'abonnements mobiles. Cela fait presque un abonnement par être-humain ! Pour être tout à fait exact, il n'est pas rare dans les pays développés qu'une personne ait plusieurs abonnements, par exemple un professionnel et un personnel. C'est pourquoi à l'heure actuelle ce marché est aussi florissant. Et saviez-vous qu'Android fait fonctionner plus de trois quart des téléphones vendus dans le monde en 2012 ? Et c'est sans compter les tablettes numériques et autres Google TV ! Avec ces chiffres, vous devriez voir l'immense potentiel qu'est le développement d'applications pour Android. Cela permet d'impacter beaucoup de gens, d'influencer leur quotidien, de faire partie d'une communauté planétaire. Les utilisateurs passent beaucoup de temps sur leur téléphone : dans le métro, au lit avant de dormir, pendant les pubs à la télé, etc. Si tout ceci vous intéresse, je pense que ce livre devrait vous intéresser ! Il vous permettra de prendre part à une véritable révolution numérique et de conquérir un média qui entre au cœur de notre vie et de celle de millions de personnes.

Les origines de ce livre

Quand j'ai commencé à apprendre la programmation pour Android, j'ai été très déçu par les différentes ressources que j'ai pu trouver. Il y avait bien entendu de bonnes sources en anglais, par exemple la documentation officielle qui regorge d'exemples et conseils en tout genre, mais pas vraiment de guide en français dans lequel je puisse me retrouver. C'est pourquoi j'ai retroussé mes manches dans l'espoir de permettre aux personnes intéressées de pouvoir se mettre à la programmation Android. Étant un gros consommateur du Site du Zéro, mon premier réflexe a été de commencer à y écrire un cours qui me ressemble dans l'espoir de toucher un maximum de monde. Après un an et demi d'efforts, le résultat de mon travail est entre vos mains. Prenez en soin, j'y ai mis une partie de moi. :-)

Qu'allez-vous apprendre en lisant ce livre ?

Ce livre est organisé de manière peu habituelle en comparaison d'autres ouvrages qui traitent du même sujet. En effet, j'ai souvent trouvé des cours qui traitaient par exemple de l'interface graphique, puis d'un élément architectural d'Android, puis de la suite de l'interface graphique... Je trouve ce type d'organisation confus. De ce fait, les chapitres sont regroupés par thématique de manière à pouvoir les retrouver plus facilement et à conserver une certaine cohérence.

Ainsi, ce livre est divisé en cinq parties :

- **Les bases indispensables à toutes applications** contient une introduction avec un rappel rapide sur certains concepts du Java, puis un guide pour télécharger tous les éléments afin que vous puissiez développer vos applications Android. Enfin, ce chapitre vous présentera les composants qu'on retrouve dans absolument toutes les applications qui existent.
- **Création d'interfaces graphiques** vous permettra d'explorer toutes les possibilités qui sont offertes par Android en ce qui concerne les interfaces graphiques. Une interface graphique étant le premier contact entre votre application et l'utilisateur, il est essentiel que vous sachiez en construire d'assez séduisantes. Dans le cas contraire, les utilisateurs pourraient ne pas avoir le courage de continuer à explorer le contenu de votre application.
- **Vers des applications plus complexes** est essentiel quand on veut réaliser une application réellement complète. Ici, nos applications prendront de l'ampleur. Nous apprendrons à élargir leur contenu ainsi qu'à passer d'une application sans état - qui ne retient rien - à une application capable de récupérer du contenu sauvegardé.
- **Concepts avancés** sera quant à elle une partie légèrement différente des précédentes. En effet, les techniques présentées ici seront moins courantes et moins faciles à maîtriser. Néanmoins, on y parlera de concepts très importants, comme l'optimisation en abordant le travail en arrière-plan, ou encore le partage de contenus entre applications.
- **Exploiter les fonctionnalités d'Android** aborde les différentes possibilités offertes par le système afin d'exploiter la majorité des options proposées par les terminaux sur lesquels fonctionneront vos applications. Nous verrons ainsi à quel point il est simple de créer un lecteur multimédia, d'utiliser le réseau GPS, de se connecter à internet ou encore de travailler avec différents capteurs contenus dans l'appareil.

Comment lire ce livre ?

Suivez l'ordre des chapitres

Lisez ce livre comme on lit un roman. Il a été conçu pour cela.

Contrairement à beaucoup de livres techniques où il est courant de lire en diagonale et de sauter certains chapitres, il est ici très fortement recommandé de suivre l'ordre du cours, à moins que vous ne soyez déjà un peu expérimentés.

Pratiquez en même temps

Pratiquez régulièrement. N'attendez pas d'avoir fini de lire ce livre pour allumer votre ordinateur et faire vos propres essais.

Utilisez les codes web !

Afin de tirer parti du Site du Zéro dont ce livre est issu, celui-ci vous propose ce qu'on appelle des « codes web ». Ce sont des codes à six chiffres à saisir sur une page du Site du Zéro pour être automatiquement redirigé vers un site web sans avoir à en recopier l'adresse.

Pour utiliser les codes web, rendez-vous sur la page suivante [1] :

`http://www.siteduzero.com/codeweb.html`

Un formulaire vous invite à rentrer votre code web. Faites un premier essai avec le code ci-dessous :

▷ Tester le code web
Code web : 123456

Ces codes web ont deux intérêts :

- ils vous redirigent vers les sites web présentés tout au long du cours, vous permettant ainsi d'obtenir les logiciels dans leur toute dernière version ;
- ils vous permettent de télécharger les codes sources inclus dans ce livre, ce qui vous évitera d'avoir à recopier certains programmes un peu longs.

Ce système de redirection nous permet de tenir à jour le livre que vous avez entre les mains sans que vous ayez besoin d'acheter systématiquement chaque nouvelle édition. Si un site web change d'adresse, nous modifierons la redirection mais le code web à utiliser restera le même. Si un site web disparaît, nous vous redirigerons vers une page du Site du Zéro expliquant ce qui s'est passé et vous proposant une alternative.

En clair, c'est un moyen de nous assurer de la pérennité de cet ouvrage sans que vous ayez à faire quoi que ce soit !

Remerciements

Il y a des gens sans qui tout cela ne serait pas possible :

- Ma femme Anaïs, pour tout ce qu'elle fait, pour tout son support et son amour.
- Jonathan Baudoin pour toute son aide, son soutien, son expertise, son travail, et tous les bons moments que nous avons passé ensemble à se raconter des âneries.
- Gérard Paligot, alias AndroWiiid, pour avoir lu, relu, rerelu, ragé, rererelu, pause-café, rerererelu le cours, et m'avoir indiqué mes erreurs et imprécisions. Il a donné tellement de son temps, et effectué du travail d'une qualité telle que je pense qu'il doit être difficile de trouver quelqu'un de plus patient et consciencieux sur Terre.

1. Vous pouvez aussi utiliser le formulaire de recherche du Site du Zéro, section « Code web ».

- Le reste du staff chez Simple IT qui a travaillé en background sans que je puisse même savoir ce qu'ils ont fait.
- Tous les lecteurs, en particuliers les plus motivés qui ont été capables de me laisser des commentaires pertinents me permettant d'améliorer encore le contenu du cours.
- J'en profite pour faire un clin d'œil à ma famille, mes copains, tous les UTCéens et tout JELB.

Sommaire

Avant-propos **i**

Les origines de ce livre i
Qu'allez-vous apprendre en lisant ce livre ? ii
Comment lire ce livre ? ii
Remerciements iii

I Les bases indispensables à toute application **1**

1 L'univers Android **3**

La création d'Android 4
La philosophie et les avantages d'Android 5
Les difficultés du développement pour des systèmes embarqués 6
Le langage Java 7

2 Installation et configuration des outils **13**

Conditions initiales 14
Le Java Development Kit 14
Le SDK d'Android 15
L'IDE Eclipse 18
L'émulateur de téléphone : Android Virtual Device 21
Test et configuration 24
Configuration du vrai terminal 29

3 Votre première application **33**

Activité et vue 34

Création d'un projet 39

Un non-Hello world ! 45

Lancement de l'application 48

4 Les ressources **51**

Le format XML 52

Les différents types de ressources 54

L'organisation 57

Ajouter un fichier avec Eclipse 60

Récupérer une ressource 63

II Création d'interfaces graphiques **71**

5 Constitution des interfaces graphiques **73**

L'interface d'Eclipse 74

Règles générales sur les vues 78

Identifier et récupérer des vues 83

6 Les widgets les plus simples **89**

Les widgets 90

Gérer les évènements sur les widgets 101

7 Organiser son interface avec des layouts **115**

LinearLayout : placer les éléments sur une ligne 116

RelativeLayout : placer les éléments les uns en fonction des autres 125

TableLayout : placer les éléments comme dans un tableau 132

FrameLayout : un layout un peu spécial 137

ScrollView : faire défiler le contenu d'une vue 138

8 Les autres ressources **141**

Aspect général des fichiers de ressources 142

Les chaînes de caractères 145

Les drawables 149

Les styles 153
Les animations 154

9 TP : un bloc-notes 163
Objectif 164
Spécifications techniques 165
Déboguer des applications Android 169
Ma solution 172
Objectifs secondaires 186

10 Des widgets plus avancés et des boîtes de dialogue 189
Les listes et les adaptateurs 190
Plus complexe : les adaptateurs personnalisés 204
Les boîtes de dialogue 208
Les autres widgets 215

11 Gestion des menus de l'application 225
Menu d'options 226
Menu contextuel 231
Maintenant que vous maîtrisez les menus, oubliez tout 232

12 Création de vues personnalisées 235
Règles avancées concernant les vues 236
Méthode 1 : à partir d'une vue préexistante 240
Méthode 2 : une vue composite 244
Méthode 3 : créer une vue en partant de zéro 247

III Vers des applications plus complexes 255

13 Préambule : quelques concepts avancés 257
Généralités sur le nœud `<manifest>` 258
Le nœud <application> 261
Les permissions 263
Gérer correctement le cycle des activités 264
Gérer le changement de configuration 269

14 La communication entre composants **275**

Aspect technique 276

Les intents explicites 279

Les intents implicites 285

La résolution des intents 288

Pour aller plus loin : navigation entre des activités 293

Pour aller plus loin : diffuser des intents 297

15 Le stockage de données **301**

Préférences partagées 302

Manipulation des fichiers 308

16 TP : un explorateur de fichiers **315**

Objectifs 316

Spécifications techniques 318

Ma solution 326

Améliorations envisageables 336

17 Les bases de données **339**

Généralités 340

Création et mise à jour 342

Opérations usuelles 345

Les curseurs 351

IV Concepts avancés **355**

18 Le travail en arrière-plan **357**

La gestion du multitâche par Android 358

Gérer correctement les threads simples 361

AsyncTask 369

19 Les services **379**

Qu'est-ce qu'un service ? 380

Gérer le cycle de vie d'un service 381

Créer un service 387

Les notifications et services de premier plan 390
Pour aller plus loin : les alarmes 398

20 Le partage de contenus entre applications **401**
Côté client : accéder à des fournisseurs 402
Créer un fournisseur 408

21 Créer un AppWidget **417**
L'interface graphique 418
Définir les propriétés 420
Le code 420
Déclarer l'AppWidget dans le Manifest 422
Application : un AppWidget pour accéder aux tutoriels du Site du Zéro . . . 423

V Exploiter les fonctionnalités d'Android **433**

22 La connectivité réseau **435**
Surveiller le réseau 436
Afficher des pages Web 437
Effectuer des requêtes HTTP 438

23 Apprenez à dessiner **445**
La toile 446
Afficher notre toile 448

24 La localisation et les cartes **455**
La localisation 456
Afficher des cartes 460

25 La téléphonie **471**
Téléphoner 472
Envoyer et recevoir des SMS et MMS 476

26 Le multimédia **479**
Le lecteur multimédia 480
Enregistrement 484

27 Les capteurs **495**

Les différents capteurs . . . 496

Opérations génériques . . . 496

Les capteurs de mouvements . . . 501

Les capteurs de position . . . 502

Les capteurs environnementaux . . . 504

28 TP : un labyrinthe **505**

Objectifs . . . 506

Spécifications techniques . . . 507

Ma solution . . . 515

Améliorations envisageables . . . 530

VI Annexes **535**

29 Publier et rentabiliser une application **537**

Préparez votre application à une distribution . . . 538

Les moyens de distribution . . . 544

Rentabilisez votre application . . . 552

30 L'architecture d'Android **561**

Le noyau Linux . . . 562

Le moteur d'exécution d'Android . . . 563

Première partie

Les bases indispensables à toute application

Chapitre 1

L'univers Android

Difficulté :

Dans ce tout premier chapitre, je vais vous présenter ce que j'appelle l'« univers Android » ! Le système, dans sa genèse, part d'une idée de base simple, et très vite son succès fut tel qu'il a su devenir indispensable pour certains constructeurs et utilisateurs, en particulier dans la sphère de la téléphonie mobile. Nous allons rapidement revenir sur cette aventure et sur la philosophie d'Android, puis je rappellerai les bases de la programmation en Java, pour ceux qui auraient besoin d'une petite piqûre de rappel. . .

La création d'Android

Quand on pense à Android, on pense immédiatement à Google, et pourtant il faut savoir que cette multinationale n'est pas à l'initiative du projet. D'ailleurs, elle n'est même pas la seule à contribuer à plein temps à son évolution. À l'origine, « Android » était le nom d'une PME américaine, créée en 2003 puis rachetée par Google en 2005, qui avait la ferme intention de s'introduire sur le marché des produits mobiles. La gageure, derrière Android, était de développer un système d'exploitation mobile plus intelligent, qui ne se contenterait pas uniquement de permettre d'envoyer des SMS et transmettre des appels, mais qui devait permettre à l'utilisateur d'interagir avec son environnement (notamment avec son emplacement géographique). C'est pourquoi, contrairement à une croyance populaire, il n'est pas possible de dire qu'Android est une réponse de Google à l'iPhone d'Apple, puisque l'existence de ce dernier n'a été révélée que deux années plus tard.

C'est en 2007 que la situation prit une autre tournure. À cette époque, chaque constructeur équipait son téléphone d'un système d'exploitation propriétaire. Chaque téléphone avait ainsi un système plus ou moins différent. Ce système entravait la possibilité de développer facilement des applications qui s'adapteraient à tous les téléphones, puisque la base était complètement différente. Un développeur était plutôt spécialisé dans un système particulier et il devait se contenter de langages de bas niveaux comme le C ou le C++. De plus, les constructeurs faisaient en sorte de livrer des bibliothèques de développement très réduites de manière à dissimuler leurs secrets de fabrication. En janvier 2007, Apple dévoilait l'iPhone, un téléphone tout simplement révolutionnaire pour l'époque. L'annonce est un désastre pour les autres constructeurs, qui doivent s'aligner sur cette nouvelle concurrence. Le problème étant que pour atteindre le niveau d'iOS (iPhone OS), il aurait fallu des années de recherche et développement à chaque constructeur…

C'est pourquoi est créée en novembre 2007 l'Open Handset Alliance (que j'appellerai désormais par son sigle OHA), et qui comptait à sa création 35 entreprises évoluant dans l'univers du mobile, dont Google. Cette alliance a pour but de développer un système *open source* (c'est-à-dire dont les sources sont disponibles librement sur internet) pour l'exploitation sur mobile et ainsi concurrencer les systèmes propriétaires, par exemple Windows Mobile et iOS. Cette alliance a pour logiciel vedette Android, mais il ne s'agit pas de sa seule activité. L'OHA compte à l'heure actuelle 80 membres.

Android est à l'heure actuelle le système d'exploitation pour smartphones et tablettes le plus utilisé.

Les prévisions en ce qui concerne la distribution d'Android sur le marché sont très bonnes avec de plus en plus de machines qui s'équipent de ce système. Bientôt, il se trouvera dans certains téléviseurs (vous avez entendu parler de Google TV, peut-être ?) et les voitures. Android sera partout. Ce serait dommage de ne pas faire partie de ça, n'est-ce pas ?

La philosophie et les avantages d'Android

Open source

Le contrat de licence pour Android respecte les principes de l'*open source*, c'est-à-dire que vous pouvez à tout moment télécharger les sources et les modifier selon vos goûts ! Bon, je ne vous le recommande vraiment pas, à moins que vous sachiez ce que vous faites... Notez au passage qu'Android utilise des bibliothèques *open source* puissantes, comme par exemple SQLite pour les bases de données et OpenGL pour la gestion d'images 2D et 3D.

Gratuit (ou presque)

Android est gratuit, autant pour vous que pour les constructeurs. S'il vous prenait l'envie de produire votre propre téléphone sous Android, alors vous n'auriez même pas à ouvrir votre porte-monnaie (mais bon courage pour tout le travail à fournir !). En revanche, pour poster vos applications sur le Play Store, il vous en coûtera la modique somme de 25$. Ces 25$ permettent de publier autant d'applications que vous le souhaitez, à vie !

Facile à développer

Toutes les API mises à disposition facilitent et accélèrent grandement le travail. Ces APIs sont très complètes et très faciles d'accès. De manière un peu caricaturale, on peut dire que vous pouvez envoyer un SMS en seulement deux lignes de code (concrètement, il y a un peu d'enrobage autour de ce code, mais pas tellement).

Une API, ou « interface de programmation » en français, est un ensemble de règles à suivre pour pouvoir dialoguer avec d'autres applications. Dans le cas de Google API, il permet en particulier de communiquer avec Google Maps.

Facile à vendre

Le *Play Store* (anciennement *Android Market*) est une plateforme immense et très visitée ; c'est donc une mine d'opportunités pour quiconque possède une idée originale ou utile.

Flexible

Le système est extrêmement portable, il s'adapte à beaucoup de structures différentes. Les smartphones, les tablettes, la présence ou l'absence de clavier ou de *trackball*, différents processeurs... On trouve même des fours à micro-ondes qui fonctionnent à l'aide d'Android ! Non seulement c'est une immense chance d'avoir autant d'opportunités,

mais en plus Android est construit de manière à faciliter le développement et la distribution en fonction des composants en présence dans le terminal (si votre application nécessite d'utiliser le Bluetooth, seuls les terminaux équipés de Bluetooth pourront la voir sur le Play Store).

Ingénieux

L'architecture d'Android est inspirée par les applications composites, et encourage par ailleurs leur développement. Ces applications se trouvent essentiellement sur internet et leur principe est que vous pouvez combiner plusieurs composants totalement différents pour obtenir un résultat surpuissant. Par exemple, si on combine l'appareil photo avec le GPS, on peut poster les coordonnées GPS des photos prises.

Les difficultés du développement pour des systèmes embarqués

Il existe certaines contraintes pour le développement Android, qui ne s'appliquent pas au développement habituel !

Prenons un cas concret : la mémoire RAM est un composant matériel indispensable. Quand vous lancez un logiciel, votre système d'exploitation lui réserve de la mémoire pour qu'il puisse créer des variables, telles que des tableaux, des listes, etc. Ainsi, sur mon ordinateur, j'ai 4 Go de RAM, alors que je n'ai que 512 Mo sur mon téléphone, ce qui signifie que j'en ai huit fois moins. Je peux donc lancer moins de logiciels à la fois et ces logiciels doivent faire en sorte de réserver moins de mémoire. C'est pourquoi votre téléphone est dit limité, il doit supporter des contraintes qui font doucement sourire votre ordinateur.

Voici les principales contraintes à prendre en compte quand on développe pour un environnement mobile :

- Il faut pouvoir interagir avec un système complet sans l'interrompre. Android fait des choses pendant que votre application est utilisée, il reçoit des SMS et des appels, entre autres. Il faut respecter une certaine priorité dans l'exécution des tâches. Sincèrement, vous allez bloquer les appels de l'utilisateur pour qu'il puisse terminer sa partie de votre jeu de sudoku ? :-°
- Comme je l'ai déjà dit, le système n'est pas aussi puissant qu'un ordinateur classique, il faudra exploiter tous les outils fournis afin de débusquer les portions de code qui nécessitent des optimisations.
- La taille de l'écran est réduite, et il existe par ailleurs plusieurs tailles et résolutions différentes. Votre interface graphique doit s'adapter à toutes les tailles et toutes les résolutions, ou vous risquez de laisser de côté un bon nombre d'utilisateurs.
- Autre chose qui est directement lié, les interfaces tactiles sont peu pratiques en cas d'utilisation avec un stylet et/ou peu précises en cas d'utilisation avec les doigts, d'où des contraintes liées à la programmation événementielle plus rigides. En effet, il est possible que l'utilisateur se trompe souvent de bouton. Très souvent s'il a de

gros doigts.
– Enfin, en plus d'avoir une variété au niveau de la taille de l'écran, on a aussi une variété au niveau de la langue, des composants matériels présents et des versions d'Android. Il y a une variabilité entre chaque téléphone et même parfois entre certains téléphones identiques. C'est un travail en plus à prendre en compte.

Les conséquences de telles négligences peuvent être terribles pour l'utilisateur. Saturez le processeur et il ne pourra plus rien faire excepté redémarrer ! Faire crasher une application ne fera en général pas complètement crasher le système, cependant il pourrait bien s'interrompre quelques temps et irriter profondément l'utilisateur.

Il faut bien comprendre que dans le paradigme de la programmation classique vous êtes dans votre propre monde et vous n'avez vraiment pas grand-chose à faire du reste de l'univers dans lequel vous évoluez, alors que là vous faites partie d'un système fragile qui évolue sans anicroche tant que vous n'intervenez pas. Votre but est de fournir des fonctionnalités de plus à ce système et faire en sorte de ne pas le perturber.

Bon, cela paraît très alarmiste dit comme ça, Android a déjà anticipé la plupart des âneries que vous commettrez et a pris des dispositions pour éviter des catastrophes qui conduiront au blocage total du téléphone. Si vous êtes un tantinet curieux, je vous invite à lire l'annexe sur l'architecture d'Android pour comprendre un peu pourquoi il faut être un barbare pour vraiment réussir à saturer le système.

Le langage Java

Cette petite section permettra à ceux fâchés avec le Java de se remettre un peu dans le bain et surtout de réviser le vocabulaire de base. Notez qu'il ne s'agit que d'un rappel, il est conseillé de connaître la programmation en Java auparavant ; je ne fais ici que rappeler quelques notions de base pour vous rafraîchir la mémoire ! Il ne s'agit absolument pas d'une introduction à la programmation.

Les variables

La seule chose qu'un programme sait faire, c'est des calculs. Il arrive qu'on puisse lui faire afficher des formes et des couleurs, mais pas toujours. Pour faire des calculs, on a besoin de **variables**. Ces variables permettent de conserver des informations avec lesquelles on va pouvoir faire des opérations. Ainsi, on peut avoir une variable `radis` qui vaudra 4 pour indiquer qu'on a quatre radis. Si on a une variable `carotte` qui vaut 2, on peut faire le calcul `radis + carotte` de manière à pouvoir déduire qu'on a six légumes.

Les primitives

En Java, il existe deux types de variable. Le premier type s'appelle les **primitives**. Ces primitives permettent de retenir des informations simples telles que des nombres sans virgule (auquel cas la variable est un entier, `int`), des chiffres à virgule (des réels,

`float`) ou des booléens (variable qui ne peut valoir que *vrai* (`true`) ou *faux* (`false`), avec les `boolean`).

Cette liste n'est bien sûr pas exhaustive !

Les objets

Le second type, ce sont les **objets**. En effet, à l'opposé des primitives (variables simples), les objets sont des variables compliquées. En fait, une primitive ne peut contenir qu'une information, par exemple la valeur d'un nombre ; tandis qu'un objet est constitué d'une ou plusieurs autres variables, et par conséquent d'une ou plusieurs valeurs. Ainsi, un objet peut lui-même contenir un objet ! Un objet peut représenter absolument ce qu'on veut : une chaise, une voiture, un concept philosophique, une formule mathématique, etc. Par exemple, pour représenter une voiture, je créerai un objet qui contient une variable `roue` qui vaudra 4, une variable `vitesse` qui variera en fonction de la vitesse et une variable `carrosserie` pour la couleur de la carrosserie et qui pourra valoir « rouge », « bleu », que sais-je ! D'ailleurs, une variable qui représente une couleur ? Ça ne peut pas être une primitive, ce n'est pas une variable facile ça, une couleur ! Donc cette variable sera aussi un objet, ce qui signifie qu'un objet peut contenir des primitives ou d'autres objets.

Mais dans le code, comment représenter un objet ? Pour cela, il va falloir déclarer ce qu'on appelle une **classe**. Cette classe aura un nom, pour notre voiture on peut simplement l'appeler `Voiture`, comme ceci :

```
1 // On déclare une classe Voiture avec cette syntaxe
2 class Voiture {
3   // Et dedans on ajoute les attributs qu'on utilisera, par
      exemple le nombre de roues
4   int roue = 4;
5   // On ne connaît pas la vitesse, alors on ne la déclare pas
6   float vitesse;
7   // Et enfin la couleur, qui est représentée par une classe de
       nom Couleur
8   Couleur carrosserie;
9 }
```

Les variables ainsi insérées au sein d'une classe sont appelées des **attributs**.

Il est possible de donner des instructions à cette voiture, comme d'accélérer ou de s'arrêter. Ces instructions s'appellent des **méthodes**, par exemple pour freiner :

```
1 //Je déclare une méthode qui s'appelle "arreter"
2 void arreter() {
3   //Pour s'arrêter, je passe la vitesse à 0
4   vitesse = 0;
5 }
```

En revanche, pour changer de vitesse, il faut que je dise si j'accélère ou décélère et de combien la vitesse change. Ces deux valeurs données avant l'exécution de la méthode s'appellent des **paramètres**. De plus, je veux que la méthode rende à la fin de son exécution la nouvelle vitesse. Cette valeur rendue à la fin de l'exécution d'une méthode s'appelle une **valeur de retour**. Par exemple :

```
// On dit ici que la méthode renvoie un float et qu'elle a
    besoin d'un float et d'un boolean pour s'exécuter
float changer_vitesse(float facteur_de_vitesse, boolean
    acceleration)
  // S'il s'agit d'une accélération
  if(acceleration == true) {
    // On augmente la vitesse
    vitesse = vitesse + facteur_de_vitesse;
  }else {
    // On diminue la vitesse
    vitesse = vitesse - facteur_de_vitesse;
  }
  // La valeur de retour est la nouvelle vitesse
  return vitesse;
}
```

Parmi les différents types de méthode, il existe un type particulier qu'on appelle les **constructeurs**. Ces constructeurs sont des méthodes qui construisent l'objet désigné par la classe. Par exemple, le constructeur de la classe `Voiture` renvoie un objet de type `Voiture` :

```
// Ce constructeur prend en paramètre la couleur de la
    carrosserie
Voiture(Couleur carros) {
  // Quand on construit une voiture, elle a une vitesse nulle
  vitesse = 0;
  carrosserie = carros;
}
```

On peut ensuite construire une voiture avec cette syntaxe :

```
Voiture v = new Voiture(rouge);
```

Construire un objet s'appelle l'**instanciation**.

L'héritage

Il existe certains objets dont l'instanciation n'aurait aucun sens. Par exemple, un objet de type `Véhicule` n'existe pas vraiment dans un jeu de course. En revanche il est possible d'avoir des véhicules de certains types, par exemple des voitures ou des motos. Si je veux une moto, il faut qu'elle ait deux roues et, si j'instancie une voiture, elle doit avoir 4 roues, mais dans les deux cas elles ont des roues. Dans les cas de ce genre, c'est-à-dire quand plusieurs classes ont des attributs en commun, on fait appel à l'**héritage**.

Quand une classe A hérite d'une classe B, on dit que la classe A est la **fille** de la classe B et que la classe B est le **parent** (ou la **superclasse**) de la classe A.

```
// Dans un premier fichier
// Classe qui ne peut être instanciée
abstract class Vehicule {
  int nombre_de_roues;
  float vitesse;
}

// Dans un autre fichier
// Une Voiture est un Vehicule
class Voiture extends Vehicule {

}

// Dans un autre fichier
// Une Moto est aussi un Vehicule
class Moto extends Vehicule {

}

// Dans un autre fichier
// Un Cabriolet est une Voiture (et par conséquent un Véhicule)
class Cabriolet extends Voiture {

}
```

Le mot-clé `abstract` signifie qu'une classe ne peut être instanciée.

Une méthode peut aussi être `abstract`, auquel cas pas besoin d'écrire son corps. En revanche, toutes les classes héritant de la classe qui contient cette méthode devront décrire une implémentation de cette méthode.

Pour contrôler les capacités des classes à utiliser les attributs et méthodes les unes des autres, on a accès à trois niveaux d'accessibilité :

- `public`, pour qu'un attribut ou une méthode soit accessible à tous.
- `protected`, pour que les éléments ne soient accessibles qu'aux classes filles.
- Enfin `private`, pour que les éléments ne soient accessibles à personne si ce n'est la classe elle-même.

On trouve par exemple :

```
// Cette classe est accessible à tout le monde
public abstract class Vehicule {
  // Cet attribut est accessible à toutes les filles de la
      classe Vehicule
  protected roue;

  // Personne n'a accès à cette méthode.
```

```
7     abstract private void decelerer();
8   }
```

Enfin, il existe un type de classe mère particulier : les **interfaces**. Une interface est impossible à instancier et toutes les classes filles de cette interface devront instancier les méthodes de cette interface — elles sont toutes forcément `abstract`.

```
1   //Interface des objets qui peuvent voler
2   interface PeutVoler {
3     void décoller();
4   }
5
6   class Avion extends Vehicule implements PeutVoler {
7     //Implémenter toutes les méthodes de PeutVoler et les mé
          thodes abstraites de Vehicule
8   }
```

La compilation et l'exécution

Votre programme est terminé et vous souhaitez le voir fonctionner, c'est tout à fait normal. Cependant, votre programme ne sera pas immédiatement compréhensible par l'ordinateur. En effet, pour qu'un programme fonctionne, il doit d'abord passer par une étape de **compilation**, qui consiste à traduire votre code Java en **bytecode**. Dans le cas d'Android, ce bytecode sera ensuite lu par un logiciel qui s'appelle la **machine virtuelle Dalvik**. Cette machine virtuelle interprète les instructions bytecode et va les traduire en un autre langage que le processeur pourra comprendre, afin de pouvoir exécuter votre programme.

En résumé

- Google n'est pas le seul à l'initiative du projet Android. C'est en 2007 que l'Open Handset Alliance (OHA) a été créé et elle comptait 35 entreprises à ses débuts.
- La philosophie du système réside sur 6 points importants : il fallait qu'il soit open source, gratuit dans la mesure du possible, facile à développer, facile à vendre, flexible et ingénieux.
- Il ne faut jamais perdre à l'esprit que vos smartphones sont (pour l'instant) moins puissants et possèdent moins de mémoire que vos ordinateurs !
- Il existe un certain nombre de bonnes pratiques qu'il faut absolument respecter dans le développement de vos applications. Sans quoi, l'utilisateur aura tendance à vouloir les désinstaller.
  - Ne bloquez jamais le smartphone. N'oubliez pas qu'il fait aussi autre chose lorsque vous exécutez vos applications.
  - Optimisez vos algorithmes : votre smartphone n'est pas comparable à votre ordinateur en terme de performance.
  - Adaptez vos interfaces à tous les types d'écran : les terminaux sont nombreux.

  - Pensez vos interfaces pour les doigts de l'utilisateur final. S'il possède des gros doigts et que vous faites des petits boutons, l'expérience utilisateur en sera altérée.
  - Si possible, testez vos applications sur un large choix de smartphones. Il existe des variations entre les versions, les constructeurs et surtout entre les matériels.
- Une bonne compréhension du langage Java est nécessaire pour suivre ce cours, et plus généralement pour développer sur Android.

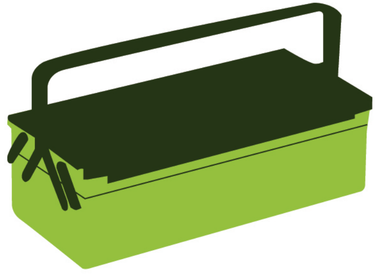
Chapitre 2

Installation et configuration des outils

Difficulté :

Avant de pouvoir entrer dans le vif du sujet, nous allons vérifier que votre ordinateur est capable de supporter la charge du développement pour Android, puis, le cas échéant, on installera tous les programmes et composants nécessaires. Vous aurez besoin de plus de **800 Mo** pour tout installer. Et si vous possédez un appareil sous Android, je vous montrerai comment le configurer de façon à pouvoir travailler directement avec.

Encore un peu de patience, les choses sérieuses démarreront dès le prochain chapitre.



Conditions initiales

De manière générale, n'importe quel matériel permet de développer sur Android du moment que vous utilisez Windows, Mac OS X ou une distribution Linux. Il y a bien sûr certaines limites à ne pas franchir.

Voyons si votre système d'exploitation est suffisant pour vous mettre au travail. Pour un environnement Windows, sont tolérés XP (en version 32 bits), Vista (en version 32 et 64 bits) et 7 (aussi en 32 et 64 bits). Officieusement (en effet, Google n'a rien communiqué à ce sujet), Windows 8 est aussi supporté en 32 et 64 bits.

Et comment savoir quelle version de Windows j'utilise ?

C'est simple, si vous utilisez Windows 7 ou Windows Vista, appuyez en même temps sur la touche `Windows` et sur la touche `R`. Si vous êtes sous Windows XP, il va falloir cliquer sur `Démarrer` puis sur `Exécuter`. Dans la nouvelle fenêtre qui s'ouvre, tapez `winver`. Si la fenêtre qui s'ouvre indique `Windows 7` ou `Windows Vista`, c'est bon, mais s'il est écrit `Windows XP`, alors vous devez vérifier qu'il n'est écrit à aucun moment `64 bits`. Si c'est le cas, alors vous ne pourrez pas développer pour Android.

Sous Mac, il vous faudra Mac OS 10.5.8 ou plus récent **et** un processeur x86.

Sous GNU/Linux, Google conseille d'utiliser une distribution Ubuntu plus récente que la 10.04. Enfin de manière générale, n'importe quelle distribution convient à partir du moment où votre bibliothèque GNU C (`glibc`) est au moins à la version 2.7. Si vous avez une distribution 64 bits, elle devra être capable de lancer des applications 32 bits.

Tout ce que je présenterai sera dans un environnement Windows 7.

Le Java Development Kit

En tant que développeur Java vous avez certainement déjà installé le JDK (pour « Java Development Kit »), cependant on ne sait jamais ! Je vais tout de même vous rappeler comment l'installer. En revanche, si vous l'avez bien installé et que vous êtes à la dernière version, ne perdez pas votre temps et filez directement à la prochaine section !

Un petit rappel technique ne fait de mal à personne. Il existe deux plateformes en Java :

- Le **JRE** (**J**ava **R**untime **E**nvironment), qui contient la **JVM** (**J**ava **V**irtual **M**achine, rappelez-vous, j'ai expliqué le concept de machine virtuelle dans le premier chapitre), les bibliothèques de base du langage ainsi que tous les composants nécessaires au lancement d'applications ou d'applets Java. En gros, c'est l'ensemble d'outils qui

vous permettra d'exécuter des applications Java.
– Le **JDK** (**J**ava **D**evelopment **K**it), qui contient le JRE (afin d'exécuter les applications Java), mais aussi un ensemble d'outils pour compiler et déboguer votre code ! Vous trouverez un peu plus de détails sur la compilation dans l'annexe sur l'architecture d'Android (page 561.

Rendez-vous sur le site d'Oracle grâce au code web suivant[1] et cliquez sur `Download` à côté de `Java SE 6 Update xx` (on va ignorer `Java SE 7` pour le moment) dans la colonne `JDK`, comme à la figure 2.1.

▷ Le site d'Oracle
Code web : 924260

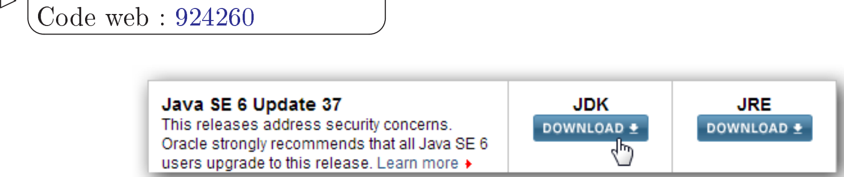


FIGURE 2.1 – On télécharge Java SE 6 et non Java SE 7

On vous demande ensuite d'accepter (`Accept License Agreement`) ou de décliner (`Decline License Agreement`) un contrat de licence, vous devez accepter ce contrat avant de continuer.

Choisissez ensuite la version adaptée à votre configuration. Une fois le téléchargement terminé, vous pouvez installer le tout là où vous le désirez. Vous aurez besoin de **200 Mo** de libre sur le disque ciblé.

Le SDK d'Android

C'est quoi un SDK ?

Un SDK (Software Development Kit, c'est-à-dire un **kit de développement** dans notre langue, est un ensemble d'outils que met à disposition un éditeur afin de vous permettre de développer des applications pour un environnement précis. Le SDK Android permet donc de développer des applications pour Android et uniquement pour Android.

Pour se le procurer, rendez-vous sur la documentation et cliquez sur `USE AN EXISTING IDE` puis sur `Download the SDK Tools`. Au premier lancement du SDK, un écran semblable à la figure 2.2 s'affichera.

▷ La documentation
Code web : 325836

1. Plus d'informations à la page iii.

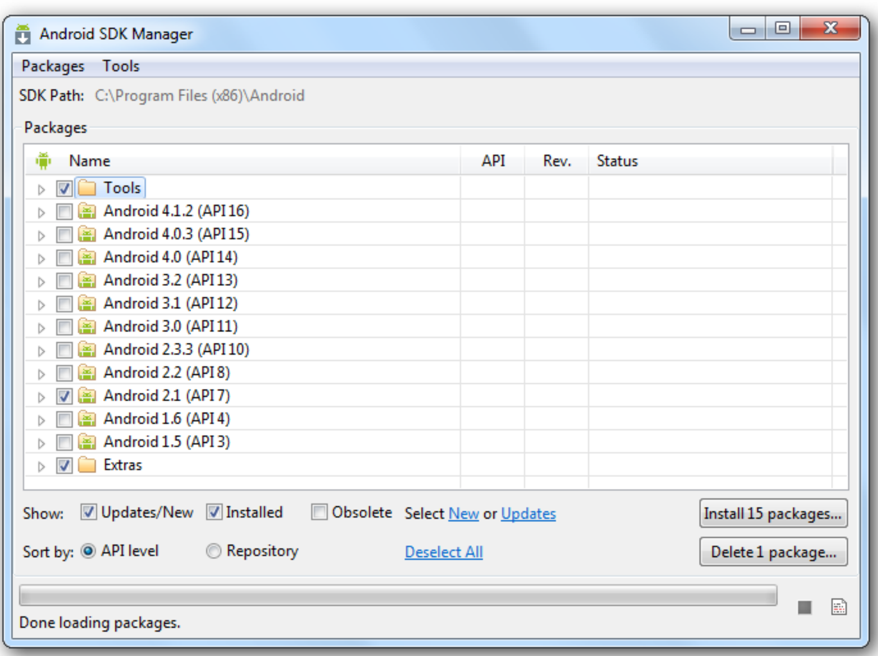


FIGURE 2.2 – L'Android SDK Manager vous permet de choisir les paquets à télécharger

Les trois paquets que je vous demanderai de sélectionner sont `Tools`, `Android 2.1 (API 7)` et `Extras`, mais vous pouvez voir que j'en ai aussi sélectionné d'autres.

À quoi servent les autres paquets ?

Regardez bien le nom des paquets, vous remarquerez qu'ils suivent tous un même motif. Il est écrit à chaque fois `Android [un nombre] (API [un autre nombre])`. La présence de ces nombres s'explique par le fait qu'il existe plusieurs versions de la plateforme Android qui ont été développées depuis ses débuts et qu'il existe donc plusieurs versions différentes en circulation. Le premier nombre correspond à la version d'Android et le second à la version de l'API Android associée. Quand on développe une application, il faut prendre en compte ces numéros, puisqu'une application développée pour une version précise d'Android ne fonctionnera pas pour les versions précédentes. J'ai choisi de délaisser les versions précédant la version 2.1 (l'API 7), de façon à ce que l'application puisse fonctionner pour 2.1, 2.2, 3.1... mais pas forcément pour 1.6 ou 1.5 !

Les API dont le numéro est compris entre 11 et 13 sont théoriquement destinées aux tablettes graphiques. En théorie, vous n'avez pas à vous en soucier, les applications développées avec les API numériquement inférieures fonctionneront, mais il y aura des petits efforts à fournir en revanche en ce qui concerne l'interface graphique (vous trouverez plus de détails dans le chapitre consacré).

Vous penserez peut-être qu'il est injuste de laisser de côté les personnes qui sont contraintes d'utiliser encore ces anciennes versions, mais sachez qu'ils ne représentent que 0,5 % du parc mondial des utilisateurs d'Android. De plus, les changements entre la version 1.6 et la version 2.1 sont trop importants pour être ignorés. Ainsi, toutes les applications que nous développerons fonctionneront sous Android 2.1 minimum. On trouve aussi pour chaque SDK des échantillons de code, `samples`, qui vous seront très utiles pour approfondir ou avoir un second regard à propos de certains aspects, ainsi qu'une API Google associée. Dans un premier temps, vous pouvez ignorer ces API, mais sachez qu'on les utilisera par la suite.

Une fois votre choix effectué, un écran vous demandera de confirmer que vous souhaitez bien télécharger ces éléments-là. Cliquez sur `Accept All` puis sur `Install` pour continuer, comme à la figure 2.3.

Si vous installez tous ces paquets, vous aurez besoin de **1,8 Go** sur le disque de destination. Eh oui, le téléchargement prendra un peu de temps.

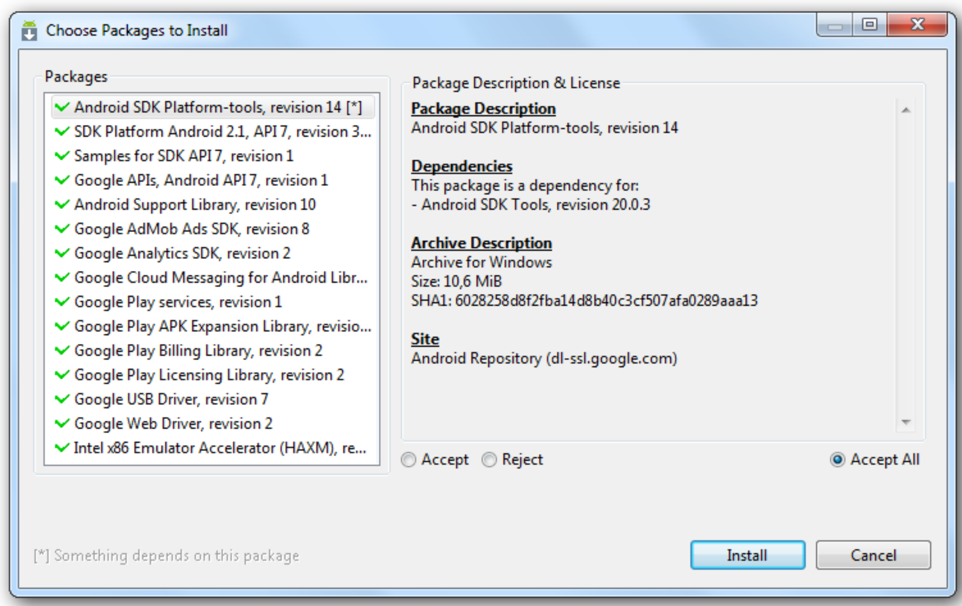


Figure 2.3 – Cliquez sur « Accept All » pour accepter toutes les licences d'un coup

L'IDE Eclipse

Un IDE est un logiciel dont l'objectif est de faciliter le développement, généralement pour un ensemble restreint de langages. Il contient un certain nombre d'outils, dont au moins un éditeur de texte - souvent étendu pour avoir des fonctionnalités avancées telles que l'auto-complétion ou la génération automatique de code - des outils de compilation et un débogueur. Dans le cas du développement Android, un IDE est très pratique pour ceux qui souhaitent ne pas avoir à utiliser les lignes de commande.

J'ai choisi pour ce cours de me baser sur Eclipse : tout simplement parce qu'il est gratuit, puissant et recommandé par Google dans la documentation officielle d'Android. Vous pouvez aussi opter pour d'autres IDE compétents tels que IntelliJ IDEA, NetBeans avec une extension ou encore MoSync.

Le cours reste en majorité valide quel que soit l'IDE que vous sélectionnez, mais vous aurez à explorer vous-mêmes les outils proposés, puisque je ne présenterai ici que ceux d'Eclipse.

Rendez-vous sur le site d'Eclipse pour choisir une version à télécharger. J'ai personnellement opté pour *Eclipse IDE for Java Developers* qui est le meilleur compromis entre contenu suffisant et taille du fichier à télécharger. Les autres versions utilisables sont *Eclipse IDE for Java EE Developers* (je ne vous le recommande pas pour notre cours, il pèse plus lourd et on n'utilisera absolument aucune fonctionnalité de *Java EE*) et *Eclipse Classic* (qui lui aussi intègre des modules que nous n'utiliserons pas).

▷ Télécharger Eclipse
Code web : 543040

Il vous faudra **110 Mo** sur le disque pour installer la version d'Eclipse que j'ai choisie.

Maintenant qu'Eclipse est installé, lancez-le. Au premier démarrage, il vous demandera de définir un `Workspace`, un espace de travail, c'est-à-dire l'endroit où il créera les fichiers indispensables contenant les informations sur les projets. Sélectionnez l'emplacement que vous souhaitez.

Vous avez maintenant un Eclipse prêt à fonctionner... mais pas pour le développement pour Android ! Pour cela, on va télécharger le plug-in (l'extension) *Android Development Tools* (que j'appellerai désormais ADT). Il vous aidera à créer des projets pour Android avec les fichiers de base, mais aussi à tester, à déboguer et à exporter votre projet au format APK (pour pouvoir publier vos applications).

ADT n'est pas le seul add-on qui permette de paramétrer Eclipse pour le développement Android, le MOTODEV Studio For Android est aussi très évolué.

Allez dans `Help` puis dans `Install New Softwares...` (installer de nouveaux programmes). Au premier encart intitulé `Work with:`, cliquez sur le bouton `Add...` qui se situe juste à côté. On va définir où télécharger ce nouveau programme. Dans l'encart `Name` écrivez par exemple `ADT` et, dans `location`, mettez cette adresse `https://dl-ssl.google.com/android/eclipse/`, comme à la figure 2.4. Avec cette adresse, on indique à Eclipse qu'on désire télécharger de nouveaux logiciels qui se trouvent à cet emplacement, afin qu'Eclipse nous propose de les télécharger. Cliquez ensuite sur `OK`.

Si cette manipulation ne fonctionne pas, essayez avec cette adresse `http://dl-ssl.google.com/android/eclipse/` (même chose mais sans le « s » à « http »).

Si vous rencontrez toujours une erreur, alors il va falloir télécharger l'ADT manuellement. Rendez-vous sur la documentation officielle puis cliquez sur le lien qui se trouve dans la colonne `Package` du tableau afin de télécharger une archive qui contient l'ADT, comme à la figure 2.5.

▷ Télécharger manuellement
Code web : 693064

Si le nom n'est pas exactement le même, ce n'est pas grave, il s'agit du même programme mais à une version différente. Ne désarchivez pas le fichier, cela ne vous mènerait à rien.

Une fois le téléchargement terminé, retournez dans la fenêtre que je vous avais demandé d'ouvrir dans Eclipse, puis cliquez sur `Archives`. Sélectionnez le fichier que vous venez de télécharger, entrez un nom dans le champ `Name:` et là seulement cliquez sur `OK`. Le reste est identique à la procédure normale.

Vous devrez patienter tant que sera écrit `Pending...`, puisque c'est ainsi qu'Eclipse indique qu'il cherche les fichiers disponibles à l'emplacement que vous avez précisé. Dès que `Developer Tools` apparaît à la place de `Pending...`, développez le menu en cliquant sur le triangle à gauche du carré de sélection et analysons les éléments proposés, comme sur la figure 2.6.

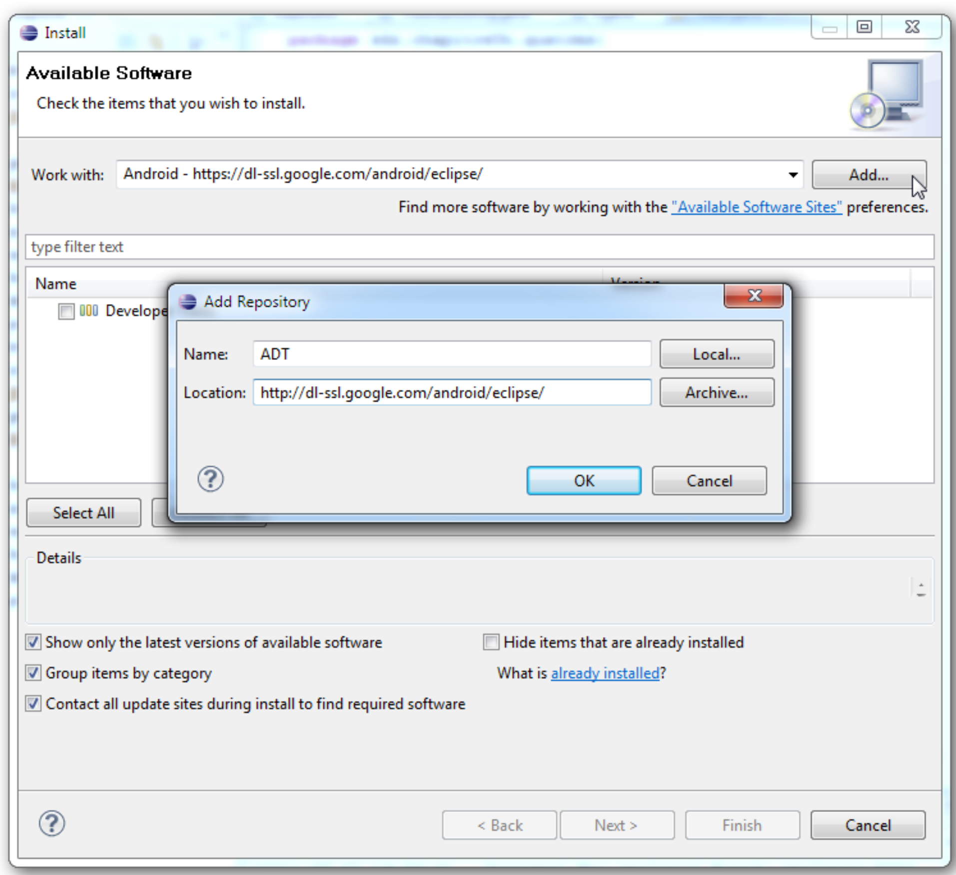


FIGURE 2.4 – On ajoute un répertoire distant d'où seront téléchargés les sources de l'ADT

Package	Size	MD5 Checksum
ADT-20.0.3.zip	12390954 bytes	869a536b1c56d0cd920ed9ae259ae619

FIGURE 2.5 – Téléchargez l'archive

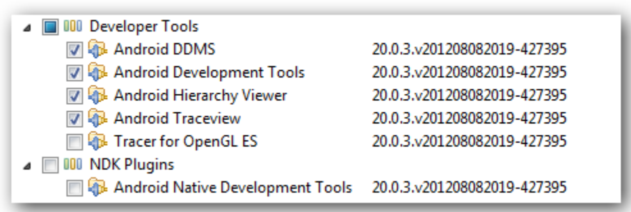


FIGURE 2.6 – Il nous faut télécharger au moins ces modules

- *Android DDMS* est l'*Android Dalvik Debug Monitor Server*, il permet d'exécuter quelques fonctions pour vous aider à déboguer votre application (simuler un appel ou une position géographique par exemple) et d'avoir accès à d'autres informations utiles.
- L'*ADT*.
- *Android Hierarchy Viewer*, qui permet d'optimiser et de déboguer son interface graphique.
- *Android Traceview*, qui permet d'optimiser et de déboguer son application.

Il existe d'autres modules que nous n'utiliserons pas pendant ce cours :

- *Tracer for OpenGL ES*, qui permet de déboguer des applications OpenGL ES.
- *Android Native Development Tools* est utilisé pour développer des applications Android en C++, mais ce cours est axé sur Java.

Sélectionnez tout et cliquez sur `Next`, à nouveau sur `Next` à l'écran suivant puis finalement sur « I accept the terms of the license agreements » après avoir lu les différents contrats. Cliquez enfin sur `Finish`.

L'ordinateur téléchargera puis installera les composants. Une fenêtre s'affichera pour vous dire qu'il n'arrive pas à savoir d'où viennent les programmes téléchargés et par conséquent qu'il n'est pas sûr qu'ils soient fonctionnels et qu'ils ne soient pas dangereux. Cependant, nous savons qu'ils sont sûrs et fonctionnels, alors cliquez sur `OK`.

Une fois l'installation et le téléchargement terminés, il vous proposera de redémarrer l'application. C'est presque fini, mais il nous reste quand même une dernière étape à accomplir.

L'émulateur de téléphone : Android Virtual Device

L'*Android Virtual Device*, aussi appelé AVD, est un émulateur de terminal sous Android, c'est-à-dire que c'est un logiciel qui fait croire à votre ordinateur qu'il est un appareil sous Android. C'est la raison pour laquelle vous n'avez pas besoin d'un périphérique sous Android pour développer et tester la plupart de vos applications ! En effet, une application qui affiche un calendrier par exemple peut très bien se tester dans un émulateur, mais une application qui exploite le GPS doit être éprouvée sur le terrain pour que l'on soit certain de son comportement.

Lancez à nouveau Eclipse si vous l'avez fermé. Au cas où vous auriez encore l'écran d'accueil, cliquez sur la croix en haut à gauche pour le fermer. Repérez tout d'abord où se trouve la barre d'outils, visible à la figure 2.7.

FIGURE 2.7 – La barre d'outils d'Eclipse

Vous voyez le couple d'icônes représenté à la figure 2.8 ? Celle de gauche permet d'ouvrir les outils du SDK et celle de droite permet d'ouvrir l'interface de gestion d'AVD. Cliquez dessus puis sur `New...` pour ajouter un nouvel AVD.

FIGURE 2.8 – Les deux icônes réservées au SDK et à l'AVD

Une fois sur deux, Eclipse me dit que je n'ai pas défini l'emplacement du SDK (« Location of the Android SDK has not been setup in the preferences »). S'il vous le dit aussi, c'est que soit vous ne l'avez vraiment pas fait, auquel cas vous devrez faire l'opération indiquée dans la section précédente, soit il se peut aussi qu'Eclipse pipote un peu, auquel cas réappuyez sur le bouton jusqu'à ce qu'il abdique.

Une fenêtre s'ouvre (voir figure 2.9), vous proposant de créer votre propre émulateur ! Bien que ce soit facultatif, je vous conseille d'indiquer un nom dans `Name`, histoire de pouvoir différencier vos AVD. Pour ma part, j'ai choisi « `Site_Du_Zero_2_1` ». Notez que certains caractères comme les caractères accentués et les espaces ne sont pas autorisés. Dans `Target`, choisissez `Android 2.1 - API Level 7`, puisque j'ai décidé que nous ferons nos applications avec la version 7 de l'API et sans le Google API. Laissez les autres options à leur valeur par défaut, nous y reviendrons plus tard quand nous confectionnerons d'autres AVD. Cliquez enfin sur `Create AVD` et vous aurez une machine prête à l'emploi !

Si vous utilisez Windows et que votre nom de session contient un caractère spécial, par exemple un accent, alors Eclipse vous enverra paître en déclarant qu'il ne trouve pas le fichier de configuration de l'AVD. Par exemple, un de nos lecteur avait une session qui s'appelait « Jérémie » et avait ce problème. Heureusement, il existe une solution à ce problème. Si vous utilisez Windows 7 ou Windows Vista, appuyez en même temps sur la touche `Windows` et sur la touche `R`. Si vous êtes sous Windows XP, il va falloir cliquer sur `Démarrer` puis sur `Exécuter`.

Dans la nouvelle fenêtre qui s'ouvre, tapez « cmd » puis appuyez sur la touche `Entrée` de votre clavier. Une nouvelle fenêtre va s'ouvrir, elle permet de manipuler Windows en ligne de commande. Tapez `cd ..` puis `Entrée`. Maintenant, tapez `dir /x`. Cette commande permet de lister tous les répertoires et fichiers présents dans le répertoire

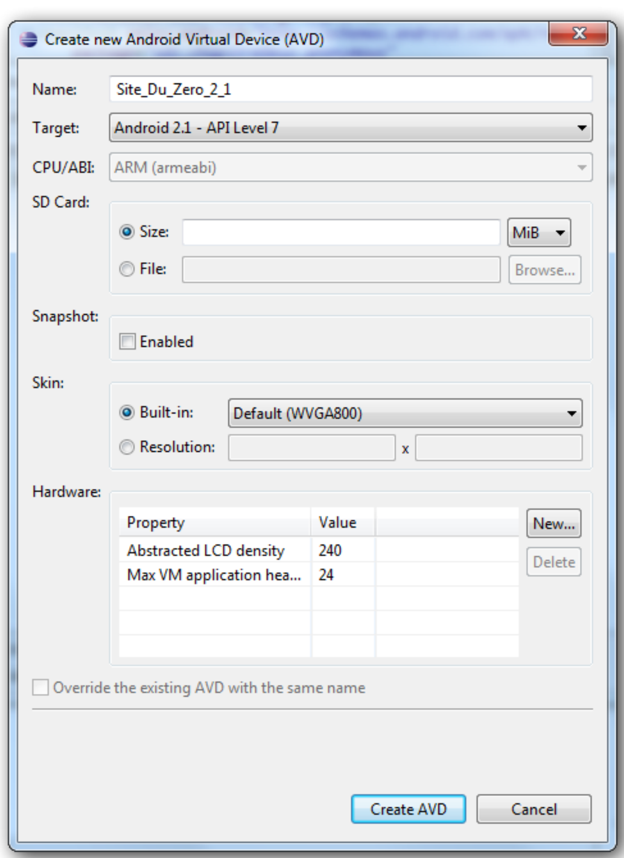


FIGURE 2.9 – Créez votre propre émulateur

actuel et aussi d'afficher le nom abrégé de chaque fichier ou répertoire. Par exemple, pour la session Administrator on obtient le nom abrégé ADMINI~1, comme le montre la figure 2.10.

07/04/2011 19:07 <REP> ADMINI~1 Administrator

FIGURE 2.10 – La valeur à gauche est le nom réduit, alors que celle de droite est le nom entier

Maintenant, repérez le nom réduit qui correspond à votre propre session, puis dirigez-vous vers le fichier X:\Utilisateurs\<Votre session>\.android\avd\<nom_de_votre_avd>.ini et ouvrez ce fichier. Il devrait ressembler au code suivant :

```
target=android-7
path=X:\Users\<Votre session>\.android\avd\SDZ_2.1.avd
```

S'il n'y a pas de retour à la ligne entre target=android-7 et path=X:\Users\<Votre session>\.android\avd\SDZ_2.1.avd, c'est que vous n'utilisez pas un bon éditeur de texte. Utilisez le lien que j'ai donné ci-dessus. Enfin, il vous suffit de remplacer <Votre session> par le nom abrégé de la session que nous avions trouvé précédemment. Par exemple pour le cas de la session Administrator, je change :

```
target=android-7
path=C:\Users\Administrator\.android\avd\SDZ_2.1.avd
```

en

```
target=android-7
path=C:\Users\ADMINI~1\.android\avd\SDZ_2.1.avd
```

Test et configuration

Bien, maintenant que vous avez créé un AVD, on va pouvoir vérifier qu'il fonctionne bien.

Si vous êtes sortis du gestionnaire Android, retournez-y en cliquant sur l'icône Bugdroid, comme nous l'avons fait auparavant. Vous aurez quelque chose de plus ou moins similaire à la figure 2.11.

Vous y voyez l'AVD que nous venons tout juste de créer. Cliquez dessus pour déverrouiller le menu de droite. Comme je n'ai pas l'intention de vraiment détailler ces options moi-même, je vais rapidement vous expliquer à quoi elles correspondent pour que vous sachiez les utiliser en cas de besoin. Les options du menu de droite sont les suivantes :

- Edit... vous permet de changer les caractéristiques de l'AVD sélectionné.
- Delete... vous permet de supprimer l'AVD sélectionné.
- Repair... ne vous sera peut-être jamais d'aucune utilité, il vous permet de réparer un AVD quand le gestionnaire vous indique qu'il faut le faire.

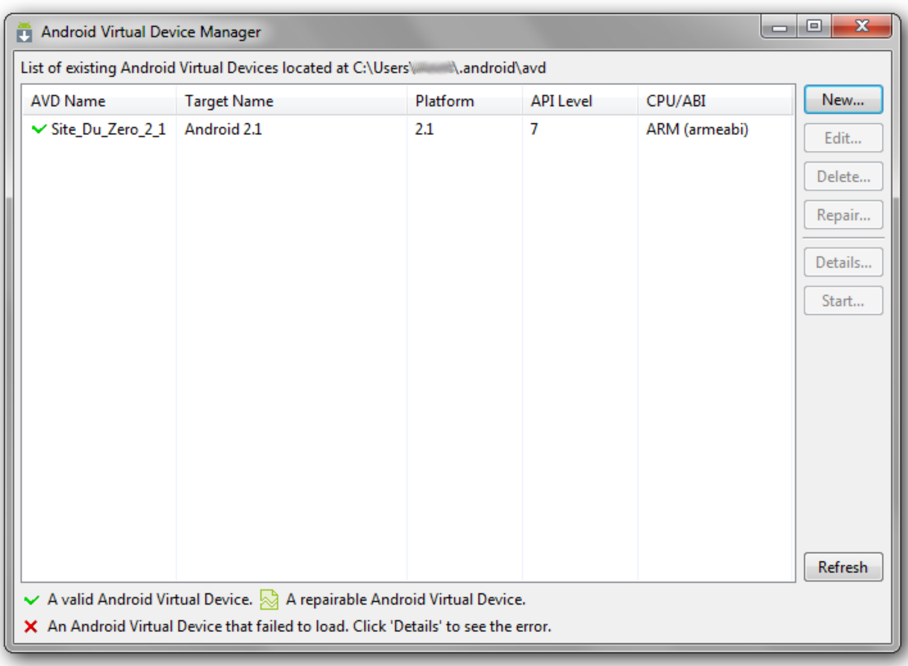


FIGURE 2.11 – La liste des émulateurs que connaît votre AVD Manager

- `Details...` lancera une nouvelle fenêtre qui listera les caractéristiques de l'AVD sélectionné.
- `Start...` est le bouton qui nous intéresse maintenant, il vous permet de lancer l'AVD.

Cliquons donc sur le bouton `Start...` et une nouvelle fenêtre se lance, qui devrait ressembler peu ou prou à la figure 2.12.

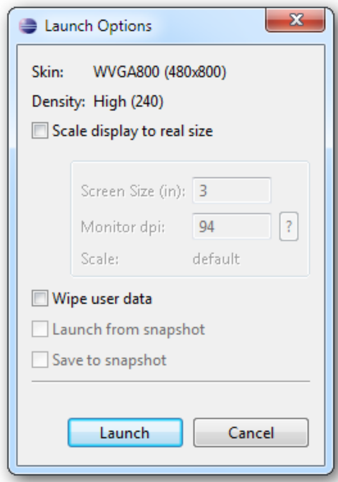


FIGURE 2.12 – Les différentes options pour l'exécution de cet AVD

Laissez les options vierges pour l'instant, on n'a absolument pas besoin de ce genre de détails ! Cliquez juste sur `Launch`. En théorie, une nouvelle fenêtre se lancera et passera par deux écrans de chargement successifs. Enfin, votre terminal se lancera. Voici la liste des boutons qui se trouvent dans le menu à droite et à quoi ils servent :

Prendre une photo

Diminuer le volume de la sonnerie ou de la musique

Augmenter le volume de la sonnerie ou de la musique

Arrêter l'émulateur

Décrocher le téléphone

Raccrocher le téléphone

Retourner sur le dashboard (l'équivalent du bureau, avec les icônes et les widgets)

Ouvrir le menu

Retour arrière

Effectuer une recherche

Mais ! L'émulateur n'est pas à l'heure ! En plus c'est de l'anglais !

La maîtrise de l'anglais devient vite indispensable dans le monde de l'informatique... Ensuite, les machines que vous achetez dans le commerce sont déjà configurées pour le pays dans lequel vous les avez acquises, et, comme ce n'est pas une machine réelle ici, Android a juste choisi les options par défaut. Nous allons devoir configurer la machine pour qu'elle réponde à nos exigences. Vous pouvez manipuler la partie de gauche avec votre souris, ce qui simulera le tactile. Faites glisser le verrou sur la gauche pour déverrouiller la machine. Vous vous retrouverez sur l'accueil. Cliquez sur le bouton `MENU` à droite pour ouvrir un petit menu en bas de l'écran de l'émulateur, comme à la figure 2.13.

Cliquez sur l'option `Settings` pour ouvrir le menu de configuration d'Android. Vous pouvez y naviguer soit en faisant glisser avec la souris (un clic, puis en laissant appuyé on dirige le curseur vers le haut ou vers le bas), soit avec la molette de votre souris. Si par mégarde vous entrez dans un menu non désiré, appuyez sur le bouton `Retour` présenté précédemment (une flèche qui effectue un demi-tour).

Cliquez sur l'option `Language & keyboard` (voir figure 2.14) ; c'est le menu qui vous permet de choisir dans quelle langue utiliser le terminal et quel type de clavier utiliser (par exemple, vous avez certainement un clavier dont les premières lettres forment le mot `AZERTY`, c'est ce qu'on s'appelle un clavier AZERTY. Oui, oui, les informaticiens

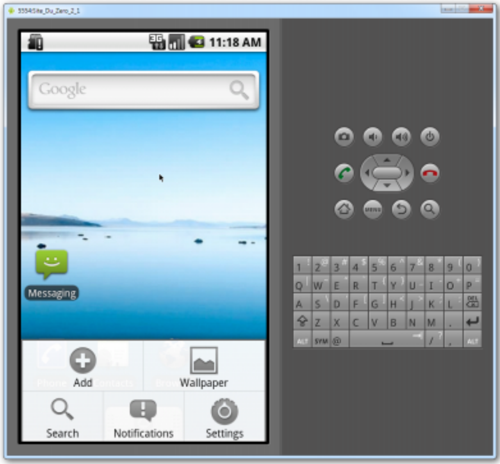


Figure 2.13 – Le menu est ouvert

ont beaucoup d'imagination).

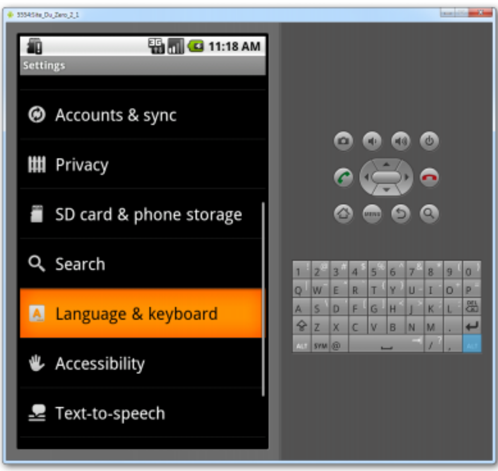


Figure 2.14 – On va sélectionner « Language & keyboard »

Puis, vous allez cliquer sur `Select locale`. Dans le prochain menu, il vous suffit de sélectionner la langue dans laquelle vous préférez utiliser Android. J'ai personnellement choisi `Français (France)`. Voilà, un problème de réglé ! Maintenant j'utiliserai les noms français des menus pour vous orienter. Pour revenir en arrière, il faut appuyer sur le bouton `Retour` du menu de droite.

Votre prochaine mission, si vous l'acceptez, sera de changer l'heure pour qu'elle s'adapte à la zone dans laquelle vous vous trouvez, et ce, par vous-mêmes. En France, nous vivons dans la zone GMT + 1. À l'heure où j'écris ces lignes, nous sommes en heure d'été, il

y a donc une heure encore à rajouter. Ainsi, si vous êtes en France, en Belgique ou au Luxembourg et en heure d'été, vous devez sélectionner une zone à GMT + 2. Sinon GMT + 1 pour l'heure d'hiver. Cliquez d'abord sur `Date & heure`, désélectionnez `Automatique`, puis cliquez sur `Définir fuseau horaire` et sélectionnez le fuseau qui vous concerne.

Très bien, votre terminal est presque complètement configuré, nous allons juste activer les options pour le rendre apte à la programmation. Toujours dans le menu de configuration, allez chercher `Applications` et cliquez dessus. Cliquez ensuite sur `Développement` et vérifiez que tout est bien activé comme à la figure 2.15.

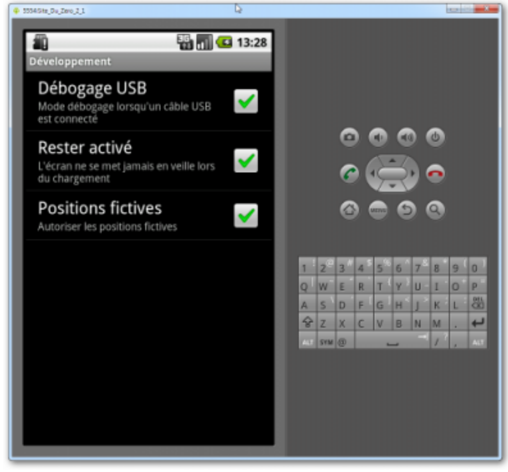


FIGURE 2.15 – Ce menu vous permet de développer pour Android

Vous l'aurez remarqué par vous-mêmes, la machine est lourde à utiliser, voire très lourde sur les machines les plus modestes ; autant dire tout de suite que c'est beaucoup moins confortable à manipuler qu'un vrai terminal sous Android.

Si vous comptez faire immédiatement le prochain chapitre qui vous permettra de commencer — enfin — le développement, ne quittez pas la machine. Dans le cas contraire, il vous suffit de rester appuyé sur le bouton pour arrêter l'émulateur puis de vous laisser guider.

Configuration du vrai terminal

Maintenant on va s'occuper de notre vrai outil, si vous en avez un !

Configuration du terminal

Tout naturellement, vous devez configurer votre téléphone comme on a configuré l'émulateur. En plus, vous devez indiquer que vous acceptez les applications qui ne proviennent pas du Market dans `Configuration > Application > Source inconnue`.

Pour les utilisateurs de Windows

Tout d'abord, vous devez télécharger les drivers adaptés à votre terminal. Je peux vous donner la marche à suivre pour certains terminaux, mais pas pour tous... En effet, chaque appareil a besoin de drivers adaptés, et ce sera donc à vous de les télécharger, souvent sur le site du constructeur. Cependant, il existe des pilotes génériques qui peuvent fonctionner sur certains appareils. En suivant ma démarche, ils sont déjà téléchargés, mais rien n'assure qu'ils fonctionnent pour votre appareil. En partant du répertoire où vous avez installé le SDK, on peut les trouver à cet emplacement : `\android-sdk\extras\google\usb_driver`. Vous trouverez l'emplacement des pilotes à télécharger pour toutes les marques dans le tableau qui se trouve sur la documentation.

▷ Emplacement des pilotes
Code web : 125950

Pour les utilisateurs de Mac

À la bonne heure, vous n'avez absolument rien à faire de spécial pour que tout fonctionne !

Pour les utilisateurs de Linux

La gestion des drivers USB de Linux étant beaucoup moins chaotique que celle de Windows, vous n'avez pas à télécharger de drivers. Il y a cependant une petite démarche à accomplir. On va en effet devoir ajouter au gestionnaire de périphériques une règle spécifique pour chaque appareil qu'on voudra relier. Je vais vous décrire cette démarche pour les utilisateurs d'Ubuntu :

1. On va d'abord créer le fichier qui contiendra ces règles à l'aide de la commande `sudo touch /etc/udev/rules.d/51-android.rules`. `touch` est la commande qui permet de créer un fichier, et `udev` est l'emplacement des fichiers du gestionnaire de périphériques. `udev` conserve ses règles dans le répertoire `./rules.d`.
2. Le système vous demandera de vous identifier en tant qu'utilisateur `root`.
3. Puis on va modifier les autorisations sur le fichier afin d'autoriser la lecture et l'écriture à tous les utilisateurs `chmod a+rw /etc/udev/rules.d/51-android.rules`.
4. Enfin, il faut rajouter les règles dans notre fichier nouvellement créé. Pour cela, on va ajouter une instruction qui ressemblera à : `SUBSYSTEM=="usb", ATTR{idVendor}`

`=="XXXX", MODE="0666", GROUP="plugdev"`. Attention, on n'écrira pas *exactement* cette phrase.

Est-il possible d'avoir une explication ?

`SUBSYSTEM` est le mode de connexion entre le périphérique et votre ordinateur, dans notre cas on utilisera une interface USB. `MODE` détermine qui peut faire quoi sur votre périphérique, et la valeur « 0666 » indique que tous les utilisateurs pourront lire des informations mais aussi en écrire. `GROUP` décrit tout simplement quel groupe UNIX possède le périphérique. Enfin, `ATTR{idVendor` est la ligne qu'il vous faudra modifier en fonction du constructeur de votre périphérique. On peut trouver quelle valeur indiquer sur la documentation.

▷ Voir les valeurs
Code web : 572703

Par exemple pour mon HTC Desire, j'indique la ligne suivante :

```
SUBSYSTEM=="usb", ATTR{idVendor}=="0bb4", MODE="0666", GROUP="
   plugdev"
```

. . . ce qui entraîne que je tape dans la console :

```
echo "SUBSYSTEM==\"usb\", ATTR{idVendor}==\"0bb4\", MODE
   =\"0666\", GROUP=\"plugdev\"" >> /etc/udev/rules.d/51-android
   .rules
```

Si cette configuration ne vous correspond pas, je vous invite à lire la documentation afin de créer votre propre règle.

▷ Créer vos règles
Code web : 873298

Et après ?

Ben rien ! La magie de l'informatique opère, reliez votre terminal à l'ordinateur et tout devrait se faire de manière automatique (tout du moins sous Windows 7, désolé pour les autres !).

En résumé

– Il est essentiel d'installer l'environnement Java sur votre ordinateur pour pouvoir développer vos applications Android.

- Vous devez également installer le SDK d'Android pour pouvoir développer vos applications. Ce kit de développement vous offrira, entre autres, les outils pour télécharger les paquets de la version d'Android pour lequel vous voulez développer.
- Eclipse n'est pas l'environnement de travail obligatoire pour développer vos applications mais c'est une recommandation de Google pour sa gratuité et sa puissance. De plus, le SDK d'Android est prévu pour s'y intégrer et les codes sources de ce cours seront développés grâce à cet IDE.
- Si vous n'avez pas de smartphone Android, Google a pensé à vous et mis à votre disposition des AVD pour tester vos applications. Ces machines virtuelles lancent un véritable système Android mais prenez garde à ne pas vous y fier à 100%, il n'y a rien de plus concret que les tests sur des terminaux physiques.

Chapitre 3

Votre première application

Difficulté :

Ce chapitre est très important. Il vous permettra d'enfin mettre la main à la pâte, mais surtout on abordera la notion de cycle d'une activité, qui est la base d'un programme pour Android. Si pour vous un programme en Java débute forcément par un `main`, vous risquez d'être surpris. :-°

On va tout d'abord voir ce qu'on appelle des activités et comment les manipuler. Sachant que la majorité de vos applications (si ce n'est toutes) contiendront plusieurs activités, il est indispensable que vous maîtrisiez ce concept ! Nous verrons aussi ce que sont les vues et nous créerons enfin notre premier projet — le premier d'une grande série — qui n'est pas, de manière assez surprenante, un « Hello World ! ». Enfin presque !

Activité et vue

Qu'est-ce qu'une activité ?

Si vous observez un peu l'architecture de la majorité des applications Android, vous remarquerez une construction toujours à peu près similaire. Prenons par exemple l'application du Play Store. Vous avez plusieurs fenêtres à l'intérieur même de cette application : si vous effectuez une recherche, une liste de résultats s'affichera dans une première fenêtre et si vous cliquez sur un résultat, une nouvelle fenêtre s'ouvre pour vous afficher la page de présentation de l'application sélectionnée. Au final, on remarque qu'une application est un assemblage de fenêtres entre lesquelles il est possible de naviguer.

Ces différentes fenêtres sont appelées des activités. Un moyen efficace de différencier des activités est de comparer leur interface graphique : si elles sont radicalement différentes, c'est qu'il s'agit d'activités différentes. De plus, comme une activité remplit tout l'écran, votre application ne peut en afficher qu'une à la fois. La figure 3.1 illustre ce concept.

FIGURE 3.1 – Cliquer sur un élément de la liste dans la première activité permet d'ouvrir les détails dans une seconde activité

Je me permets de faire un petit aparté pour vous rappeler ce qu'est une interface graphique : il s'agit d'un ensemble d'éléments visuels avec lesquels peuvent interagir les utilisateurs, ou qui leur fournissent des informations. Tout ça pour vous dire qu'une activité est un support sur lequel nous allons greffer une interface graphique. Cependant, ce n'est pas le rôle de l'activité que de créer et de disposer les éléments graphiques, elle n'est que l'échafaudage sur lequel vont s'insérer les objets graphiques.

De plus, une activité contient des informations sur l'état actuel de l'application : ces

informations s'appellent le `context`. Ce `context` constitue un lien avec le système Android ainsi que les autres activités de l'application, comme le montre la figure 3.2.

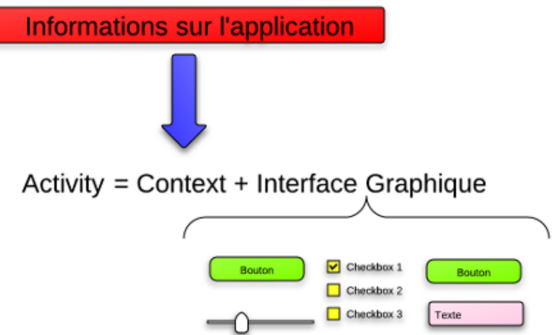


FIGURE 3.2 – Une activité est constituée du contexte de l'application et d'une seule et unique interface graphique

Comme il est plus aisé de comprendre à l'aide d'exemples, imaginez que vous naviguiez sur le Site du Zéro avec votre téléphone, le tout en écoutant de la musique sur ce même téléphone. Il se passe deux choses dans votre système :

– La navigation sur internet, permise par une interface graphique (la barre d'adresse et le contenu de la page web, au moins) ;
– La musique, qui est diffusée en fond sonore, mais qui n'affiche pas d'interface graphique à l'heure actuelle puisque l'utilisateur consulte le navigateur.

On a ainsi au moins deux applications lancées en même temps ; cependant, le navigateur affiche une activité alors que le lecteur audio n'en affiche pas.

États d'une activité

Si un utilisateur reçoit un appel, il devient plus important qu'il puisse y répondre que d'émettre la chanson que votre application diffuse. Pour pouvoir toujours répondre à ce besoin, les développeurs d'Android ont eu recours à un système particulier :

– À tout moment votre application peut laisser place à une autre application, qui a une priorité plus élevée. Si votre application utilise trop de ressources système, alors elle empêchera le système de fonctionner correctement et Android l'arrêtera sans vergogne.
– Votre activité existera dans plusieurs états au cours de sa vie, par exemple un état actif pendant lequel l'utilisateur l'exploite, et un état de pause quand l'utilisateur reçoit un appel.

Pour être plus précis, quand une application se lance, elle se met tout en haut de ce qu'on appelle la pile d'activités.

Une pile est une structure de données de type « LIFO », c'est-à-dire qu'il n'est possible d'avoir accès qu'à un seul élément de la pile, le tout premier élément, aussi appelé **sommet**. Quand on ajoute un élément à cette pile, le nouvel élément prendra la première place et deviendra le nouveau sommet. Quand on veut récupérer un élément, ce sera le sommet qui sera récupéré, sorti de la liste et l'objet en deuxième place deviendra le nouveau sommet, comme illustré à la figure 3.3.

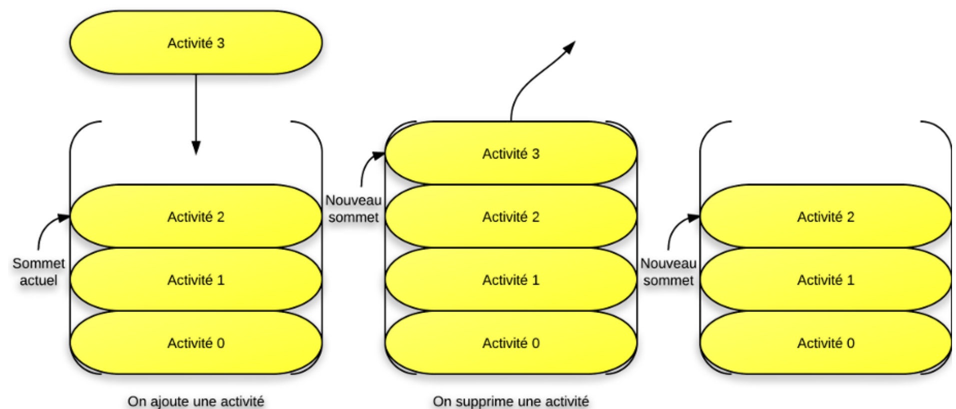


FIGURE 3.3 – Fonctionnement de la pile d'activités

L'activité que voit l'utilisateur est celle qui se trouve au-dessus de la pile. Ainsi, lorsqu'un appel arrive, il se place au sommet de la pile et c'est lui qui s'affiche à la place de votre application, qui n'est plus qu'à la deuxième place. Votre activité ne reviendra qu'à partir du moment où toutes les activités qui se trouvent au-dessus d'elle seront arrêtées et sorties de la pile. On retrouve ainsi le principe expliqué précédemment, on ne peut avoir qu'une application visible en même temps sur le terminal, et ce qui est visible est l'interface graphique de l'activité qui se trouve au sommet de la pile.

Une activité peut se trouver dans trois états qui se différencient surtout par leur visibilité.

Active (`active` ou `running`)

L'activité est visible en totalité. Elle est sur le dessus de la pile, c'est ce que l'utilisateur consulte en ce moment même et il peut l'utiliser dans son intégralité. C'est cette application qui a le *focus*, c'est-à-dire que l'utilisateur agit directement sur l'application.

Suspendue (`paused`)

L'activité est partiellement visible à l'écran. C'est le cas quand vous recevez un SMS et qu'une fenêtre semi-transparente se pose devant votre activité pour afficher le contenu du message et vous permettre d'y répondre par exemple.

Ce n'est pas sur cette activité qu'agit l'utilisateur. L'application n'a plus le focus, c'est l'application sus-jacente qui l'a. Pour que notre application récupère le focus, l'utilisateur devra se débarrasser de l'application qui l'obstrue, puis l'utilisateur pourra à nouveau interagir avec.

Si le système a besoin de mémoire, il peut très bien tuer l'application (cette affirmation n'est plus vraie si vous utilisez un SDK avec l'API 11 minimum).

Arrêtée (`stopped`)

L'activité est tout simplement oblitérée par une autre activité, on ne peut plus la voir du tout. L'application n'a évidemment plus le focus, et puisque l'utilisateur ne peut pas la voir, il ne peut pas agir dessus. Le système retient son état pour pouvoir reprendre, mais il peut arriver que le système tue votre application pour libérer de la mémoire système.

Mais j'ai pourtant déjà vu des systèmes Android avec deux applications visibles en même temps !

Ah oui, c'est possible. Mais il s'agit d'un artifice, il n'y a vraiment qu'une application qui est active. Pour faciliter votre compréhension, je vous conseille d'oublier ces systèmes.

Cycle de vie d'une activité

Une activité n'a pas de contrôle direct sur son propre état (et par conséquent vous non plus en tant que programmeur), il s'agit plutôt d'un cycle rythmé par les interactions avec le système et d'autres applications. La figure 13.2 est un schéma qui présente ce que l'on appelle **le cycle de vie d'une activité**, c'est-à-dire qu'il indique les étapes que va traverser notre activité pendant sa vie, de sa naissance à sa mort. Vous verrez que chaque étape du cycle est représentée par une méthode. Nous verrons comment utiliser ces méthodes en temps voulu.

Les activités héritent de la classe `Activity`. Or, la classe `Activity` hérite de l'interface `Context` dont le but est de représenter tous les composants d'une application. On les trouve dans le package `android.app.Activity`.

Pour rappel, un package est un répertoire qui permet d'organiser notre code source, un récipient dans lequel nous allons mettre nos classes de façon à pouvoir trier votre code et différencier des classes qui auraient le même nom. Concrètement, supposez que vous ayez à créer deux classes `X` — qui auraient deux utilisations différentes, bien sûr. Vous vous rendez bien compte que vous seriez dans l'incapacité totale de différencier les deux classes si vous deviez instancier un objet de l'une des deux classes `X`, et Java vous houspillera en déclarant qu'il ne peut pas savoir à quelle classe vous faites référence. C'est exactement comme avoir deux fichiers avec le même nom et la même extension

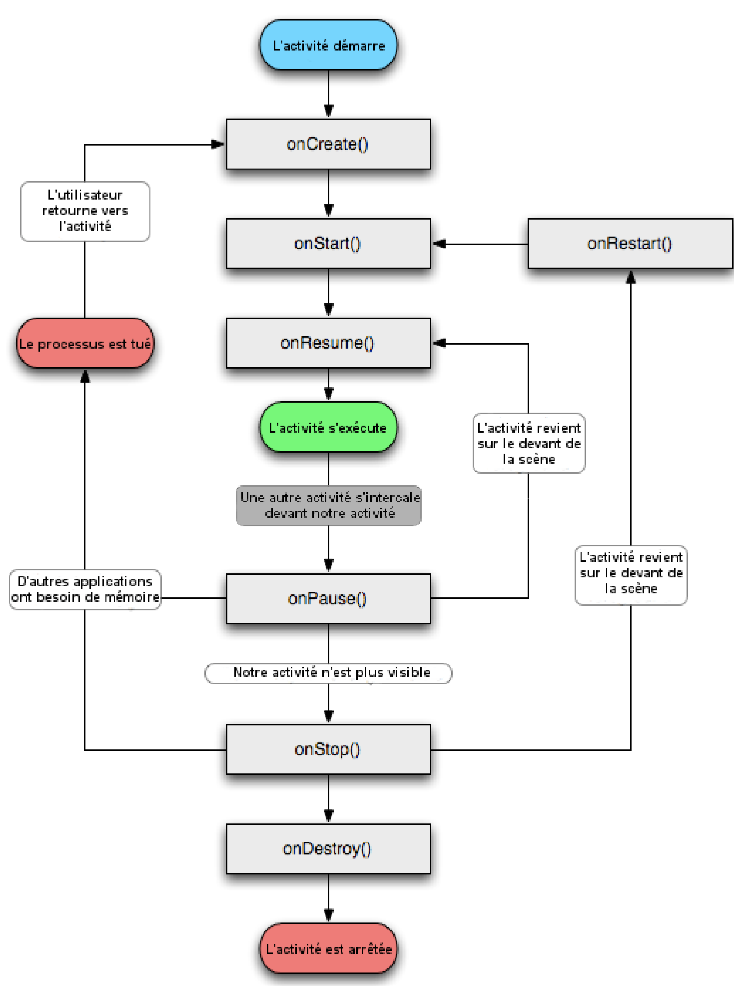


FIGURE 3.4 – Cycle de vie d'une activité

dans un même répertoire : c'est impossible car c'est incohérent.

Pour contrer ce type de désagrément, on organise les classes à l'aide d'une hiérarchie. Si je reprends mon exemple des deux classes `X`, je peux les placer dans deux packages différents `Y` et `Z` par exemple, de façon à ce que vous puissiez préciser dans quel package se trouve la classe `X` sollicitée. On utilisera la syntaxe `Y.X` pour la classe `X` qui se trouve dans le package `Y` et `Z.X` pour la classe `X` qui se trouve dans le package `Z`. Dans le cas un peu farfelu du code source d'un navigateur internet, on pourrait trouver les packages `Web.Affichage.Image`, `Web.Affichage.Video` et `Web.Telechargement`.

Les **vues** (que nos amis anglais appellent ***view***), sont ces fameux composants qui viendront se greffer sur notre échafaudage, il s'agit de l'unité de base de l'interface graphique. Leur rôle est de fournir du contenu visuel avec lequel il est éventuellement possible d'interagir. À l'instar de l'interface graphique en Java, il est possible de disposer les vues à l'aide de conteneurs, nous verrons comment plus tard.

Les vues héritent de la classe `View`. On les trouve dans le package `android.view.View`.

Création d'un projet

Une fois Eclipse démarré, repérez les icônes visibles à la figure 3.5 et cliquez sur le bouton le plus à gauche de la section consacrée à la gestion de projets Android.

FIGURE 3.5 – Ces trois boutons permettent de gérer des projets Android

La fenêtre visible à la figure 3.6 s'ouvre ; voyons ensemble ce qu'elle contient :

Tous ces champs nous permettent de définir certaines caractéristiques de notre projet :

- Tout d'abord, vous pouvez choisir le nom de votre application avec `Application name`. Il s'agit du nom qui apparaîtra sur l'appareil et sur Google Play pour vos futures applications ! Choisissez donc un nom qui semble à la fois judicieux, assez original pour attirer l'attention et qui reste politiquement correct au demeurant.
- `Project name` est le nom de votre projet pour Eclipse. Ce champ n'influence pas l'application en elle-même, il s'agit juste du nom sous lequel Eclipse la connaîtra. Le vrai nom de notre application, celui que reconnaîtra Android et qui a été défini dans `Application name`, peut très bien n'avoir aucune similitude avec ce que vous mettrez dans ce champ.
- Il faudra ensuite choisir dans quel package ira votre application, je vous ai déjà expliqué l'importance des packages précédemment. Sachez que ce package agira comme une sorte d'identifiant pour votre application sur le marché d'applications, alors faites en sorte qu'il soit unique et constant pendant tout le développement de votre

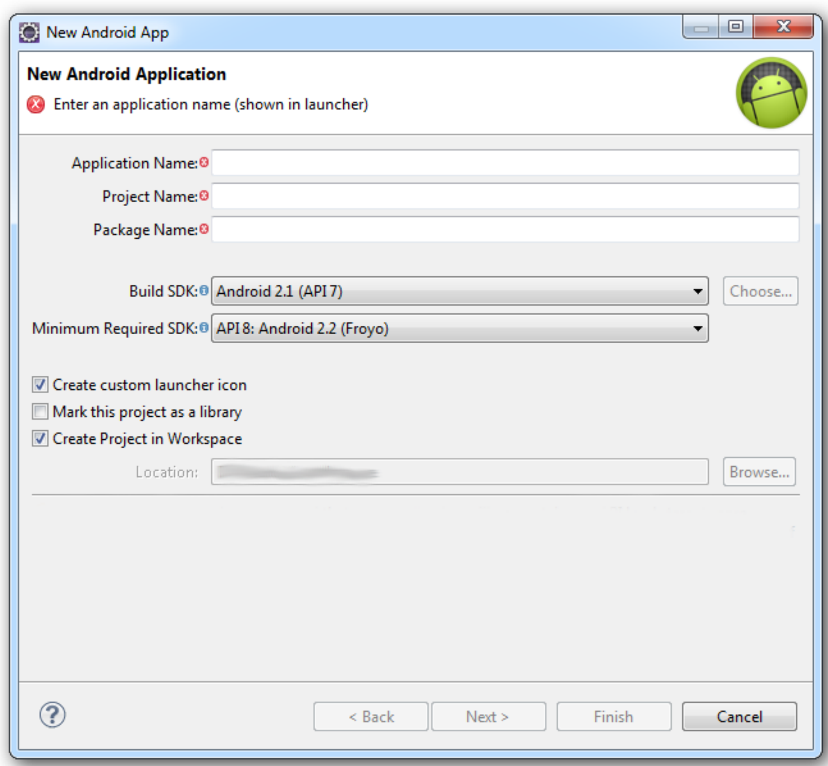


FIGURE 3.6 – Création d'un nouveau projet

application.

Ces trois champs sont indispensables, vous devrez donc tous les renseigner.

Vous vous retrouvez ensuite confronté à deux listes défilantes :

- La liste `Build SDK` vous permet de choisir pour quelle version du SDK vous allez compiler votre application. Comme indiqué précédemment, on va choisir l'API 7.
- La liste suivante, `Minimum Required SDK`, est un peu plus subtile. Elle vous permet de définir à partir de quelle version d'Android votre application sera visible sur le marché d'applications. Ce n'est pas parce que vous compilez votre application pour l'API 7 que vous souhaitez que votre application fonctionne sous les téléphones qui utilisent Android 2.1, vous pouvez très bien viser les téléphones qui exploitent des systèmes plus récents que la 2.2 pour profiter de leur stabilité par exemple, mais sans exploiter les capacités du SDK de l'API 8. De même, vous pouvez très bien rendre disponibles aux utilisateurs d'Android 1.6 vos applications développées avec l'API 7 si vous n'exploitez pas les nouveautés introduites par l'API 7, mais c'est plus complexe.

Enfin, cette fenêtre se conclut par trois cases à cocher :

- La première, intitulée `Create custom launcher icon`, ouvrira à la fenêtre suivante un outil pour vous aider à construire une icône pour votre application à partir d'une image préexistante.
- Cochez la deuxième, `Mark this project as a library`, si votre projet est uniquement une bibliothèque de fonctions. Si vous ne comprenez pas, laissez cette case décochée.
- Et la dernière, celle qui s'appelle `Create Project in Workspace`, si vous souhaitez que soit créé pour votre projet un répertoire dans votre espace de travail (*workspace*), vous savez, l'emplacement qu'on a défini au premier lancement d'Eclipse ! Si vous décochez cette case, vous devrez alors spécifier où vous souhaitez que vos fichiers soient créés.

Pour passer à la page suivante, cliquez sur `Next`. Si vous avez cliqué sur `Create custom launcher icon`, alors c'est la fenêtre visible à la figure 3.7 qui s'affichera.

Je vous invite à jouer avec les boutons pour découvrir toutes les fonctionnalités de cet outil. Cliquez sur `Next` une fois obtenu un résultat satisfaisant et vous retrouverez la page que vous auriez eue si vous n'aviez pas cliqué sur `Create custom launcher icon` (voir figure 3.8).

Il s'agit ici d'un outil qui vous demande si vous voulez qu'Eclipse crée une activité pour vous, et si oui à partir de quelle mise en page. On va déclarer qu'on veut qu'il crée une activité, cliquez sur la case à gauche de `Create Activity`, mais on va sélectionner `BlankActivity` parce qu'on veut rester maître de notre mise en page. Cliquez à nouveau sur `Next`.

Si vous ne souhaitez pas qu'Eclipse crée une activité, alors vous devrez cliquer sur `Finish`, car la prochaine page concerne l'activité que nous venons de créer. Cependant, pour notre premier projet, on voudra créer automatiquement une activité.

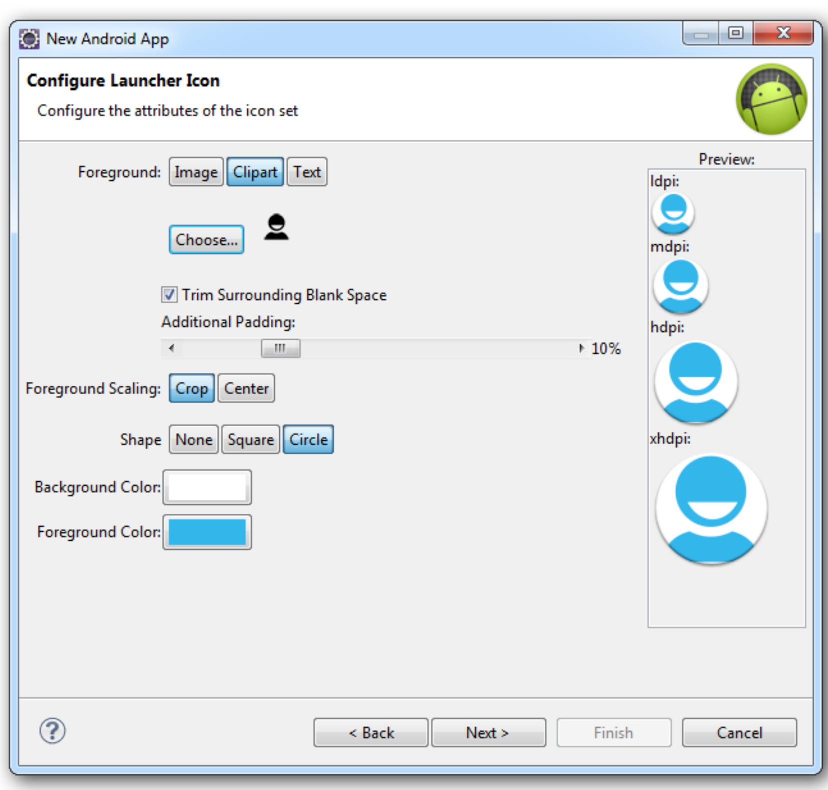


FIGURE 3.7 – Cet outil facilite la création d'icônes

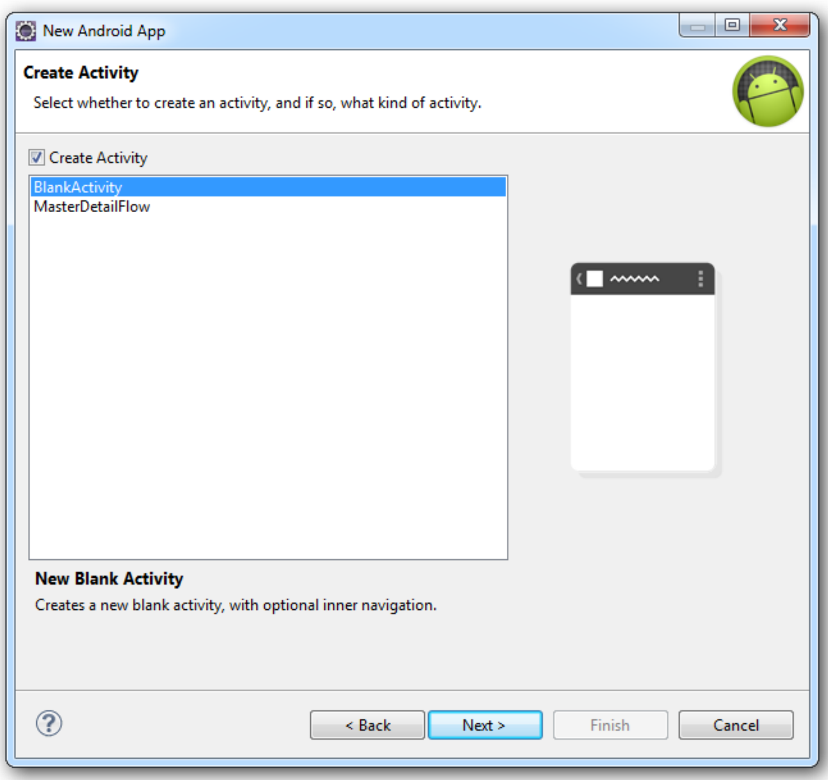


FIGURE 3.8 – Vous pouvez ici choisir une mise en page standard

Dans la fenêtre représentée à la figure 3.9, il faut déclarer certaines informations relatives à notre nouvelle activité.

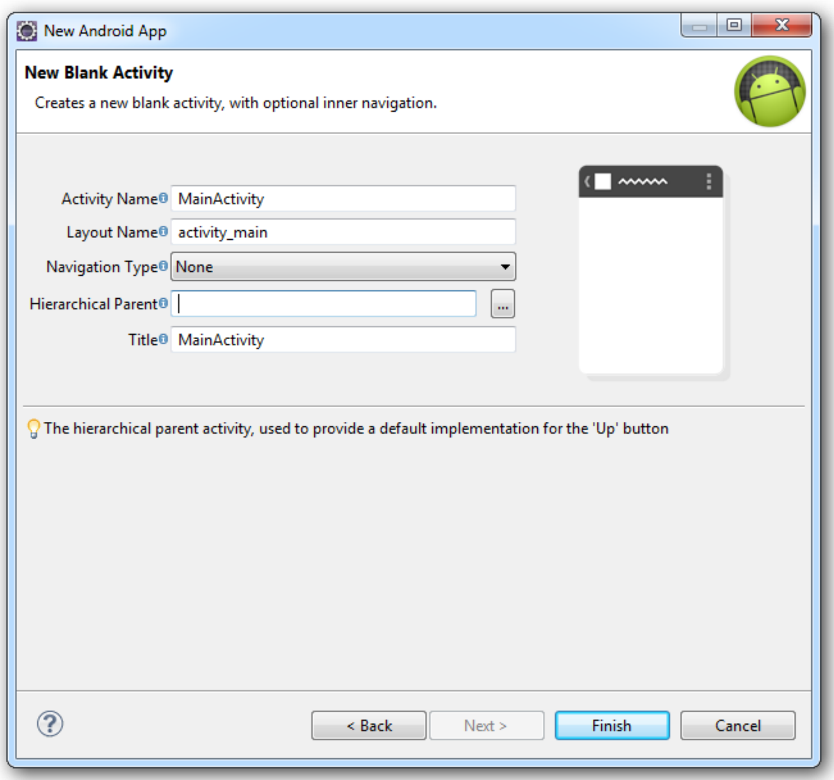


FIGURE 3.9 – Permet de créer une première activité facilement

Ici encore une fois, on fait face à cinq champs à renseigner :

- `Activity Name` permet d'indiquer le nom de la classe Java qui contiendra votre activité, ce champ doit donc respecter la syntaxe Java standard.
- Le champ suivant, `Layout Name`, renseignera sur le nom du fichier qui contiendra l'interface graphique qui correspondra à cette activité.
- En ce qui concerne `Navigation Type`, son contenu est trop complexe pour être analysé maintenant. Sachez qu'il permet de définir facilement comment s'effectueront les transitions entre plusieurs activités.
- Un peu inutile ici, `Hierarchical Parent` permet d'indiquer vers quelle activité va être redirigé l'utilisateur quand il utilisera le bouton `Retour` de son terminal. Comme il s'agit de la première activité de notre application, il n'y a pas de navigation à gérer en cas de retour en arrière.
- Enfin, `Title` est tout simplement le titre qui s'affichera en haut de l'activité.

Pour finaliser la création, cliquez sur `Finish`.

Un non-Hello world !

Vous trouverez les fichiers créés dans le `Package Explorer` (voir figure 3.10).

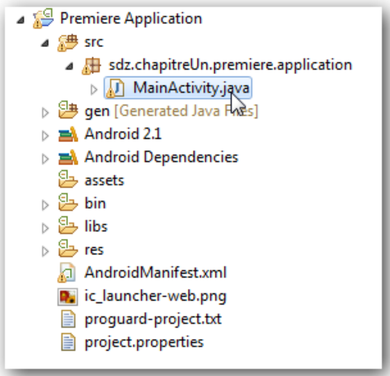


FIGURE 3.10 – Le Package Explorer permet de naviguer entre vos projets

On y trouve notre premier grand répertoire `src/`, celui qui contiendra tous les fichiers sources `.java`. Ouvrez le seul fichier qui s'y trouve, chez moi `MainActivity.java` (en double cliquant dessus). Vous devriez avoir un contenu plus ou moins similaire à celui-ci :

```
package sdz.chapitreUn.premiere.application;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.MenuItem;
import android.support.v4.app.NavUtils;

public class MainActivity extends Activity {

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.activity_main, menu);
    return true;
  }
}
```

Ah ! On reconnaît certains termes que je viens tout juste d'expliquer ! Je vais prendre

toutes les lignes une par une, histoire d'être certain de ne déstabiliser personne.

```
1 package sdz.chapitreUn.premiere.application;
```

Là, on déclare que notre programme se situe dans le package `sdz.chapitreUn.premiere.application`, comme expliqué précédemment. Si on veut faire référence à notre application, il faudra faire référence à ce package.

```
1 import android.os.Bundle;
2 import android.app.Activity;
3 import android.view.Menu;
4 import android.view.MenuItem;
5 import android.support.v4.app.NavUtils;
```

On importe des classes qui se trouvent dans des packages différents : les classes `Activity`, `Bundle`, `Menu` et `MenuItem` qui se trouvent dans le même package, puis `NavUtils`. Chez moi, deux de ces packages sont inutiles car inutilisés dans le code, comme le montre la figure 3.11.

FIGURE 3.11 – Eclipse souligne les importations inutiles en jaune

Il existe trois manières de résoudre ces problèmes :

- Vous pouvez tout simplement ignorer ces avertissements. Votre application fonctionnera toujours, et les performances n'en souffrirons pas. Mais je vois au moins deux raisons de le faire tout de même : pour entretenir un code plus lisible et pour éviter d'avoir par inadvertance deux classes avec le même nom, ce qui peut provoquer des conflits.
- Supprimer les lignes manuellement, mais comme nous avons un outil puissant entre les mains, autant laisser Eclipse s'en charger pour nous !
- Demander à Eclipse d'organiser les importations automatiquement. Il existe un raccourci qui fait cela : `CTRL` + `SHIFT` + `O`. Hop ! Tous les imports inutilisés sont supprimés !

```
1 public class MainActivity extends Activity {
2   //...
3 }
```

On déclare ici une nouvelle classe, `MainActivity`, et on la fait dériver de `Activity`, puisqu'il s'agit d'une activité.

```
1 @Override
2 public void onCreate(Bundle savedInstanceState) {
3   //...
4 }
```

Le petit `@Override` permet d'indiquer que l'on va redéfinir une méthode qui existait auparavant dans la classe parente, ce qui est logique puisque vous saviez déjà qu'une activité avait une méthode `void onCreate()` et que notre classe héritait de `Activity`.

L'instruction `@Override` est facultative. Elle permet au compilateur d'optimiser le bytecode, mais, si elle ne fonctionne pas chez vous, n'insistez pas, supprimez-la.

Cette méthode est la première qui est lancée au démarrage d'une application, mais elle est aussi appelée après qu'une application a été tuée par le système en manque de mémoire ! C'est à cela que sert le paramètre de type `Bundle` :

- S'il s'agit du premier lancement de l'application ou d'un démarrage alors qu'elle avait été quittée normalement, il vaut `null`.
- Mais s'il s'agit d'un retour à l'application après qu'elle a perdu le focus et redémarré, alors il se peut qu'il ne soit pas `null` si vous avez fait en sorte de sauvegarder des données dedans, mais nous verrons comment dans quelques chapitres, puisque ce n'est pas une chose indispensable à savoir pour débuter.

Dans cette méthode, vous devez définir ce qui doit être créé à chaque démarrage, en particulier l'interface graphique.

```
1 super.onCreate(savedInstanceState);
```

L'instruction `super` signifie qu'on fait appel à une méthode ou un attribut qui appartient à la superclasse de la méthode actuelle, autrement dit la classe juste au-dessus dans la hiérarchie de l'héritage — la classe parente, c'est-à-dire la classe `Activity`.

Ainsi, `super.onCreate` fait appel au `onCreate` de la classe `Activity`, mais pas au `onCreate` de `MainActivity`. Il gère bien entendu le cas où le `Bundle` est `null`. Cette instruction est obligatoire.

L'instruction suivante :

```
1 setContentView(R.layout.activity_main);
```

sera expliquée dans le prochain chapitre.

En revanche, l'instruction suivante :

```
1 @Override
2 public boolean onCreateOptionsMenu(Menu menu) {
3   getMenuInflater().inflate(R.menu.activity_main, menu);
4   return true;
5 }
```

... sera expliquée bien, bien plus tard.

En attendant, vous pouvez remplacer le contenu du fichier par celui-ci :

```
1 //N'oubliez pas de déclarer le bon package dans lequel se
      trouve le fichier !
2
```

```
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.widget.TextView;
6
7  public class MainActivity extends Activity {
8    private TextView coucou = null;
9
10   @Override
11   public void onCreate(Bundle savedInstanceState) {
12     super.onCreate(savedInstanceState);
13
14     coucou = new TextView(this);
15     coucou.setText("Bonjour, vous me devez 1 000 000€.");
16     setContentView(coucou);
17   }
18
19 }
```

Nous avons ajouté un attribut de classe que j'ai appelé `coucou`. Cet attribut est de type `TextView`, j'imagine que le nom est déjà assez explicite. Il s'agit d'une vue (`View`)... qui représente un texte (`Text`). J'ai changé le texte qu'affichera cette vue avec la méthode `void setText(String texte)`.

La méthode `void setContentView (View vue)` permet d'indiquer l'interface graphique de notre activité. Si nous lui donnons un `TextView`, alors l'interface graphique affichera ce `TextView` et rien d'autre.

Lancement de l'application

Souvenez-vous, je vous ai dit précédemment qu'il était préférable de ne pas fermer l'AVD, celui-ci étant long à se lancer. Si vous l'avez fermé, ce n'est pas grave, il s'ouvrira tout seul. Mais ce sera loooong.

Pour lancer notre application, regardez la barre d'outils d'Eclipse et cherchez l'encart visible à la figure 3.12.

FIGURE 3.12 – Les outils pour exécuter votre code

Il vous suffit de cliquer sur le deuxième bouton (celui qui ressemble au symbole « *play* »). Une fenêtre s'ouvre (voir figure 3.13) pour vous demander comment exécuter l'application. Sélectionnez `Android Application`.

Si vous avez plusieurs terminaux, l'écran visible à la figure 3.14 s'affichera (sauf si vous n'avez pas de terminal connecté).

On vous demande sur quel terminal vous voulez lancer votre application. Vous pouvez

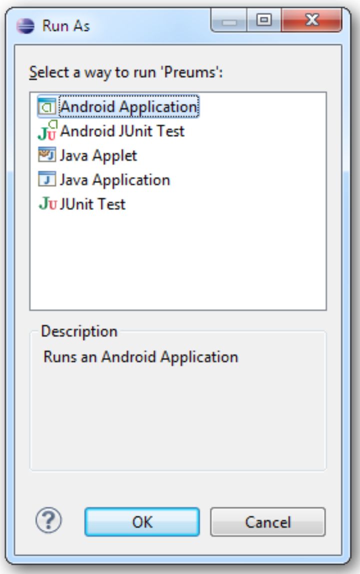


FIGURE 3.13 – Sélectionnez « Android Application »

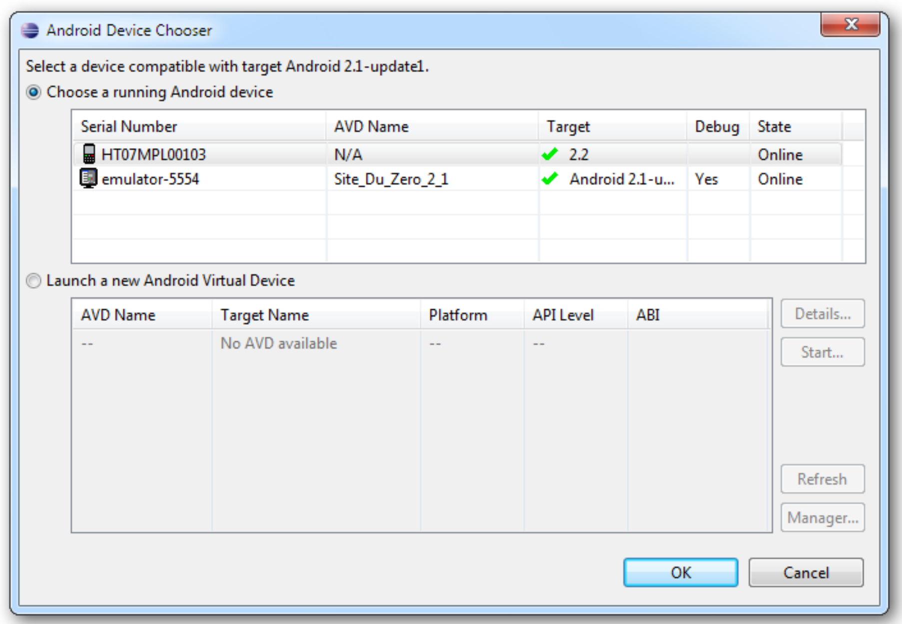


FIGURE 3.14 – Choisissez le terminal de test

valider en cliquant sur OK. Le résultat devrait s'afficher sur votre terminal ou dans l'émulateur (voir figure 3.15). Génial ! L'utilisateur (naïf) vous doit 1 000 000 € !

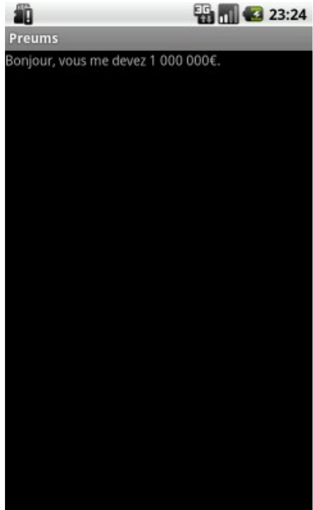


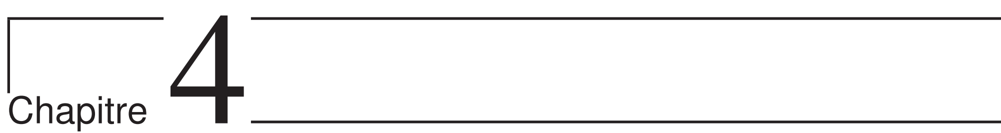
FIGURE 3.15 – Les couleurs peuvent être différentes chez vous, ce n'est pas grave

J'ai une erreur ! Apparemment liée au(x) @Override, le code ne fonctionne pas !

Le problème est que vous utilisez le JDK 7, alors que j'utilise le JDK 6 comme je l'ai indiqué dans le chapitre précédent. Ce n'est pas grave, il vous suffit de supprimer tous les @Override et le code fonctionnera normalement.

En résumé

- Pour avoir des applications fluides et optimisées, il est essentiel de bien comprendre le cycle de vie des activités.
- Chaque écran peut être considéré comme une Activity, qui est constitué d'un contexte et d'une interface graphique. Le contexte fait le lien entre l'application et le système alors que l'interface graphique se doit d'afficher à l'écran des données et permettre à l'utilisateur d'interagir avec l'activité.
- Pour concevoir une navigation impeccable entre vos différentes activités, vous devez comprendre comment fonctionne la pile des activités. Cette structure retirera en premier la dernière activité qui aura été ajoutée.



Chapitre 4

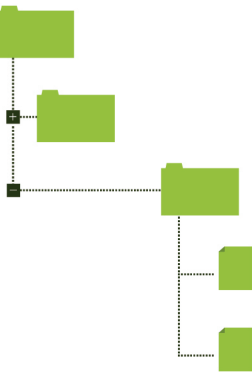
Les ressources

Difficulté :

Je vous ai déjà présenté le répertoire `src/` qui contient toutes les sources de votre programme. On va maintenant s'intéresser à un autre grand répertoire : `res/`. Vous l'aurez compris, c'est dans ce répertoire que sont conservées les ressources, autrement dit les éléments qui s'afficheront à l'écran ou avec lesquels l'utilisateur pourra interagir.

Android est destiné à être utilisé sur un très grand nombre de supports différents, et il faut par conséquent s'adapter à ces supports. Un des moyens d'adapter nos applications à tous les terminaux est d'utiliser les **ressources**. Les ressources sont des fichiers organisés d'une manière particulière de façon à ce qu'Android sache quelle ressource utiliser pour s'adapter au matériel sur lequel s'exécute l'application. Comme je l'ai dit précédemment, adapter nos applications à tous les types de terminaux est indispensable. Cette adaptation passe par la maîtrise des ressources.

Pour déclarer des ressources, on passe très souvent par le format XML, c'est pourquoi un point sur ce langage est nécessaire.



Le format XML

Si vous maîtrisez déjà le XML, vous pouvez passer directement à la suite.

Les langages de balisage

Le XML est un langage de balisage un peu comme le HTML — le HTML est d'ailleurs indirectement un dérivé du XML. Le principe d'un langage de programmation (Java, C++, etc.) est d'effectuer des calculs, puis éventuellement de mettre en forme le résultat de ces calculs dans une interface graphique. À l'opposé, un langage de balisage (XML, donc) n'effectue ni calcul, ni affichage, mais se contente de mettre en forme des informations. Concrètement, un langage de balisage est une syntaxe à respecter, de façon à ce qu'on sache de manière exacte la structuration d'un fichier. Et si on connaît l'architecture d'un fichier, alors il est très facile de retrouver l'emplacement des informations contenues dans ce fichier et de pouvoir les exploiter. Ainsi, il est possible de développer un programme appelé **interpréteur** qui récupérera les données d'un fichier (structuré à l'aide d'un langage de balisage).

Par exemple pour le HTML, c'est un navigateur qui interprète le code afin de donner un sens aux instructions ; si vous lisez un document HTML sans interpréteur, vous ne verrez que les sources, pas l'interprétation des balises.

Un exemple pratique

Imaginons un langage de balisage très simple, que j'utilise pour stocker mes contacts téléphoniques :

```
1 Anaïs Romain Thomas Xavier
```

Ce langage est très simple : les prénoms de mes contacts sont séparés par une espace. Ainsi, quand je demanderai à mon interpréteur de lire le fichier, il saura que j'ai 4 contacts parce que les prénoms sont séparés par des espaces. Il lit une suite de caractères et dès qu'il tombe sur une espace, il sait qu'on va passer à un autre prénom.

On va maintenant rendre les choses plus complexes pour introduire les numéros de téléphone :

```
1 Anaïs : 1111111111
2 Romain: 2222222222
3 Thomas: 3333333333
4 Xavier: 4444444444
```

Là, l'interpréteur sait que pour chaque ligne, la première suite de caractères correspond à un prénom qui se termine par un deux-points, puis on trouve le numéro de téléphone qui se termine par un retour à la ligne. Et, si j'ai bien codé mon interpréteur, il sait

que le premier prénom est « Anaïs » sans prendre l'espace à la fin, puisque ce n'est pas un caractère qui rentre dans la composition d'un prénom.

Si j'avais écrit mon fichier sans syntaxe particulière à respecter, alors il m'aurait été impossible de développer un interpréteur qui puisse retrouver les informations.

La syntaxe XML

Comme pour le format HTML, un fichier XML débute par une déclaration qui permet d'indiquer qu'on se trouve bien dans un fichier XML.

```
1 <?xml version="1.0" encoding="utf-8"?>
```

Cette ligne permet d'indiquer que :

- On utilise la `version 1.0` de XML.
- On utilise l'encodage des caractères qui s'appelle `utf-8` ; c'est une façon de décrire les caractères que contiendra notre fichier.

Je vais maintenant vous détailler un fichier XML :

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <bibliotheque>
3    <livre style="fantaisie">
4      <auteur>George R. R. MARTIN</auteur>
5      <titre>A Game Of Thrones</titre>
6      <langue>klingon</langue>
7      <prix>10.17</prix>
8    </livre>
9    <livre style="aventure">
10     <auteur>Alain Damasio</auteur>
11     <titre>La Horde Du Contrevent</titre>
12     <prix devise="euro">9.40</prix>
13     <recommandation note="20"/>
14   </livre>
15 </bibliotheque>
```

L'élément de base du format XML est la *balise*. Elle commence par un chevron ouvrant $<$ et se termine par un chevron fermant $>$. Entre ces deux chevrons, on trouve au minimum un mot. Par exemple `<bibliotheque>`. Cette balise s'appelle *balise ouvrante*, et autant vous le dire tout de suite : il va falloir la fermer ! Il existe deux manières de fermer une balise ouvrante :

- Soit par une *balise fermante* `</bibliotheque>`, auquel cas vous pourrez avoir du contenu entre la balise ouvrante et la balise fermante. Étant donné que notre bibliothèque est destinée à contenir plusieurs livres, nous avons opté pour cette solution.
- Soit on ferme la balise directement dans son corps : `<bibliotheque />`. La seule différence est qu'on ne peut pas mettre de contenu entre deux balises… puisqu'il n'y en a qu'une. Dans notre exemple, nous avons mis la balise `<recommandation note="20"/>` sous cette forme par choix, mais nous aurions tout aussi bien pu utiliser `<recommandation>20</recommandation>`, cela n'aurait pas été une erreur.

Ce type d'informations, qu'il soit fermé par une balise fermante ou qu'il n'en n'ait pas besoin, s'appelle un *nœud*. Vous voyez donc que l'on a un nœud appelé `bibliotheque`, deux nœuds appelés `livre`, etc.

Un langage de balisage n'a pas de sens en lui-même. Dans notre exemple, notre nœud s'appelle `bibliotheque`, on en déduit, nous humains et peut-être, s'ils nous lisent, vous Cylons, qu'il représente une bibliothèque, mais si on avait décidé de l'appeler `fkldjsdfljsdfkls`, il aurait autant de sens au niveau informatique. C'est à vous d'attribuer un sens à votre fichier XML au moment de l'interprétation.

Le nœud `<bibliotheque>`, qui est le nœud qui englobe tous les autres nœuds, s'appelle la **racine**. Il y a dans un fichier XML *au moins une racine* et *au plus une racine*. Oui ça veut dire qu'il y a exactement une racine par fichier.

On peut établir toute une hiérarchie dans un fichier XML. En effet, entre la balise ouvrante et la balise fermante d'un nœud, il est possible de mettre d'autres nœuds. Les nœuds qui se trouvent dans un autre nœud s'appellent des **enfants**, et le nœud encapsulant s'appelle le **parent**.

Les nœuds peuvent avoir des **attributs** pour indiquer des informations. Dans notre exemple, le nœud `<prix>` a l'attribut `devise` afin de préciser en quelle devise est exprimé ce prix : `<prix devise="euro">9.40</prix>` pour *La Horde Du Contrevent*, qui vaut donc 9€40. Vous remarquerez que pour *A Game Of Thrones* on a aussi le nœud `prix`, mais il n'a pas l'attribut `devise` ! C'est tout à fait normal : dans l'interpréteur, si la devise est précisée, alors je considère que le prix est exprimé en cette devise ; mais si l'attribut `devise` n'est pas précisé, alors le prix est en dollars. *A Game Of Thrones* vaut donc $10.17. Le format XML en lui-même ne peut pas détecter si l'absence de l'attribut `devise` est une anomalie, cela retirerait toute la liberté que permet le format.

En revanche, le XML est intransigeant sur la syntaxe. Si vous ouvrez une balise, n'oubliez pas de la fermer par exemple !

Les différents types de ressources

Les ressources sont des éléments capitaux dans une application Android. On y trouve par exemple des chaînes de caractères ou des images. Comme Android est destiné à être utilisé sur une grande variété de supports, il fallait trouver une solution pour permettre à une application de s'afficher de la même manière sur un écran 7« que sur un écran 10 », ou faire en sorte que les textes s'adaptent à la langue de l'utilisateur. C'est pourquoi les différents éléments qui doivent s'adapter de manière très précise sont organisés de manière tout aussi précise, de façon à ce qu'Android sache quels éléments utiliser pour quels types de terminaux.

On découvre les ressources à travers une hiérarchie particulière de répertoires. Vous pouvez remarquer qu'à la création d'un nouveau projet, Eclipse crée certains répertoires par défaut, comme le montre la figure 4.1.

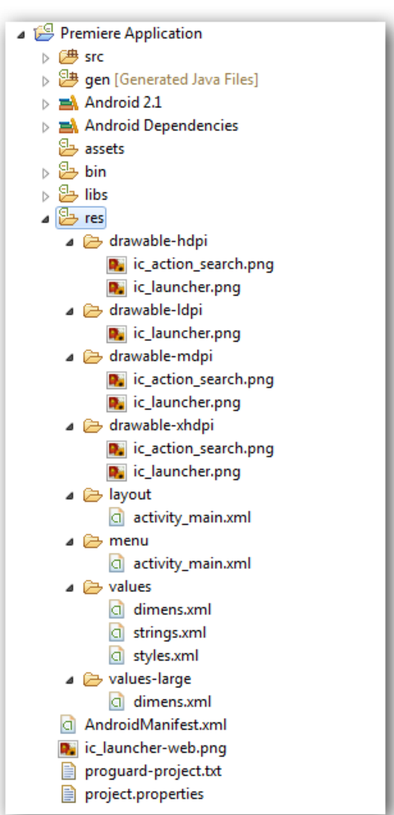


FIGURE 4.1 – L'emplacement des ressources au sein d'un projet

Je vous ai déjà dit que les ressources étaient divisées en plusieurs types. Pour permettre à Android de les retrouver facilement, chaque type de ressources est associé à un répertoire particulier. Voici un tableau qui vous indique les principales ressources que l'on peut trouver, avec le nom du répertoire associé. Vous remarquerez que seuls les répertoires les plus courants sont créés par défaut.

Type	**Description**	**Analyse syntaxique**
Dessin et image (`res/drawable`)	On y trouve les images matricielles (les images de type PNG, JPEG ou encore GIF) ainsi que des fichiers XML qui permettent de décrire des dessins simples (par exemple des cercles ou des carrés).	Oui
Mise en page ou interface graphique (`res/layout`)	Les fichiers XML qui représentent la disposition des vues (on abordera cet aspect, qui est très vaste, dans la prochaine partie).	Exclusivement
Menu (`res/menu`)	Les fichiers XML pour pouvoir constituer des menus.	Exclusivement
Donnée brute (`res/raw`)	Données diverses au format brut. Ces données ne sont pas des fichiers de ressources standards, on pourrait y mettre de la musique ou des fichiers HTML par exemple.	**Le moins possible**
Différentes variables (`res/values`)	Il est plus difficile de cibler les ressources qui appartiennent à cette catégorie tant elles sont nombreuses. On y trouve entre autre des variables standards, comme des chaînes de caractères, des dimensions, des couleurs, etc.	Exclusivement

La colonne « Analyse syntaxique » indique la politique à adopter pour les fichiers XML de ce répertoire. Elle vaut :

- « Exclusivement », si les fichiers de cette ressource sont tout le temps des fichiers XML.
- « Oui », si les fichiers peuvent être d'un autre type que XML, en fonction de ce qu'on veut faire. Ainsi, dans le répertoire `drawable/`, on peut mettre des images ou des fichiers XML dont le contenu sera utilisé par un interpréteur pour dessiner des images.
- « Le moins possible », si les fichiers doivent de préférence ne pas être de type XML. Pourquoi ? Parce que tous les autres répertoires sont suffisants pour stocker des fichiers XML. Alors, si vous voulez placer un fichier XML dans le répertoire `raw/`, c'est qu'il ne trouve *vraiment* pas sa place dans un autre répertoire.

Il existe d'autres répertoires pour d'autres types de ressources, mais je ne vais pas toutes vous les présenter. De toute manière, on peut déjà faire des applications complexes avec ces ressources-là.

Ne mettez pas de ressources directement dans `res/`, sinon vous aurez une erreur de compilation !

L'organisation

Si vous êtes observateurs, vous avez remarqué sur l'image précédente que nous avions trois répertoires `res/drawable/`, alors que dans le tableau que nous venons de voir, je vous disais que les drawables allaient tous dans le répertoire `res/drawable/` et point barre ! C'est tout à fait normal et ce n'est pas anodin du tout.

Comme je vous le disais, nous avons plusieurs ressources à gérer en fonction du matériel. Les emplacements indiqués dans le tableau précédent sont les emplacements par défaut, c'est-à-dire qu'il s'agit des emplacements qui visent le matériel le plus générique possible. Par exemple, vous pouvez considérer que le matériel le plus générique est un système *qui n'est pas en coréen*, alors vous allez mettre dans le répertoire par défaut tous les fichiers qui correspondent aux systèmes qui ne sont pas en coréen (par exemple les fichiers de langue). Pour placer des ressources destinées aux systèmes en coréen, on va créer un sous-répertoire et préciser qu'il est destiné aux systèmes en coréen. Ainsi, automatiquement, quand un utilisateur français ou anglais utilisera votre application, Android choisira les fichiers dans l'emplacement par défaut, alors que si c'est un utilisateur coréen, il ira chercher dans les sous-répertoires consacrés à cette langue.

En d'autres termes, en partant du nom du répertoire par défaut, il est possible de créer d'autres répertoires qui permettent de préciser à quels types de matériels les ressources contenues dans ce répertoire sont destinées. Les restrictions sont représentées par des **quantificateurs** et ce sont ces quantificateurs qui vous permettront de préciser le matériel pour lequel les fichiers dans ce répertoire sont destinés. La syntaxe à respecter peut être représentée ainsi : `res/<type_de_ressource>[<-quantificateur 1><-quantificateur 2>...<-quantificateur N>]`

Autrement dit, on peut n'avoir aucun quantificateur si l'on veut définir l'emplacement par défaut, ou en avoir un pour réduire le champ de destination, deux pour réduire encore plus, etc. Ces quantificateurs sont séparés par un tiret. Si Android ne trouve pas d'emplacement dont le nom corresponde exactement aux spécifications techniques du terminal, il cherchera parmi les autres répertoires qui existent la solution la plus proche. Je vais vous montrer les principaux quantificateurs (il y en a quatorze en tout, dont un bon paquet qu'on utilise rarement, j'ai donc décidé de les ignorer).

Vous n'allez pas comprendre l'attribut `Priorité` tout de suite, d'ailleurs il est possible que vous ne compreniez pas tout immédiatement. Lisez cette partie tranquillement, zieutez ensuite les exemples qui suivent, puis revenez à cette partie une fois que vous aurez tout compris.

Langue et région

`Priorité :` **2** La langue du système de l'utilisateur. On indique une langue puis, éventuellement, on peut préciser une région avec « -r ». Exemples :

- `en` pour l'anglais ;
- `fr` pour le français ;
- `fr-rFR` pour le français mais uniquement celui utilisé en France ;
- `fr-rCA` pour le français mais uniquement celui utilisé au Québec ;
- Etc.

Taille de l'écran

`Priorité :` **3** Il s'agit de la taille de la diagonale de l'écran :

- `small` pour les écrans de petite taille ;
- `normal` pour les écrans standards ;
- `large` pour les grands écrans, comme dans les tablettes tactiles ;
- `xlarge` pour les très grands écrans, là on pense carrément aux téléviseurs.

Orientation de l'écran

`Priorité :` **5** Il existe deux valeurs :

- `port` : c'est le diminutif de *portrait*, donc quand le terminal est en mode portrait ;
- `land` : c'est le diminutif de *landscape*, donc quand le terminal est en mode paysage.

Résolution de l'écran

`Priorité :` **8**

- `ldpi` : environ 120 dpi ;
- `mdpi` : environ 160 dpi ;
- `hdpi` : environ 240 dpi ;
- `xhdpi` : environ 320 dpi (disponible à partir de l'API 8 uniquement) ;
- `nodpi` : pour ne pas redimensionner les images matricielles (vous savez, JPEG, PNG et GIF !).

Version d'Android

`Priorité :` **14** Il s'agit du niveau de l'API (v3, v5, v7 (c'est celle qu'on utilise nous !), etc.).

Regardez l'image précédente (qui de toute façon représente les répertoires créés automatiquement pour tous les projets), que se passe-t-il si l'écran du terminal de l'utilisateur a une grande résolution ? Android ira chercher dans `res/drawable-hdpi` ! L'écran du terminal de l'utilisateur a une petite résolution ? Il ira chercher dans `res/drawable-ldpi/` ! L'écran du terminal de l'utilisateur a une très grande résolution ? Eh bien… il ira

chercher dans `res/drawable-hdpi` puisqu'il s'agit de la solution la plus proche de la situation matérielle réelle.

Exemples et règles à suivre

- `res/drawable-small` pour avoir des images spécifiquement pour les petits écrans.
- `res/drawable-large` pour avoir des images spécifiquement pour les grands écrans.
- `res/layout-fr` pour avoir une mise en page spécifique destinée à tous ceux qui ont un système en français.
- `res/layout-fr-rFR` pour avoir une mise en page spécifique destinée à ceux qui ont choisi la langue *Français (France)*.
- `res/values-fr-rFR-port` pour des données qui s'afficheront uniquement à ceux qui ont choisi la langue *Français (France)* et dont le téléphone se trouve en orientation portrait.
- `res/values-port-fr-rFR` n'est pas possible, c'est à ça que servent les priorités : *il faut impérativement mettre les quantificateurs par ordre croissant de priorité*. La priorité de la langue est 2, celle de l'orientation est 5, comme $2 < 5$ on doit placer les langues avant l'orientation.
- `res/layout-fr-rFR-en` n'est pas possible puisqu'on a deux quantificateurs de même priorité et qu'il faut toujours respecter l'ordre croissant des priorités. Il nous faudra créer un répertoire pour le français et un répertoire pour l'anglais.

Tous les répertoires de ressources qui sont différenciés par des quantificateurs devront avoir le même contenu : on indique à Android de quelle ressource on a besoin, sans se préoccuper dans quel répertoire aller le chercher, Android le fera très bien pour nous. Sur l'image précédente, vous voyez que l'icône se trouve dans les trois répertoires `drawable/`, sinon Android ne pourrait pas la trouver pour les trois types de configuration.

Mes recommandations

Voici les règles que je respecte pour la majorité de mes projets, quand je veux faire bien les choses :

- `res/drawable-hdpi` ;
- `res/drawable-ldpi` ;
- `res/drawable-mdpi` ;
- Pas de `res/drawable` ;
- `res/layout-land` ;
- `res/layout`.

Une mise en page pour chaque orientation et des images adaptées pour chaque résolution. Le quantificateur de l'orientation est surtout utile pour l'interface graphique. Le quantificateur de la résolution sert plutôt à ne pas avoir à ajuster une image et par conséquent à ne pas perdre de qualité.

Pour finir, sachez que les écrans de taille `small` et `xlarge` se font rares.

Ajouter un fichier avec Eclipse

Heureusement, les développeurs de l'ADT ont pensé à nous en créant un petit menu qui vous aidera à créer des répertoires de manière simple, sans avoir à retenir de syntaxe. En revanche, il vous faudra parler un peu anglais, je le crains. Faites un clic droit sur n'importe quel répertoire ou fichier de votre projet. Vous aurez un menu un peu similaire à celui représenté à l'image 4.2, qui s'affichera.

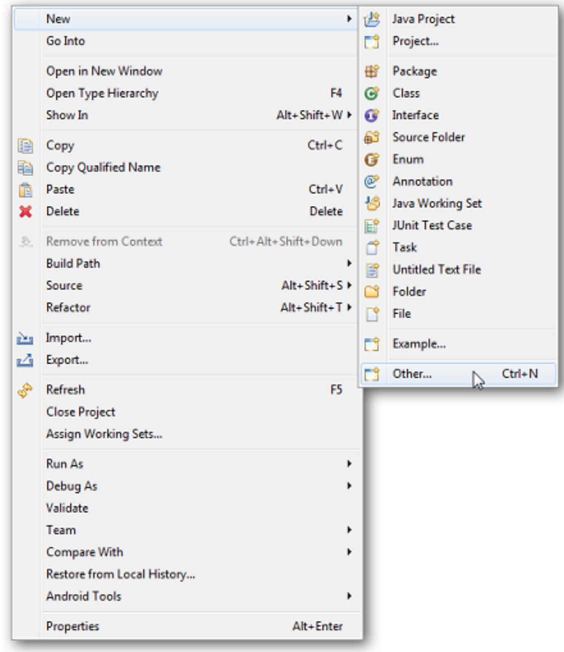


FIGURE 4.2 – L'ADT permet d'ajouter des répertoires facilement

Dans le sous-menu `New`, soit vous cliquez directement sur `Android XML File`, soit, s'il n'est pas présent, vous devrez cliquer sur `Other...`, puis chercher `Android XML File` dans le répertoire `Android`. Cette opération ouvrira un assistant de création de fichiers XML visible à la figure 4.3.

Le premier champ vous permet de sélectionner le type de ressources désiré. Vous retrouverez les noms des ressources que nous avons décrites dans le premier tableau, ainsi que d'autres qui nous intéressent moins, à l'exception de `raw` puisqu'il n'est pas destiné à contenir des fichiers XML. À chaque fois que vous changez de type de ressources, la seconde partie de l'écran change et vous permet de choisir plus facilement quel genre de ressources vous souhaitez créer. Par exemple pour `Layout`, vous pouvez choisir de créer un bouton (`Button`) ou un encart de texte (`TextView`). Vous pouvez ensuite choisir dans quel projet vous souhaitez ajouter le fichier. Le champ `File` vous permet quant à lui de choisir le nom du fichier à créer.

Une fois votre sélection faite, vous pouvez cliquer sur `Next` pour passer à l'écran suivant

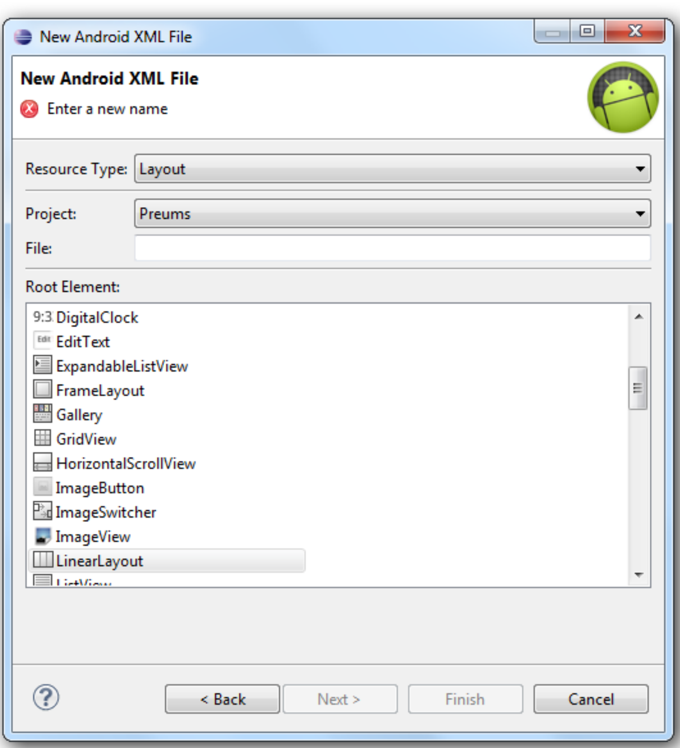


FIGURE 4.3 – L'assistant de création de fichiers XML

(voir figure 4.4) qui vous permettra de choisir des quantificateurs pour votre ressource ou `Finish` pour que le fichier soit créé dans un répertoire sans quantificateurs.

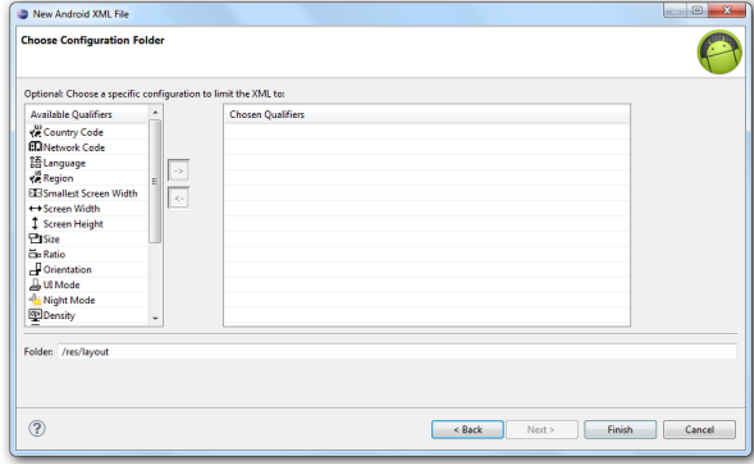


FIGURE 4.4 – Cette fenêtre vous permet de choisir des quantificateurs pour votre ressource

Cette section contient deux listes. Celle de gauche présente les quantificateurs à appliquer au répertoire de destination. Vous voyez qu'ils sont rangés dans l'ordre de priorité que j'ai indiqué.

Mais il y a beaucoup plus de quantificateurs et de ressources que ce que tu nous as indiqué !

Oui. Je n'écris pas une documentation officielle pour Android. Si je le faisais, j'en laisserais plus d'un confus et vous auriez un nombre impressionnant d'informations qui ne vous serviraient pas ou peu. Je m'attelle à vous apprendre à faire de jolies applications optimisées et fonctionnelles, pas à faire de vous des encyclopédies vivantes d'Android.

Le champ suivant, `Folder`, est le répertoire de destination. Quand vous sélectionnez des quantificateurs, vous pouvez avoir un aperçu en temps réel de ce répertoire. Si vous avez commis une erreur dans les quantificateurs, par exemple choisi une langue qui n'existe pas, le quantificateur ne s'ajoutera pas dans le champ du répertoire. Si ce champ ne vous semble pas correct vis-à-vis des quantificateurs sélectionnés, c'est que vous avez fait une faute d'orthographe. Si vous écrivez directement un répertoire dans `Folder`, les quantificateurs indiqués s'ajouteront dans la liste correspondante.

À mon humble avis, la meilleure pratique est d'écrire le répertoire de destination dans `Folder` et de regarder si les quantificateurs choisis s'ajoutent bien dans la liste. Mais personne ne vous en voudra d'utiliser l'outil prévu pour.

Cet outil peut gérer les erreurs et conflits. Si vous indiquez comme nom « strings » et comme ressource une donnée (« values »), vous verrez un petit avertissement qui s'affichera en haut de la fenêtre, puisque ce fichier existe déjà (il est créé par défaut).

Petit exercice

Vérifions que vous avez bien compris : essayez, sans passer par les outils d'automatisation, d'ajouter une mise en page destinée à la version 8, quand l'utilisateur penche son téléphone en mode portrait alors qu'il utilise le français des Belges (`fr-rBE`) et que son terminal a une résolution moyenne.

Le `Folder` doit contenir **exactement** `/res/layout-fr-rBE-port-mdpi-v8`.

Il vous suffit de cliquer sur `Finish` si aucun message d'erreur ne s'affiche.

Récupérer une ressource

La classe `R`

On peut accéder à cette classe qui se trouve dans le répertoire `gen/` (comme *generated*, c'est-à-dire que tout ce qui se trouvera dans ce répertoire sera généré automatiquement), comme indiqué à la figure 4.5.

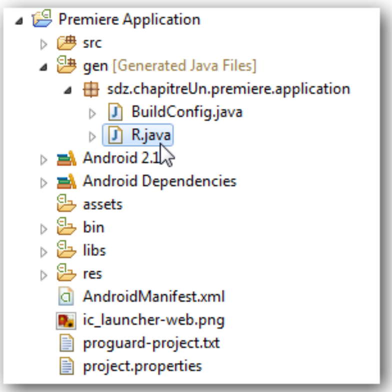


FIGURE 4.5 – On retrouve le fichier `R.java dans gen/<votre_package>/`

Ouvrez donc ce fichier et regardez le contenu.

```
public final class R {
  public static final class attr {
  }
  public static final class dimen {
    public static final int padding_large=0x7f040002;
```

```
      public static final int padding_medium=0x7f040001;
      public static final int padding_small=0x7f040000;
    }
    public static final class drawable {
      public static final int ic_action_search=0x7f020000;
      public static final int ic_launcher=0x7f020001;
    }
    public static final class id {
      public static final int menu_settings=0x7f080000;
    }
    public static final class layout {
      public static final int activity_main=0x7f030000;
    }
    public static final class menu {
      public static final int activity_main=0x7f070000;
    }
    public static final class string {
      public static final int app_name=0x7f050000;
      public static final int hello_world=0x7f050001;
      public static final int menu_settings=0x7f050002;
      public static final int title_activity_main=0x7f050003;
    }
    public static final class style {
      public static final int AppTheme=0x7f060000;
    }
}
```

Ça vous rappelle quelque chose ? Comparons avec l'ensemble des ressources que comporte notre projet (voir figure 4.6).

On remarque en effet une certaine ressemblance, mais elle n'est pas parfaite ! Décryptons certaines lignes de ce code.

La classe `layout`

```
public static final class layout {
    public static final int activity_main=0x7f030000;
}
```

Il s'agit d'une classe déclarée dans une autre classe : c'est ce qui s'appelle une classe **interne**. La seule particularité d'une classe interne est qu'elle est déclarée dans une autre classe, mais elle peut agir comme toutes les autres classes. Cependant, pour y accéder, il faut faire référence à la classe qui la contient. Cette classe est de type `public static final` et de nom `layout`.

- Un élément `public` est un élément auquel tout le monde peut accéder sans aucune restriction.
- Le mot-clé `static`, dans le cas d'une classe interne, signifie que la classe n'est pas liée à une instanciation de la classe qui l'encapsule. Pour accéder à `layout`, on ne doit pas nécessairement créer un objet de type `R`. On peut y accéder par `R.layout`.

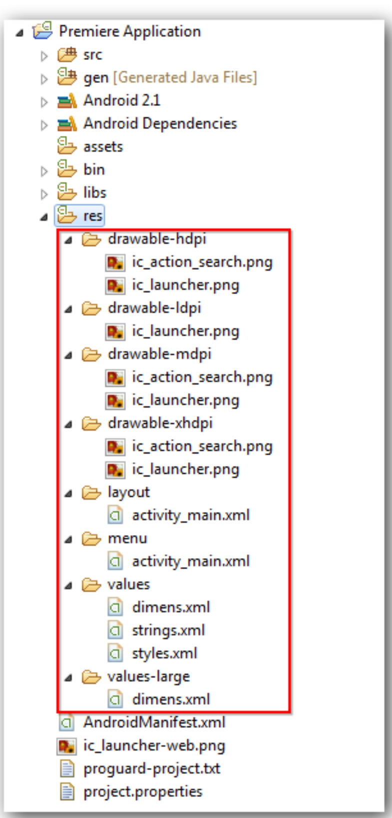


FIGURE 4.6 – Tiens, ces noms me disent quelque chose. . .

– Le mot-clé `final` signifie que l'on ne peut pas créer de classe dérivée de `layout`.

Cette classe contient un unique `public int`, affublé des modificateurs `static` et `final`. Il s'agit par conséquent d'une *constante*, à laquelle n'importe quelle autre classe peut accéder sans avoir à créer d'objet de type `layout` ni de type `R`.

Cet entier est de la forme `0xZZZZZZZZ`. Quand un entier commence par `0x`, c'est qu'il s'agit d'un nombre *hexadécimal* sur 32 bits. Si vous ignorez ce dont il s'agit, ce n'est pas grave, dites-vous juste que ce type de nombre est un nombre exactement comme un autre, sauf qu'il respecte ces règles-ci :

– Il commence par `0x`.
– Après le `0x`, on trouve huit chiffres (ou moins, mais on préfère mettre des 0 pour arriver à 8 chiffres) : `0x123` est équivalent à `0x00000123`, tout comme `123` est la même chose que `00000123`.
– Ces chiffres peuvent aller de 0 à... F. C'est-à-dire qu'au lieu de compter « 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 » on compte « 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F ». A, B, C, D, E et F sont des chiffres normaux, banals, même s'ils n'en n'ont pas l'air, c'est juste qu'il n'y a pas de chiffre après 9, alors il a fallu improviser avec les moyens du bord . Ainsi, après 9 on a A, après A on a B, après E on a F, et après F on a 10 ! Puis à nouveau 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, etc.

Regardez les exemples suivants :

```
1 int deuxNorm = 2; // Valide !
2 int deuxHexa = 0x00000002; // Valide, et vaut la même chose que
    « deuxNorm »
3 int deuxRed = 0x2; // Valide, et vaut la même chose que «
    deuxNorm » et « deuxHexa » (évidemment, 00000002, c'est la m
    ême chose que 2 !)
4 //Ici, nous allons toujours écrire les nombres hexadécimaux
    avec huit chiffres, même les 0 inutiles !
5 int beaucoup = 0x0AFA1B00; // Valide !
6 int marcheraPas = 1x0AFA1B00; // Non ! Un nombre hexadécimal
    commence toujours par « 0x » !
7 int marcheraPasNonPlus = 0xG00000000; // Non ! Un chiffre hexad
    écimal va de 0 à F, on n'accepte pas les autres lettres !
8 int caVaPasLaTete = 0x124!AZ5%; // Alors là c'est carrément n'
    importe quoi !
```

Cet entier a le même nom qu'un fichier de ressources (`activity_main`), tout simplement parce qu'il représente ce fichier (`activity_main.xml`). On ne peut donc avoir qu'un seul attribut de ce nom-là dans la classe, puisque deux fichiers qui appartiennent à la même ressource se trouvent dans le même répertoire et ne peuvent par conséquent pas avoir le même nom. Cet entier est un identifiant unique pour le fichier de mise en page qui s'appelle `activity_main`. Si un jour on veut utiliser ou accéder à cette mise en page depuis notre code, on y fera appel à l'aide de cet identifiant.

La classe `drawable`

```
public static final class drawable {
  public static final int ic_action_search=0x7f020000;
  public static final int ic_launcher=0x7f020001;
}
```

Contrairement au cas précédent, on a un seul entier pour plusieurs fichiers qui ont le même nom ! On a vu dans la section précédente qu'il fallait nommer de façon identique ces fichiers qui ont la même fonction, pour une même ressource, mais avec des quantificateurs différents. Eh bien, quand vous ferez appel à l'identificateur, Android saura qu'il lui faut le fichier `ic_launcher` et déterminera automatiquement quel est le répertoire le plus adapté à la situation du matériel parmi les répertoires des ressources *drawable*, puisqu'on se trouve dans la classe `drawable`.

La classe `string`

```
public static final class string {
  public static final int app_name=0x7f050000;
  public static final int hello_world=0x7f050001;
  public static final int menu_settings=0x7f050002;
  public static final int title_activity_main=0x7f050003;
}
```

Cette fois, si on a quatre entiers, c'est tout simplement parce qu'on a quatre chaînes de caractères dans le fichier `res/values/strings.xml`, qui contient les chaînes de caractères (qui sont des données). Vous pouvez le vérifier par vous-mêmes en fouillant le fichier `strings.xml`.

Je ne le répéterai jamais assez, ne modifiez **jamais** ce fichier par vous-mêmes. Eclipse s'en occupera.

Il existe d'autres variables dont je n'ai pas discuté, mais vous avez tout compris déjà avec ce que nous venons d'étudier.

Application

Énoncé

J'ai créé un nouveau projet pour l'occasion, mais vous pouvez très bien vous amuser avec le premier projet. L'objectif ici est de récupérer la ressource de type chaîne de caractères qui s'appelle `hello_world` (créée automatiquement par Eclipse) afin de la mettre comme texte dans un `TextView`. On affichera ensuite le `TextView`.

On utilisera la méthode `public final void setText (int id)` (`id` étant l'identifiant de la ressource) de la classe `TextView`.

Solution

```
import android.app.Activity;
```

```
2 import android.os.Bundle;
3 import android.widget.TextView;
4
5 public class Main extends Activity {
6   private TextView text = null;
7
8   @Override
9   public void onCreate(Bundle savedInstanceState) {
10    super.onCreate(savedInstanceState);
11
12    text = new TextView(this);
13    text.setText(R.string.hello_world);
14
15    setContentView(text);
16  }
17 }
```

Comme indiqué auparavant, on peut accéder aux identificateurs sans instancier de classe. On récupère dans la classe `R` l'identificateur de la ressource du nom `hello_world` qui se trouve dans la classe `string`, puisqu'il s'agit d'une chaîne de caractères, donc `R.string.hello_world`.

Et si je mets à la place de l'identifiant d'une chaîne de caractères un identifiant qui correspond à un autre type de ressources ?

Eh bien, les ressources sont des objets Java comme les autres. Par conséquent ils peuvent aussi posséder une méthode `public String toString()` ! Pour ceux qui l'auraient oublié, la méthode `public String toString()` est appelée sur un objet pour le transformer en chaîne de caractères, par exemple si on veut passer l'objet dans un `System.out.println`. Ainsi, si vous mettez une autre ressource qu'une chaîne de caractères, ce sera la valeur rendue par la méthode `toString()` qui sera affichée.

Essayez par vous-mêmes, vous verrez ce qui se produit. Soyez curieux, c'est comme ça qu'on apprend !

Application

Énoncé

Je vous propose un autre exercice. Dans le précédent, le `TextView` a récupéré l'identifiant et a été chercher la chaîne de caractères associée pour l'afficher. Dans cet exercice, on va plutôt récupérer la chaîne de caractères pour la manipuler.

Instructions

- On va récupérer le gestionnaire de ressources afin d'aller chercher la chaîne de caractères. C'est un objet de la classe `Resource` que possède notre activité et qui permet d'accéder aux ressources de cette activité. On peut le récupérer grâce à la méthode

`public Resources getResources()`.
– On récupère la chaîne de caractères `hello_world` grâce à la méthode `string getString(int id)`, avec `id` l'identifiant de la ressource.
– Et on modifie la chaîne récupérée.

Solution

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class Main extends Activity {
  private TextView text = null;
  private String hello = null;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    hello = getResources().getString(R.string.hello_world);
    // Au lieu d'afficher "Hello World!" on va afficher "Hello
        les Zéros !"
    hello = hello.replace("world", "les Zéros ");

    text = new TextView(this);
    text.setText(hello);

    setContentView(text);
  }
}
```

J'ai une erreur à la compilation ! Et un fichier similaire à mon fichier XML mais qui se finit par `.out` vient d'apparaître !

Ah, ça veut dire que vous avez téléchargé une version d'Eclipse avec un analyseur syntaxique XML. En fait si vous lancez la compilation alors que vous étiez en train de consulter un fichier XML, alors c'est l'analyseur qui se lancera et pas le compilateur. La solution est donc de cliquer sur n'importe quel autre fichier que vous possédez qui ne soit pas un XML, puis de relancer la compilation.

En résumé

– Au même titre que le langage Java est utile pour développer vos application, le langage XML l'est tout autant puisqu'il a été choisi pour mettre en place les différentes ressources de vos projets.
– Il existe 5 types de ressources que vous utiliserez majoritairement :

  - `drawable` qui contient toutes les images matricielles et les fichiers XML décrivant des dessins simples.
  - `layout` qui contient toutes les interfaces que vous attacherez à vos activités pour mettre en place les différentes vues.
  - `menu` qui contient toutes les déclarations d'éléments pour confectionner des menus.
  - `raw` qui contient toutes les autres ressources au format brut.
  - `values` qui contient des valeurs pour un large choix comme les chaînes de caractères, les dimensions, les couleurs, etc.
- Les quantificateurs sont utilisés pour cibler précisément un certain nombre de priorités ; à savoir la langue et la région, la taille de l'écran, l'orientation de l'écran, la résolution de l'écran et la version d'Android.
- Chaque ressource présente dans le dossier `res` de votre projet génère un identifiant unique dans le fichier `R.java` pour permettre de les récupérer dans la partie Java de votre application.

Deuxième partie

Création d'interfaces graphiques

Chapitre 5

Constitution des interfaces graphiques

Difficulté :

Bien , maintenant que vous avez compris le principe et l'utilité des ressources, voyons comment appliquer nos nouvelles connaissances aux interfaces graphiques. Avec la diversité des machines sous lesquelles fonctionne Android, il faut vraiment exploiter toutes les opportunités offertes par les ressources pour développer des applications qui fonctionneront sur la majorité des terminaux.

Une application Android polyvalente possède un fichier XML pour chaque type d'écran, de façon à pouvoir s'adapter. En effet, si vous développez une application uniquement à destination des petits écrans, les utilisateurs de tablettes trouveront votre travail illisible et ne l'utiliseront pas du tout. Ici on va voir un peu plus en profondeur ce que sont les vues, comment créer des ressources d'interface graphique et comment récupérer les vues dans le code Java de façon à pouvoir les manipuler.

L'interface d'Eclipse

La bonne nouvelle, c'est qu'Eclipse nous permet de créer des interfaces graphiques à la souris. Il est en effet possible d'ajouter un élément et de le positionner grâce à sa souris. La mauvaise, c'est que c'est beaucoup moins précis qu'un véritable code et qu'en plus l'outil est plutôt buggé. Tout de même, voyons voir un peu comment cela fonctionne.

Ouvrez le seul fichier qui se trouve dans le répertoire `res/layout`. Il s'agit normalement du fichier `activity_main.xml`. Une fois ouvert, vous devriez avoir quelque chose qui ressemble à la figure 5.1.

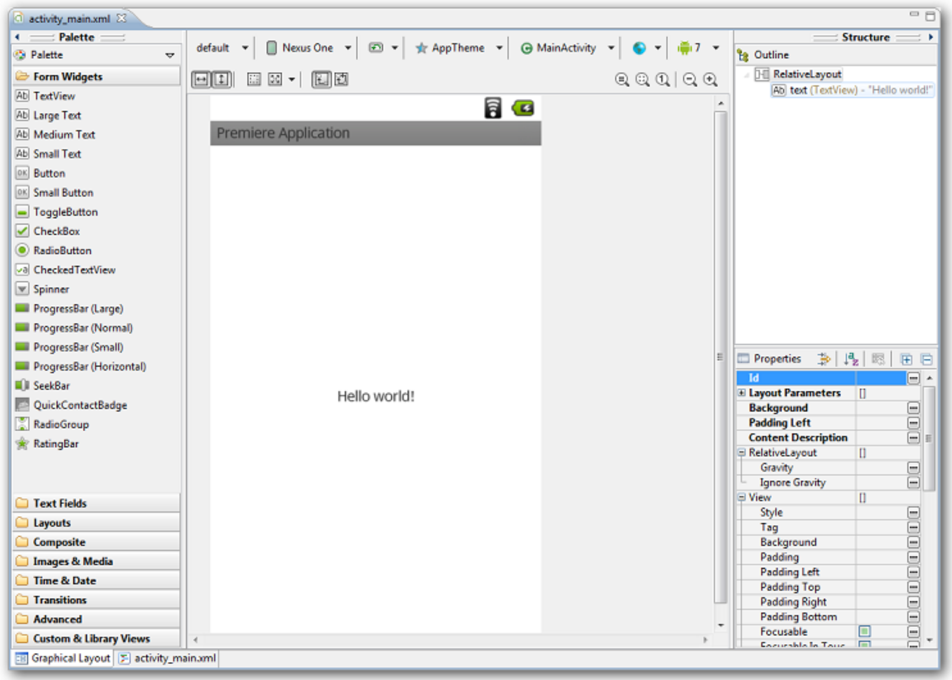


FIGURE 5.1 – Le fichier est ouvert

Cet outil vous aide à mettre en place les vues directement dans le layout de l'application, représenté par la fenêtre du milieu. Comme il ne peut remplacer la manipulation de fichiers XML, je ne le présenterai pas dans les détails. En revanche, il est très pratique dès qu'il s'agit d'afficher un petit aperçu final de ce que donnera un fichier XML.

Présentation de l'outil

C'est à l'aide du menu en haut, celui visible à la figure 5.2, que vous pourrez observer le résultat avec différentes options.

Ce menu est divisé en deux parties : les icônes du haut et celles du bas. Nous allons

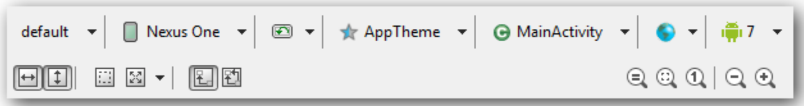


Figure 5.2 – Menu d'options

nous concentrer sur les icônes du haut pour l'instant (voir figure 5.3).

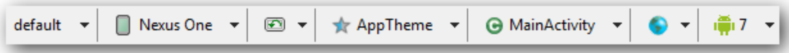


Figure 5.3 – Les icônes du haut du menu d'options

- La première liste déroulante vous permet de naviguer rapidement entre les répertoires de layouts. Vous pouvez ainsi créer des versions alternatives à votre layout actuel en créant des nouveaux répertoires différenciés par leurs quantificateurs.
- La deuxième permet d'observer le résultat en fonction de différentes résolutions. Le chiffre indique la taille de la diagonale en pouces (sachant qu'un pouce fait 2,54 centimètres, la diagonale du **Nexus One** fait $3,7 \times 2,54 = 9\,,4$ cm) et la suite de lettres en majuscules la résolution de l'écran. Pour voir à quoi correspondent ces termes en taille réelle, n'hésitez pas à consulter l'image accessible grâce au code web suivant.
- La troisième permet d'observer l'interface graphique en fonction de certains facteurs. Se trouve-t-on en mode portrait ou en mode paysage ? Le périphérique est-il attaché à un matériel d'amarrage ? Enfin, fait-il jour ou nuit ?
- La suivante permet d'associer un thème à votre activité. Nous aborderons plus tard les thèmes et les styles.
- L'avant-dernière permet de choisir une langue si votre interface graphique change en fonction de la langue.
- Et enfin la dernière vérifie le comportement en fonction de la version de l'API, si vous aviez défini des quantificateurs à ce niveau-là.

▷ Résolutions d'écrans
Code web : 824193

Occupons-nous maintenant de la deuxième partie, tout d'abord avec les icônes de gauche, visibles à la figure 5.4.

Figure 5.4 – Les icônes de gauche du bas menu

Ces boutons sont spécifiques à un composant et à son layout parent, contrairement aux boutons précédents qui étaient spécifiques à l'outil. Ainsi, si vous ne sélectionnez

aucune vue, ce sera la vue racine qui sera sélectionnée par défaut. Comme les boutons changent en fonction du composant et du layout parent, je ne vais pas les présenter en détail.

Enfin l'ensemble de boutons de droite, visibles à la figure 5.5.

Figure 5.5 – Les icônes de droite du bas menu

- Le premier bouton permet de modifier l'affichage en fonction d'une résolution que vous choisirez. Très pratique pour tester, si vous n'avez pas tous les terminaux possibles.
- Le deuxième fait en sorte que l'interface graphique fasse exactement la taille de la fenêtre dans laquelle elle se trouve.
- Le suivant remet le zoom à 100%.
- Enfin les deux suivants permettent respectivement de dézoomer et de zoomer.

Rien, jamais rien ne remplacera un test sur un vrai terminal. Ne pensez pas que parce votre interface graphique est esthétique dans cet outil elle le sera aussi en vrai. Si vous n'avez pas de terminal, l'émulateur vous donnera déjà un meilleur aperçu de la situation.

Utilisation

Autant cet outil n'est pas aussi précis, pratique et surtout dénué de bugs que le XML, autant il peut s'avérer pratique pour certaines manipulations de base. Il permet par exemple de modifier les attributs d'une vue à la volée. Sur la figure suivante, vous voyez au centre de la fenêtre une activité qui ne contient qu'un `TextView`. Si vous effectuez un clic droit dessus, vous pourrez voir les différentes options qui se présentent à vous, comme le montre la figure 5.6.

Vous comprendrez plus tard la signification de ces termes, mais retenez bien qu'il est possible de modifier les attributs via un clic droit. Vous pouvez aussi utiliser l'encart `Properties` en bas à droite (voir figure 5.7).

De plus, vous pouvez placer différentes vues en cliquant dessus depuis le menu de gauche, puis en les déposant sur l'activité, comme le montre la figure 5.8.

Il vous est ensuite possible de les agrandir, de les rapetisser ou de les déplacer en fonction de vos besoins, comme le montre la figure 5.9.

Nous allons maintenant voir la véritable programmation graphique. Pour accéder au fichier XML correspondant à votre projet, cliquez sur le deuxième onglet `activity_main.xml`.

Dans la suite du cours, je considérerai le fichier `activity_main.xml` vierge de toute modification, alors si vous avez fait des manipulations vous aurez des différences avec moi.

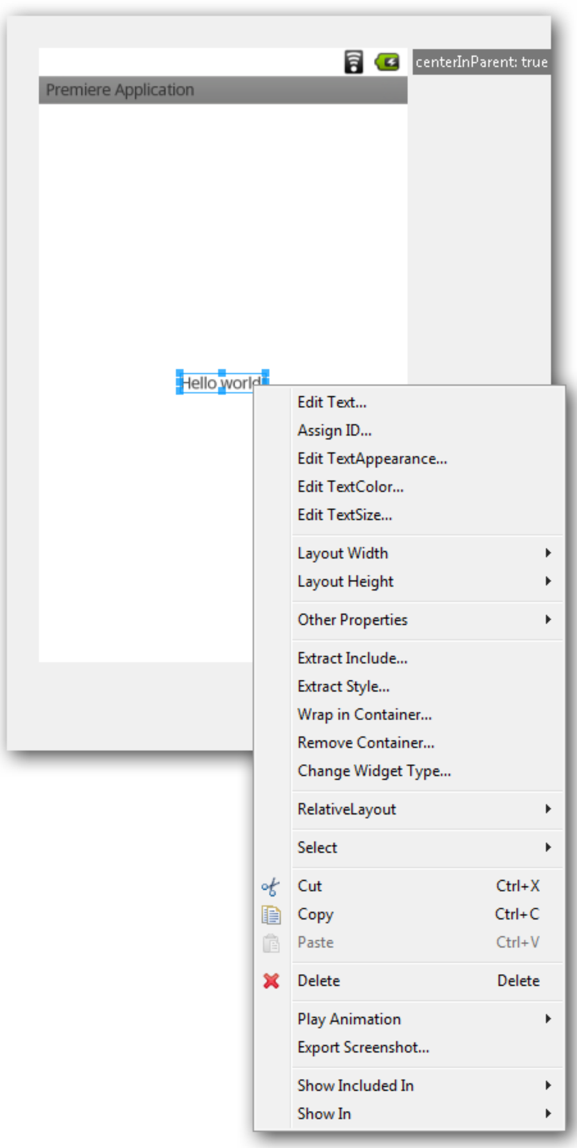


FIGURE 5.6 – Un menu apparaît lors d'un clic droit sur une vue

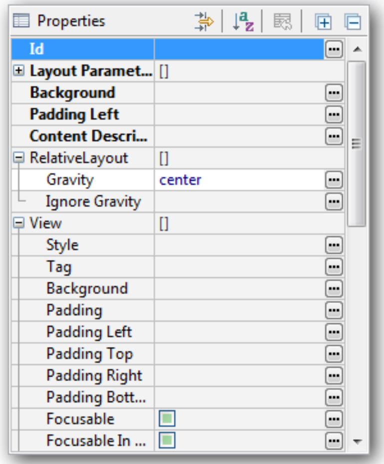


FIGURE 5.7 – L'encart « Properties »

Règles générales sur les vues

Différenciation entre un layout et un widget

Normalement, Eclipse vous a créé un fichier XML par défaut :

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/
    res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:padding="@dimen/padding_medium"
    android:text="@string/hello_world"
    tools:context=".MainActivity" />

</RelativeLayout>
```

La racine possède deux attributs similaires : `xmlns:android="http://schemas.android.com/apk/res/android"` et `xmlns:tools="http://schemas.android.com/tools"`. Ces

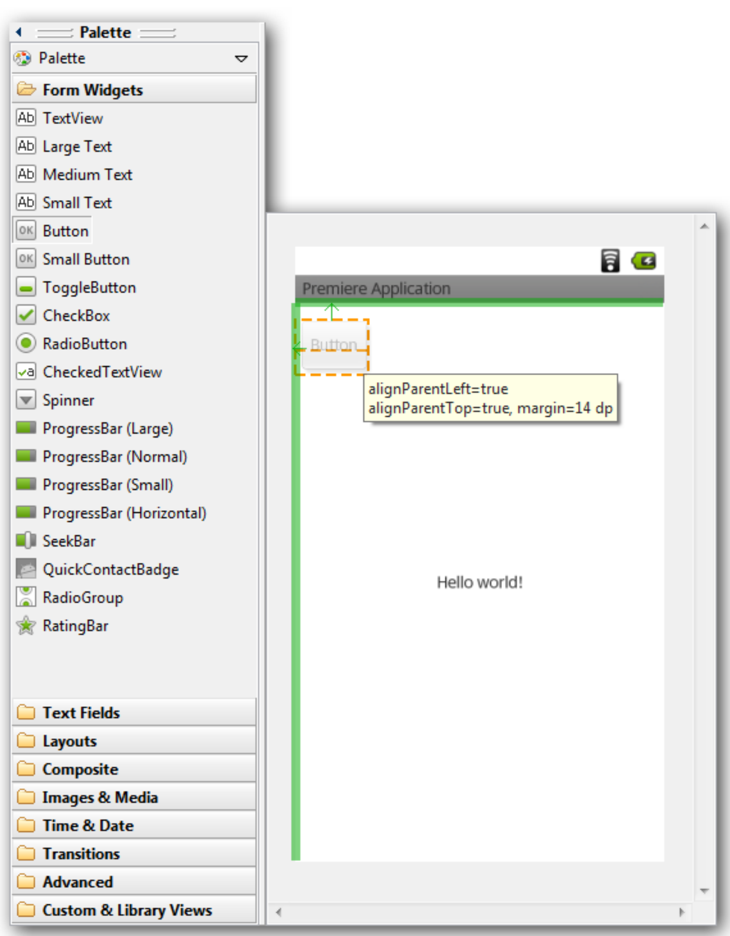


FIGURE 5.8 – Il est possible de faire un cliquer/glisser

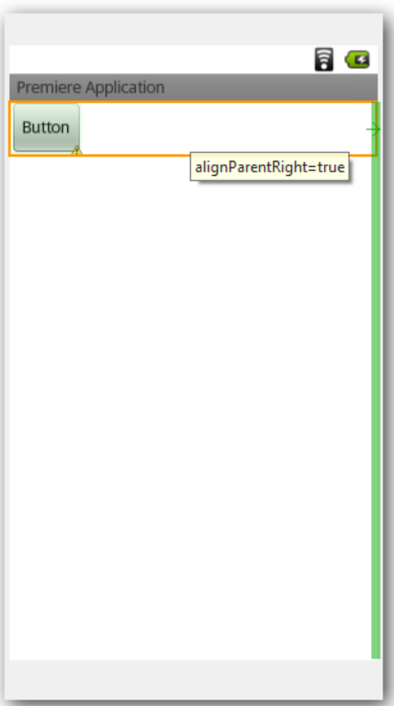


Figure 5.9 – Vous pouvez redimensionner les vues

deux lignes permettent d'utiliser des attributs spécifiques à Android. Si vous ne les mettez pas, vous ne pourrez pas utiliser les attributs et le fichier XML sera un fichier XML banal au lieu d'être un fichier spécifique à Android. De plus, Eclipse refusera de compiler.

On trouve ensuite une racine qui s'appelle `RelativeLayout`. Vous voyez qu'elle englobe un autre nœud qui s'appelle `TextView`. Ah ! Ça vous connaissez ! Comme indiqué précédemment, une interface graphique pour Android est constituée uniquement de vues. Ainsi, tous les nœuds de ces fichiers XML seront des vues.

Revenons à la première vue qui en englobe une autre. Avec **Swing** vous avez déjà rencontré ces objets graphiques qui englobent d'autres objets graphiques. On les appelle en anglais des *layouts* et en français des gabarits. Un layout est donc une vue spéciale qui peut contenir d'autres vues et qui n'est pas destinée à fournir du contenu ou des contrôles à l'utilisateur. Les layouts se contentent de disposer les vues d'une certaine façon. Les vues contenues sont les *enfants*, la vue englobante est le *parent*, comme en XML. Une vue qui ne peut pas en englober d'autres est appelée un *widget* (composant, en français).

Un layout hérite de `ViewGroup` (classe abstraite, qu'on ne peut donc pas instancier), et `ViewGroup` hérite de `View`. Donc quand je dis qu'un `ViewGroup` peut contenir des `View`, c'est qu'il peut aussi contenir d'autres `ViewGroup` !

Vous pouvez bien sûr avoir en racine un simple widget si vous souhaitez que votre mise en page consiste en cet unique widget.

Attributs en commun

Comme beaucoup de nœuds en XML, une vue peut avoir des attributs, qui permettent de moduler certains de ses aspects. Certains de ces attributs sont spécifiques à des vues, d'autres sont communs. Parmi ces derniers, les deux les plus courants sont `layout_width`, qui définit la largeur que prend la vue (la place sur l'axe horizontal), et `layout_height`, qui définit la hauteur qu'elle prend (la place sur l'axe vertical). Ces deux attributs peuvent prendre une valeur parmi les trois suivantes :

- `fill_parent` : signifie qu'elle prendra autant de place que son parent sur l'axe concerné ;
- `wrap_content` : signifie qu'elle prendra le moins de place possible sur l'axe concerné. Par exemple si votre vue affiche une image, elle prendra à peine la taille de l'image, si elle affiche un texte, elle prendra juste la taille suffisante pour écrire le texte ;
- Une valeur numérique précise avec une unité.

Je vous conseille de ne retenir que deux unités :

- `dp` ou `dip` : il s'agit d'une unité qui est indépendante de la résolution de l'écran. En effet, il existe d'autres unités comme le pixel (px) ou le millimètre (`mm`), mais celles-ci varient d'un écran à l'autre… Par exemple si vous mettez une taille de 500 dp pour un widget, il aura toujours la même dimension quelque soit la taille de l'écran. Si

vous mettez une dimension de 500 mm pour un widget, il sera grand pour un grand écran... et énorme pour un petit écran.
- `sp` : cette unité respecte le même principe, sauf qu'elle est plus adaptée pour définir la taille d'une police de caractères.

Depuis l'API 8 (dans ce cours, on travaille sur l'API 7), vous pouvez remplacer `fill_parent` par `match_parent`. Il s'agit d'exactement la même chose, mais en plus explicite.

Il y a quelque chose que je trouve étrange : la racine de notre layout, le nœud `RelativeLayout`, utilise `fill_parent` en largeur et en hauteur. Or, tu nous avais dit que cet attribut signifiait qu'on prenait toute la place du parent... Mais il n'a pas de parent, puisqu'il s'agit de la racine !

C'est parce qu'on ne vous dit pas tout, on vous cache des choses, la vérité est ailleurs. En fait, même notre racine a une vue parent, c'est juste qu'on n'y a pas accès. Cette vue parent invisible prend toute la place possible dans l'écran.

Vous pouvez aussi définir une marge interne pour chaque widget, autrement dit l'espacement entre le contour de la vue et son contenu (voir figure 5.10).

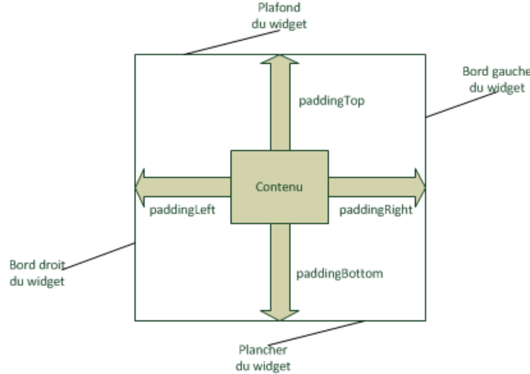


FIGURE 5.10 – Il est possible de définir une marge interne pour chaque widget

Ci-dessous avec l'attribut `android:padding` dans le fichier XML pour définir un carré d'espacement ; la valeur sera suivie d'une unité, 10.5dp par exemple.

```
<TextView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:padding="10.5dp"
  android:text="@string/hello" />
```

La méthode Java équivalente est `public void setPadding (int left, int top, int right, int bottom)`.

```
textView.setPadding(15, 105, 21, 105);
```

En XML on peut aussi utiliser des attributs `android:paddingBottom` pour définir uniquement l'espacement avec le plancher, `android:paddingLeft` pour définir uniquement l'espacement entre le bord gauche du widget et le contenu, `android:paddingRight` pour définir uniquement l'espacement de droite et enfin `android:paddingTop` pour définir uniquement l'espacement avec le plafond.

Identifier et récupérer des vues

Identification

Vous vous rappelez certainement qu'on a dit que certaines ressources avaient un identifiant. Eh bien, il est possible d'accéder à une ressource à partir de son identifiant à l'aide de la syntaxe `@X/Y`. Le `@` signifie qu'on va parler d'un identifiant, le `X` est la classe où se situe l'identifiant dans `R.java` et enfin, le `Y` sera le nom de l'identifiant. Bien sûr, la combinaison `X/Y` doit pointer sur un identifiant qui existe. Reprenons notre classe créée par défaut :

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/
    res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:padding="@dimen/padding_medium"
    android:text="@string/hello_world"
    tools:context=".MainActivity" />

</RelativeLayout>
```

On devine d'après la ligne surlignée que le `TextView` affichera le texte de la ressource qui se trouve dans la classe `String` de `R.java` et qui s'appelle `hello_world`. Enfin, vous vous rappelez certainement aussi que l'on a récupéré des ressources à l'aide de l'identifiant que le fichier `R.java` créait automatiquement dans le chapitre précédent. Si vous allez voir ce fichier, vous constaterez qu'il ne contient aucune mention à nos vues, juste au fichier `activity_main.xml`. Eh bien, c'est tout simplement parce qu'il faut créer cet identifiant nous-mêmes (dans le fichier XML hein, ne modifiez jamais `R.java` par vous-mêmes, malheureux !).

Afin de créer un identifiant, on peut rajouter à chaque vue un attribut `android:id`. La valeur doit être de la forme `@+X/Y`. Le `+` signifie qu'on parle d'un identifiant qui n'est pas encore défini. En voyant cela, Android sait qu'il doit créer un attribut.

La syntaxe `@+X/Y` est aussi utilisée pour faire référence à l'identifiant d'une vue créée plus tard dans le fichier XML.

Le `X` est la classe dans laquelle sera créé l'identifiant. Si cette classe n'existe pas, alors elle sera créée. Traditionnellement, `X` vaut `id`, mais donnez-lui la valeur qui vous plaît. Enfin, le `Y` sera le nom de l'identifiant. Cet identifiant doit être unique au sein de la classe, comme d'habitude.

Par exemple, j'ai décidé d'appeler mon `TextView` « text » et de changer le padding pour qu'il vaille 25.7dp, ce qui nous donne :

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >

  <TextView
    android:id="@+id/text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:padding="25.7dp"
    android:text="@string/hello_world"
    tools:context=".MainActivity" />

</RelativeLayout>
```

Dès que je sauvegarde, mon fichier `R` sera modifié automatiquement :

```
public final class R {
  public static final class attr {
  }
  public static final class dimen {
    public static final int padding_large=0x7f040002;
    public static final int padding_medium=0x7f040001;
    public static final int padding_small=0x7f040000;
  }
  public static final class drawable {
    public static final int ic_action_search=0x7f020000;
    public static final int ic_launcher=0x7f020001;
  }
  public static final class id {
    public static final int menu_settings=0x7f080000;
  }
  public static final class layout {
    public static final int activity_main=0x7f030000;
  }
  public static final class menu {
```

```
20        public static final int activity_main=0x7f070000;
21      }
22      public static final class string {
23        public static final int app_name=0x7f050000;
24        public static final int hello_world=0x7f050001;
25        public static final int menu_settings=0x7f050002;
26        public static final int title_activity_main=0x7f050003;
27      }
28      public static final class style {
29        public static final int AppTheme=0x7f060000;
30      }
31    }
```

Instanciation des objets XML

Enfin, on peut utiliser cet identifiant dans le code, comme avec les autres identifiants. Pour cela, on utilise la méthode `public View findViewById (int id)`. Attention, cette méthode renvoie une `View`, il faut donc la « caster » dans le type de destination.

On caste ? Aucune idée de ce que cela peut vouloir dire !

Petit rappel en ce qui concerne la programmation objet : quand une classe `Classe_1` hérite (ou dérive, on trouve les deux termes) d'une autre classe `Classe_2`, il est possible d'obtenir un objet de type `Classe_1` à partir d'un de `Classe_2` avec le **transtypage**. Pour dire qu'on convertit une classe mère (`Classe_2`) en sa classe fille (`Classe_1`) on dit qu'on **caste** `Classe_2` en `Classe_1`, et on le fait avec la syntaxe suivante :

```
1  //avec « class Class_1 extends Classe_2 »
2  Classe_2 objetDeux = null;
3  Classe_1 objetUn = (Classe_1) objetDeux;
```

Ensuite, et c'est là que tout va devenir clair, vous pourrez déclarer que votre activité utilise comme interface graphique la vue que vous désirez à l'aide de la méthode `void setContentView (View view)`. Dans l'exemple suivant, l'interface graphique est référencée par `R.layout.activity_main`, il s'agit donc du layout d'identifiant `main`, autrement dit celui que nous avons manipulé un peu plus tôt.

```
1  import android.app.Activity;
2  import android.os.Bundle;
3  import android.widget.TextView;
4
5  public class TroimsActivity extends Activity {
6    TextView monTexte = null;
7
8    @Override
9    public void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    monTexte = (TextView)findViewById(R.id.text);
    monTexte.setText("Le texte de notre TextView");
  }
}
```

Je peux tout à fait modifier le padding *a posteriori*.

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class TroimsActivity extends Activity {
  TextView monTexte = null;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    monTexte = (TextView)findViewById(R.id.text);
    // N'oubliez pas que cette fonction n'utilise que des
        entiers
    monTexte.setPadding(50, 60, 70, 90);
  }
}
```

Y a-t-il une raison pour laquelle on accède à la vue après le `setContentView` ?

Oui ! Essayez de le faire avant, votre application va planter.

En fait, à chaque fois qu'on récupère un objet depuis un fichier XML dans notre code Java, on procède à une opération qui s'appelle la **désérialisation**. Concrètement, la désérialisation, c'est transformer un objet qui n'est pas décrit en Java – dans notre cas l'objet est décrit en XML – en un objet Java réel et concret. C'est à cela que sert la fonction `View findViewById (int id)`. Le problème est que cette méthode va aller chercher dans un arbre de vues, qui est créé automatiquement par l'activité. Or, cet arbre ne sera créé qu'après le `setContentView` ! Donc le `findViewById` retournera `null` puisque l'arbre n'existera pas et l'objet ne sera donc pas dans l'arbre. On va à la place utiliser la méthode `static View inflate (Context context, int id, ViewGroup parent)`. Cette méthode va désérialiser l'arbre XML au lieu de l'arbre de vues qui sera créé par l'activité.

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.RelativeLayout;
```

```
4  import android.widget.TextView;
5
6  public class TroimsActivity extends Activity {
7    RelativeLayout layout = null;
8    TextView text = null;
9
10   @Override
11   public void onCreate(Bundle savedInstanceState) {
12     super.onCreate(savedInstanceState);
13
14     // On récupère notre layout par désérialisation. La méthode
          inflate retourne un View
15     // C'est pourquoi on caste (on convertit) le retour de la m
          éthode avec le vrai type de notre layout, c'est-à-dire
          RelativeLayout
16     layout = (RelativeLayout) RelativeLayout.inflate(this, R.
          layout.activity_main, null);
17     // ... puis on récupère TextView grâce à son identifiant
18     text = (TextView) layout.findViewById(R.id.text);
19     text.setText("Et cette fois, ça fonctionne !");
20     setContentView(layout);
21     // On aurait très bien pu utiliser « setContentView(R.
          layout.activity_main) » bien sûr !
22   }
23 }
```

C'est un peu contraignant ! Et si on se contentait de faire un premier `setContentView` pour « inflater » (désérialiser) l'arbre et récupérer la vue pour la mettre dans un second `setContentView` ?

Un peu comme cela, voulez-vous dire ?

```
1  import android.app.Activity;
2  import android.os.Bundle;
3  import android.widget.TextView;
4
5  public class TroimsActivity extends Activity {
6    TextView monTexte = null;
7
8    @Override
9    public void onCreate(Bundle savedInstanceState) {
10     super.onCreate(savedInstanceState);
11     setContentView(R.layout.activity_main);
12
13     monTexte = (TextView)findViewById(R.id.text);
14     monTexte.setPadding(50, 60, 70, 90);
15
16     setContentView(R.layout.activity_main);
17   }
```

```
18 }
```

Ah d'accord, comme cela l'arbre sera *inflate* et on n'aura pas à utiliser la méthode compliquée vue au-dessus…

C'est une idée… mais je vous répondrais que vous avez oublié l'optimisation ! Un fichier XML est très lourd à parcourir, donc construire un arbre de vues prend du temps et des ressources. À la compilation, si on détecte qu'il y a deux `setContentView` dans `onCreate`, eh bien on ne prendra en compte que la dernière ! Ainsi, toutes les instances de `setContentView` précédant la dernière sont rendues caduques.

En résumé

- Eclipse vous permet de confectionner des interfaces à la souris, mais cela ne sera jamais aussi précis que de travailler directement dans le code.
- Tous les layouts héritent de la super classe `ViewGroup` qui elle même hérite de la super classe `View`. Puisque les widgets héritent aussi de `View` et que les `ViewGroup` peuvent contenir des `View`, les layouts peuvent contenir d'autres layouts et des widgets. C'est là toute la puissance de la hiérarchisation et la confection des interfaces.
- `View` regroupe un certain nombre de propriétés qui deviennent communes aux widgets et aux layouts.
- Lorsque vous désérialisez (ou inflatez) un layout dans une activité, vous devez récupérer les widgets et les layouts pour lesquels vous désirez rajouter des fonctionnalités. Cela se fait grâce à la classe `R.java` qui liste l'ensemble des identifiants de vos ressources.

Chapitre 6

Les widgets les plus simples

Difficulté :

Maintenant qu'on sait comment est construite une interface graphique, on va voir avec quoi il est possible de la peupler. Ce chapitre traitera uniquement des widgets, c'est-à-dire des vues qui *fournissent* un contenu et non qui le *mettent en forme* — ce sont les layouts qui s'occupent de ce genre de choses.

Fournir un contenu, c'est permettre à l'utilisateur d'interagir avec l'application, ou afficher une information qu'il est venu consulter.

Les widgets

Un widget est un élément de base qui permet d'afficher du contenu à l'utilisateur ou lui permet d'interagir avec l'application. Chaque widget possède un nombre important d'attributs XML et de méthodes Java, c'est pourquoi je ne les détaillerai pas, mais vous pourrez trouver toutes les informations dont vous avez besoin sur la documentation officielle d'Android (cela tombe bien, j'en parle à la fin du chapitre).

`TextView`

Vous connaissez déjà cette vue, elle vous permet d'afficher une chaîne de caractères que l'utilisateur ne peut modifier. Vous verrez plus tard qu'on peut aussi y insérer des chaînes de caractères formatées, à l'aide de balises HTML, ce qui nous servira à souligner du texte ou à le mettre en gras par exemple.

Exemple en XML

```
1 <TextView
2   android:layout_width="fill_parent"
3   android:layout_height="wrap_content"
4   android:text="@string/textView"
5   android:textSize="8sp"
6   android:textColor="#112233" />
```

Vous n'avez pas encore vu comment faire, mais cette syntaxe `@string/textView` signifie qu'on utilise une ressource de type `string`. Il est aussi possible de passer directement une chaîne de caractères dans `android:text`, mais ce n'est pas recommandé. On précise également la taille des caractères avec `android:textSize`, puis on précise la couleur du texte avec `android:textColor`. Cette notation avec un `#` permet de décrire des couleurs à l'aide de nombres hexadécimaux.

Exemple en Java

```
1 TextView textView = new TextView(this);
2 textView.setText(R.string.textView);
3 textView.setTextSize(8);
4 textView.setTextColor(0x112233);
```

Vous remarquerez que l'équivalent de `#112233` est `0x112233` (il suffit de remplacer le `#` par `0x`).

Rendu

Le rendu se trouve à la figure 6.1.

TextView

Figure 6.1 – Rendu d'un TextView

EditText

Ce composant est utilisé pour permettre à l'utilisateur d'écrire des textes. Il s'agit en fait d'un `TextView` éditable.

Il hérite de `TextView`, ce qui signifie qu'il peut prendre les mêmes attributs que `TextView` en XML et qu'on peut utiliser les mêmes méthodes Java.

Exemple en XML

```
1 <EditText
2   android:layout_width="fill_parent"
3   android:layout_height="wrap_content"
4   android:hint="@string/editText"
5   android:inputType="textMultiLine"
6   android:lines="5" />
```

- Au lieu d'utiliser `android:text`, on utilise `android:hint`. Le problème avec le premier est qu'il remplit l'`EditText` avec le texte demandé, alors qu'`android:hint` affiche juste un texte d'indication, qui n'est pas pris en compte par l'`EditText` en tant que valeur (si vous avez du mal à comprendre la différence, essayez les deux).
- On précise quel type de texte contiendra notre `EditText` avec `android:inputType`. Dans ce cas précis un texte sur plusieurs lignes. Cet attribut change la nature du clavier qui est proposé à l'utilisateur, par exemple si vous indiquez que l'`EditText` servira à écrire une adresse e-mail, alors l'arobase sera proposé tout de suite à l'utilisateur sur le clavier. Vous trouverez une liste de tous les `inputTypes` possibles sur la documentation.
- On peut préciser la taille en lignes que doit occuper l'`EditText` avec `android:lines`.

▷ Les types d'`inputTypes`
Code web : 419933

Exemple en Java

```
1 EditText editText = new EditText(this);
2 editText.setHint(R.string.editText);
3 editText.setInputType(InputType.TYPE_TEXT_FLAG_MULTI_LINE);
4 editText.setLines(5);
```

Rendu

Le rendu se trouve à la figure 6.2.

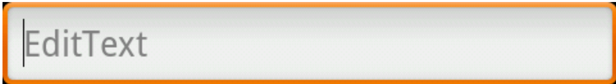


FIGURE 6.2 – Rendu d'un EditText

`Button`

Un simple bouton, même s'il s'agit en fait d'un `TextView` cliquable.

Il hérite de `TextView`, ce qui signifie qu'il peut prendre les mêmes attributs que `TextView` en XML et qu'on peut utiliser les mêmes méthodes Java.

Exemple en XML

```
<Button
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="@string/button" />
```

Exemple en Java

```
Button button = new Button(this);
editText.setText(R.string.button);
```

Rendu

Le rendu se trouve à la figure 6.3.

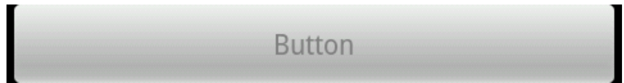


FIGURE 6.3 – Rendu d'un Button

`CheckBox`

Une case qui peut être dans deux états : cochée ou pas.

Elle hérite de Button, ce qui signifie qu'elle peut prendre les mêmes attributs que Button en XML et qu'on peut utiliser les mêmes méthodes Java.

Exemple en XML

```
<CheckBox
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="@string/checkBox"
  android:checked="true" />
```

android:checked="true" signifie que la case est cochée par défaut.

Exemple en Java

```
CheckBox checkBox = new CheckBox(this);
checkBox.setText(R.string.checkBox);
checkBox.setChecked(true)
if(checkBox.isChecked())
  // Faire quelque chose si le bouton est coché
```

Rendu

Le rendu se trouve à la figure 6.4.

FIGURE 6.4 – Rendu d'une CheckBox : cochée à gauche, non cochée à droite

RadioButton et RadioGroup

Même principe que la CheckBox, à la différence que l'utilisateur ne peut cocher qu'une seule case. Il est plutôt recommandé de les regrouper dans un RadioGroup.

RadioButton hérite de Button, ce qui signifie qu'il peut prendre les mêmes attributs que Button en XML et qu'on peut utiliser les mêmes méthodes Java.

Un RadioGroup est en fait un layout, mais il n'est utilisé qu'avec des RadioButton, c'est pourquoi on le voit maintenant. Son but est de faire en sorte qu'il puisse n'y avoir qu'un seul RadioButton sélectionné dans tout le groupe.

Exemple en XML

```
<RadioGroup
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:orientation="horizontal" >
  <RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true" />
  <RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
  <RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</RadioGroup>
```

Exemple en Java

```
RadioGroup radioGroup = new RadioGroup(this);
RadioButton radioButton1 = new RadioButton(this);
RadioButton radioButton2 = new RadioButton(this);
RadioButton radioButton3 = new RadioButton(this);

// On ajoute les boutons au RadioGroup
radioGroup.addView(radioButton1, 0);
radioGroup.addView(radioButton2, 1);
radioGroup.addView(radioButton3, 2);

// On sélectionne le premier bouton
radioGroup.check(0);

// On récupère l'identifiant du bouton qui est coché
int id = radioGroup.getCheckedRadioButtonId();
```

Rendu

Le rendu se trouve à la figure 6.5.

FIGURE 6.5 – Le bouton radio de droite est sélectionné

Utiliser la documentation pour trouver une information

Je fais un petit aparté afin de vous montrer comment utiliser la documentation pour trouver les informations que vous recherchez, parce que tout le monde en a besoin. Que ce soit vous, moi, des développeurs Android professionnels ou n'importe qui chez

Google, nous avons tous besoin de la documentation. Il n'est pas possible de tout savoir, et surtout, je ne peux pas tout vous dire ! La documentation est là pour ça, et vous ne pourrez pas devenir de bons développeurs Android — voire de bons développeurs tout court — si vous ne savez pas chercher des informations par vous-mêmes.

Je vais procéder à l'aide d'un exemple. Je me demande comment faire pour changer la couleur du texte de ma `TextView`. Pour cela, je me dirige vers la documentation officielle : `http://developer.android.com`.

Vous voyez un champ de recherche en haut à gauche. Je vais insérer le nom de la classe que je recherche : `TextView`. Vous voyez une liste qui s'affiche et qui permet de sélectionner la classe qui pourrait éventuellement vous intéresser, comme à la figure 6.6.

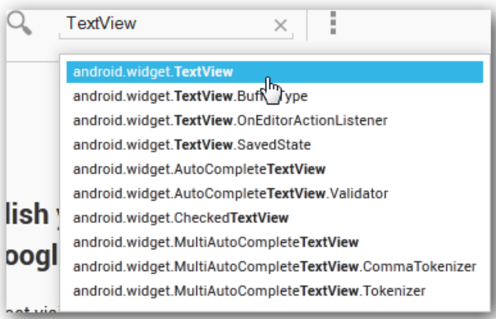


FIGURE 6.6 – Une liste s'affiche afin que vous sélectionniez ce qui vous intéresse

J'ai bien sûr cliqué sur `Android.widget.TextView` puisque c'est celle qui m'intéresse. Nous arrivons alors sur une page qui vous décrit toutes les informations possibles et imaginables sur la classe `TextView` (voir figure 6.7).

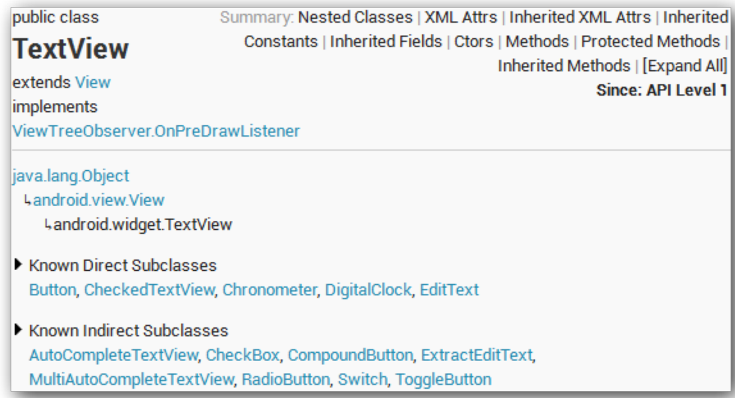


FIGURE 6.7 – Vous avez accès à beaucoup d'informations sur la classe

On voit par exemple qu'il s'agit d'une classe, publique, qui dérive de `View` et implémente une interface.

La partie suivante représente un arbre qui résume la hiérarchie de ses superclasses.

Ensuite, on peut voir les classes qui dérivent directement de cette classe (`Known Direct Subclasses`) et les classes qui en dérivent indirectement, c'est-à-dire qu'un des ancêtres de ces classes dérive de `View` (`Known Indirect Subclasses`).

Enfin, on trouve en haut à droite un résumé des différentes sections qui se trouvent dans le document (je vais aussi parler de certaines sections qui ne se trouvent pas dans cette classe mais que vous pourrez rencontrer dans d'autres classes) :

– `Nested Classes` est la section qui regroupe toutes les classes internes. Vous pouvez cliquer sur une classe interne pour ouvrir une page similaire à celle de la classe `View`.
– `XML Attrs` est la section qui regroupe tous les attributs que peut prendre un objet de ce type en XML. Allez voir le tableau, vous verrez que pour chaque attribut XML on trouve associé un équivalent Java.
– `Constants` est la section qui regroupe toutes les constantes dans cette classe.
– `Fields` est la section qui regroupe toutes les structures de données constantes dans cette classe (listes et tableaux).
– `Ctors` est la section qui regroupe tous les constructeurs de cette classe.
– `Methods` est la section qui regroupe toutes les méthodes de cette classe.
– `Protected Methods` est la section qui regroupe toutes les méthodes protégées (accessibles uniquement par cette classe ou les enfants de cette classe).

Vous rencontrerez plusieurs fois l'adjectif `Inherited`, il signifie que cet attribut ou classe a hérité d'une de ses superclasses.

Ainsi, si je cherche un attribut XML, je peux cliquer sur `XML Attrs` et parcourir la liste des attributs pour découvrir celui qui m'intéresse (voir figure 6.8), ou alors je peux effectuer une recherche sur la page (le raccourci standard pour cela est `Ctrl` + `F`).

J'ai trouvé ! Il s'agit de `android:textColor` ! Je peux ensuite cliquer dessus pour obtenir plus d'informations et ainsi l'utiliser correctement dans mon code.

Calcul de l'IMC - Partie 1

Énoncé

On va commencer un mini-TP (TP signifie « travaux pratiques » ; ce sont des exercices pour vous entraîner à programmer). Vous voyez ce qu'est l'IMC (Indice de Masse Corporelle) ? C'est un nombre qui se calcule à partir de la taille et de la masse corporelle d'un individu, afin qu'il puisse déterminer s'il est trop svelte ou trop corpulent.

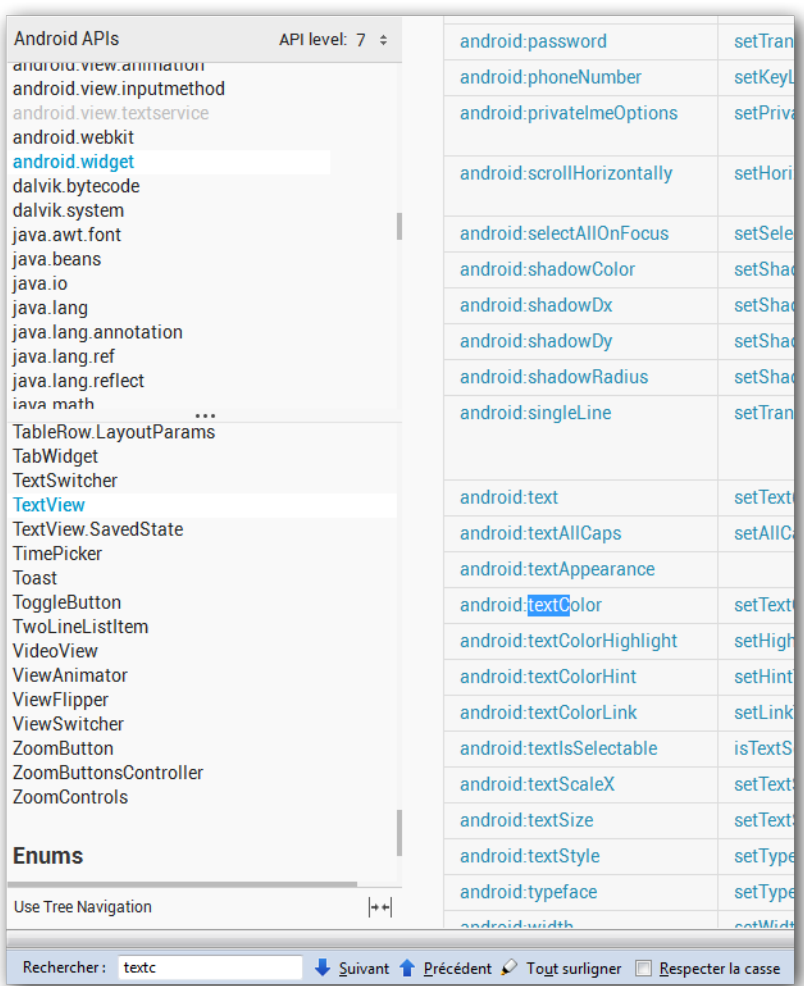


FIGURE 6.8 – Apprenez à utiliser les recherches

Ayant travaillé dans le milieu médical, je peux vous affirmer qu'il ne faut pas faire trop confiance à ce nombre (c'est pourquoi je ne propose pas d'interprétation du résultat pour ce mini-TP). S'il vous indique que vous êtes en surpoids, ne complexez pas ! Sachez que tous les *bodybuilders* du monde se trouvent obèses d'après ce nombre.

Pour l'instant, on va se contenter de faire l'interface graphique. Elle ressemblera à la figure 6.9.

FIGURE 6.9 – Notre programme ressemblera à ça

Instructions

Avant de commencer, voici quelques instructions :

- On utilisera uniquement le XML.
- Pour mettre plusieurs composants dans un layout, on se contentera de mettre les composants entre les balises de ce layout.
- On n'utilisera qu'un seul layout.
- Les deux `EditText` permettront de n'insérer que des nombres. Pour cela, on utilise l'attribut `android:inputType` auquel on donne la valeur `numbers`.
- Les `TextView` qui affichent « Poids : » et « Taille : » sont centrés, en rouge et en gras.
- Pour mettre un `TextView` en gras on utilisera l'attribut `android:textStyle` en lui attribuant comme valeur `bold`.
- Pour mettre un `TextView` en rouge on utilisera l'attribut `android:textColor` en lui attribuant comme valeur `#FF0000`.
- Afin de centrer du texte dans un `TextView`, on utilise l'attribut `android:gravity="center"`.

Voici le layout de base :

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical">

  <!-- mettre les composants ici -->

</LinearLayout>
```

Solution

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical">
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Poids : "
    android:textStyle="bold"
    android:textColor="#FF0000"
    android:gravity="center"
  />
  <EditText
    android:id="@+id/poids"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="Poids"
    android:inputType="numberDecimal"
  />
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Taille : "
    android:textStyle="bold"
    android:textColor="#FF0000"
    android:gravity="center"
  />
  <EditText
    android:id="@+id/taille"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="Taille"
    android:inputType="numberDecimal"
  />
  <RadioGroup
```

```
37        android:id="@+id/group"
38        android:layout_width="wrap_content"
39        android:layout_height="wrap_content"
40        android:checkedButton="@+id/radio2"
41        android:orientation="horizontal"
42      >
43        <RadioButton
44          android:id="@+id/radio1"
45          android:layout_width="wrap_content"
46          android:layout_height="wrap_content"
47          android:text="Mètre"
48        />
49        <RadioButton
50          android:id="@+id/radio2"
51          android:layout_width="wrap_content"
52          android:layout_height="wrap_content"
53          android:text="Centimètre"
54        />
55      </RadioGroup>
56      <CheckBox
57        android:id="@+id/mega"
58        android:layout_width="wrap_content"
59        android:layout_height="wrap_content"
60        android:text="Mega fonction !"
61      />
62      <Button
63        android:id="@+id/calcul"
64        android:layout_width="wrap_content"
65        android:layout_height="wrap_content"
66        android:text="Calculer l'IMC"
67      />
68      <Button
69        android:id="@+id/raz"
70        android:layout_width="wrap_content"
71        android:layout_height="wrap_content"
72        android:text="RAZ"
73      />
74      <TextView
75        android:layout_width="wrap_content"
76        android:layout_height="wrap_content"
77        android:text="Résultat:"
78      />
79      <TextView
80        android:id="@+id/result"
81        android:layout_width="fill_parent"
82        android:layout_height="fill_parent"
83        android:text="Vous devez cliquer sur le bouton « Calculer l
             'IMC » pour obtenir un résultat."
84      />
85    </LinearLayout>
```

Vous pouvez copier ce code grâce au code web suivant :

▷ Copier ce code
Code web : 974282

Et voilà, notre interface graphique est prête ! Bon pour le moment, elle ne fait rien : si vous appuyez sur les différents élements, rien ne se passe. Mais nous allons y remédier d'ici peu, ne vous inquiétez pas.

Gérer les évènements sur les widgets

On va voir ici comment gérer les interactions entre l'interface graphique et l'utilisateur.

Les listeners

Il existe plusieurs façons d'interagir avec une interface graphique. Par exemple cliquer sur un bouton, entrer un texte, sélectionner une portion de texte, etc. Ces interactions s'appellent des **évènements**. Pour pouvoir réagir à l'apparition d'un évènement, il faut utiliser un objet qui va détecter l'évènement et afin de vous permettre le traiter. Ce type d'objet s'appelle un **listener**. Un listener est une interface qui vous oblige à redéfinir des méthodes de `callback` et chaque méthode sera appelée au moment où se produira l'évènement associé.

Par exemple, pour intercepter l'évènement `clic` sur un `Button`, on appliquera l'interface `View.OnClickListener` sur ce bouton. Cette interface contient la méthode de `callback void onClick(View vue)` — le paramètre de type `View` étant la vue sur laquelle le clic a été effectué, qui sera appelée à chaque clic et qu'il faudra implémenter pour déterminer que faire en cas de clic. Par exemple pour gérer d'autres évènements, on utilisera d'autres méthodes (liste non exhaustive) :

- `View.OnLongClickListener` pour les clics qui durent longtemps, avec la méthode `boolean onLongClick(View vue)`. Cette méthode doit retourner `true` une fois que l'action associée a été effectuée.
- `View.OnKeyListener` pour gérer l'appui sur une touche. On y associe la méthode `boolean onKey(View vue, int code, KeyEvent event)`. Cette méthode doit retourner `true` une fois que l'action associée a été effectuée.

J'ai bien dit qu'il fallait utiliser `View.OnClickListener`, de la classe `View` ! Il existe d'autres types de `OnClickListener` et Eclipse pourrait bien vous proposer d'importer n'importe quel package qui n'a rien à voir, auquel cas votre application ne fonctionnerait pas. Le package à utiliser pour `OnClickListener` est `android.view.View.OnClickListener`.

Que veux-tu dire par « Cette méthode doit retourner `true` une fois que l'action associée a été effectuée » ?

Petite subtilité pas forcément simple à comprendre. Il faut indiquer à Android quand vous souhaitez que l'évènement soit considéré comme traité, achevé. En effet, il est possible qu'un évènement continue à agir dans le temps. Un exemple simple est celui du toucher. Le toucher correspond au fait de toucher l'écran, pendant que vous touchez l'écran et avant même de lever le doigt pour le détacher de l'écran. Si vous levez ce doigt, le toucher s'arrête et un nouvel évènement est lancé : le clic, mais concentrons-nous sur le toucher. Quand vous touchez l'écran, un évènement de type `onTouch` est déclenché. Si vous retournez `true` au terme de cette méthode, ça veut dire que cet évènement *toucher* a été géré, et donc si l'utilisateur continue à bouger son doigt sur l'écran, Android considérera les mouvements sont de nouveaux évènements *toucher* et à nouveaux la méthode de *callback* `onTouch` sera appelée pour chaque mouvement. En revanche, si vous retournez `false`, l'évènement ne sera pas considéré comme terminé et si l'utilisateur continue à bouger son doigt sur l'écran, Android ne considérera pas que ce sont de nouveaux évènements et la méthode `onTouch` ne sera plus appelée. Il faut donc réfléchir en fonction de la situation.

Enfin pour associer un listener à une vue, on utilisera une méthode du type `setOn[Evenement]Listener(On[Evenenement]Listener listener)` avec `Evenement` l'évènement concerné, par exemple pour détecter les clics sur un bouton on fera :

```
1 Bouton b = new Button(getContext());
2 b.setOnClickListener(notre_listener);
```

Par héritage

On va faire implémenter un listener à notre classe, ce qui veut dire que l'activité interceptera d'elle-même les évènements. N'oubliez pas que lorsqu'on implémente une interface, il faut nécessairement implémenter toutes les méthodes de cette interface. Enfin, il n'est bien entendu pas indispensable que vous gériez tous les évènements d'une interface, vous pouvez laisser une méthode vide si vous ne voulez pas vous préoccuper de ce style d'évènements.

Un exemple d'implémentation :

```
1  import android.view.View.OnTouchListener;
2  import android.view.View.OnClickListener;
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.view.MotionEvent;
6  import android.view.View;
7  import android.widget.Button;
8
9  // Notre activité détectera les touchers et les clics sur les
       vues qui se sont inscrites
10 public class Main extends Activity implements View.
       OnTouchListener, View.OnClickListener {
11   private Button b = null;
12
13   @Override
```

```
14    public void onCreate(Bundle savedInstanceState) {
15      super.onCreate(savedInstanceState);
16
17      setContentView(R.layout.main);
18
19      b = (Button) findViewById(R.id.boutton);
20      b.setOnTouchListener(this);
21      b.setOnClickListener(this);
22    }
23
24    @Override
25    public boolean onTouch(View v, MotionEvent event) {
26      /* Réagir au toucher */
27      return true;
28    }
29
30    @Override
31    public void onClick(View v) {
32      /* Réagir au clic */
33    }
34  }
```

Cependant, un problème se pose. À chaque fois qu'on appuiera sur un bouton, quel qu'il soit, on rentrera dans la même méthode, et on exécutera donc le même code... C'est pas très pratique, si nous avons un bouton pour rafraîchir un onglet dans une application de navigateur internet et un autre pour quitter un onglet, on aimerait bien que cliquer sur le bouton de rafraîchissement ne quitte pas l'onglet et vice-versa. Heureusement, la vue passée dans la méthode `onClick(View)` permet de différencier les boutons. En effet, il est possible de récupérer l'identifiant de la vue (vous savez, l'identifiant défini en XML et qu'on retrouve dans le fichier `R` !) sur laquelle le clic a été effectué. Ainsi, nous pouvons réagir différemment en fonction de cet identifiant :

```
1  public void onClick(View v) {
2    // On récupère l'identifiant de la vue, et en fonction de cet
         identifiant...
3    switch(v.getId()) {
4
5      // Si l'identifiant de la vue est celui du premier bouton
6      case R.id.bouton1:
7      /* Agir pour bouton 1 */
8      break;
9
10     // Si l'identifiant de la vue est celui du deuxième bouton
11     case R.id.bouton2:
12     /* Agir pour bouton 2 */
13     break;
14
15     /* etc. */
16   }
17 }
```

Par une classe anonyme

L'inconvénient principal de la technique précédente est qu'elle peut très vite allonger les méthodes des listeners, ce qui fait qu'on s'y perd un peu s'il y a beaucoup d'éléments à gérer. C'est pourquoi il est préférable de passer par une classe anonyme dès qu'on a un nombre élevé d'éléments qui réagissent au même évènement.

Pour rappel, une classe anonyme est une classe interne qui dérive d'une superclasse ou implémente une interface, et dont on ne précise pas le nom. Par exemple pour créer une classe anonyme qui implémente `View.OnClickListener()` je peux faire :

```
widget.setTouchListener(new View.OnTouchListener() {
  /**
   * Contenu de ma classe
   * Comme on implémente une interface, il y aura des méthodes
       à implémenter, dans ce cas-ci
   * « public boolean onTouch(View v, MotionEvent event) »
  */
}); // Et on n'oublie pas le point-virgule à la fin ! C'est une
    instruction comme les autres !
```

Voici un exemple de code :

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class AnonymousExampleActivity extends Activity {
  // On cherchera à détecter les touchers et les clics sur ce
     bouton
  private Button touchAndClick = null;
  // On voudra détecter uniquement les clics sur ce bouton
  private Button clickOnly = null;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    touchAndClick = (Button)findViewById(R.id.touchAndClick);
    clickOnly = (Button)findViewById(R.id.clickOnly);

    touchAndClick.setOnLongClickListener(new View.
        OnLongClickListener() {
      @Override
      public boolean onLongClick(View v) {
        // Réagir à un long clic
        return false;
      }
    });
```

```
27
28      touchAndClick.setOnClickListener(new View.OnClickListener()
            {
29        @Override
30        public void onClick(View v) {
31          // Réagir au clic
32        }
33      });
34
35      clickOnly.setOnClickListener(new View.OnClickListener() {
36        @Override
37        public void onClick(View v) {
38          // Réagir au clic
39        }
40      });
41    }
42  }
```

Par un attribut

C'est un dérivé de la méthode précédente : en fait on implémente des classes anonymes dans des attributs de façon à pouvoir les utiliser dans plusieurs éléments graphiques différents qui auront la même réaction pour le même évènement. C'est la méthode que je privilégie dès que j'ai, par exemple, plusieurs boutons qui utilisent le même code.

```
1   import android.app.Activity;
2   import android.os.Bundle;
3   import android.view.MotionEvent;
4   import android.view.View;
5   import android.widget.Button;
6
7   public class Main extends Activity {
8     private OnClickListener clickListenerBoutons = new View.
          OnClickListener() {
9       @Override
10      public void onClick(View v) {
11        /* Réagir au clic pour les boutons 1 et 2*/
12      }
13    };
14
15    private OnTouchListener touchListenerBouton1 = new View.
          OnTouchListener() {
16      @Override
17      public boolean onTouch(View v, MotionEvent event) {
18        /* Réagir au toucher pour le bouton 1*/
19        return onTouch(v, event);
20      }
21    };
22
```

```
23    private OnTouchListener touchListenerBouton3 = new View.
        OnTouchListener() {
24      @Override
25      public boolean onTouch(View v, MotionEvent event) {
26        /* Réagir au toucher pour le bouton 3*/
27        return super.onTouch(v, event);
28      }
29    };
30
31    Button b1 = null;
32    Button b2 = null;
33    Button b3 = null;
34
35    @Override
36    public void onCreate(Bundle savedInstanceState) {
37      super.onCreate(savedInstanceState);
38
39      setContentView(R.layout.main);
40
41      b1 = (Button) findViewById(R.id.bouton1);
42      b2 = (Button) findViewById(R.id.bouton2);
43      b3 = (Button) findViewById(R.id.bouton3);
44
45      b1.setOnTouchListener(touchListenerBouton1);
46      b1.setOnClickListener(clickListenerBoutons);
47      b2.setOnClickListener(clickListenerBoutons);
48      b3.setOnTouchListener(touchListenerBouton3);
49    }
50  }
```

Application

Énoncé

On va s'amuser un peu : nous allons créer un bouton qui prend tout l'écran et faire en sorte que le texte à l'intérieur du bouton grossisse quand on s'éloigne du centre du bouton, et rétrécisse quand on s'en rapproche.

Instructions

- On va se préoccuper non pas du clic mais du toucher, c'est-à-dire l'évènement qui débute dès qu'on touche le bouton jusqu'au moment où on le relâche (contrairement au clic qui ne se déclenche qu'au moment où on relâche la pression).
- La taille du `TextView` sera fixée avec la méthode `setTextSize(Math.abs(coordonnee_x - largeur_du_bouton / 2) + Math.abs(coordonnee_y - hauteur_du_bouton / 2))`.

- Pour obtenir la coordonnée en abscisse (X) on utilise `float getX()` d'un `MotionEvent`, et pour obtenir la coordonnée en ordonnée (Y) on utilise `float getY()`.

Je vous donne le code pour faire en sorte d'avoir le bouton bien au milieu du layout :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
  <Button
    android:id="@+id/bouton"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_gravity="center"
    android:text="@string/hello" />
</LinearLayout>
```

Maintenant, c'est à vous de jouer !

Solution

```
// On fait implémenter OnTouchListener par notre activité
public class Main extends Activity implements View.
    OnTouchListener {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    // On récupère le bouton par son identifiant
    Button b = (Button) findViewById(R.id.bouton);
    // Puis on lui indique que cette classe sera son listener
        pour l'évènement Touch
    b.setOnTouchListener(this);
  }

  // Fonction qui sera lancée à chaque fois qu'un toucher est d
      étecté sur le bouton rattaché
  @Override
  public boolean onTouch(View view, MotionEvent event) {
    // Comme l'évènement nous donne la vue concernée par le
        toucher, on le récupère et on le caste en Button
    Button bouton = (Button)view;

    // On récupère la largeur du bouton
    int largeur = bouton.getWidth();
    // On récupère la hauteur du bouton
```

```
24      int hauteur = bouton.getHeight();
25
26      // On récupère la coordonnée sur l'abscisse (X) de l'évè
            nement
27      float x = event.getX();
28      // On récupère la coordonnée sur l'ordonnée (Y) de l'évè
            nement
29      float y = event.getY();
30
31      // Puis on change la taille du texte selon la formule
            indiquée dans l'énoncé
32      bouton.setTextSize(Math.abs(x - largeur / 2) + Math.abs(y -
             hauteur / 2));
33      // Le toucher est fini, on veut continuer à détecter les
            touchers d'après
34      return true;
35    }
36  }
```

On a procédé par héritage puisqu'on a qu'un seul bouton sur lequel agir.

Calcul de l'IMC - Partie 2

Énoncé

Il est temps maintenant de relier tous les boutons de notre application pour pouvoir effectuer tous les calculs, en respectant les quelques règles suivantes :

- La `CheckBox` de megafonction permet de changer le résultat du calcul en un message élogieux pour l'utilisateur.
- La formule pour calculer l'IMC est $\frac{\text{poids (en kilogrammes)}}{\text{taille (en metres)}^2}$.
- Le bouton `RAZ` remet à zéro tous les champs (sans oublier le texte pour le résultat).
- Les éléments dans le `RadioGroup` permettent à l'utilisateur de préciser en quelle unité il a indiqué sa taille. Pour obtenir la taille en mètres depuis la taille en centimètres il suffit de diviser par 100 : $\frac{171 \text{ centimetres}}{100} = 1.71 \text{ metres}$.
- Dès qu'on change les valeurs dans les champs `Poids` et `Taille`, on remet le texte du résultat par défaut puisque la valeur calculée n'est plus valable pour les nouvelles valeurs.
- On enverra un message d'erreur si l'utilisateur essaie de faire le calcul avec une taille égale à zéro grâce à un `Toast`.

Un `Toast` est un widget un peu particulier qui permet d'afficher un message à n'importe quel moment sans avoir à créer de vue. Il est destiné à informer l'utilisateur sans le déranger outre mesure ; ainsi l'utilisateur peut continuer à utiliser l'application comme si le `Toast` n'était pas présent.

Consignes

- La syntaxe pour construire un `Toast` est `static Toast makeText(Context context, CharSequence texte, int duration)`. La durée peut être indiquée à l'aide de la constante `Toast.LENGTH_SHORT` pour un message court et `Toast.LENGTH_LONG` pour un message qui durera plus longtemps. Enfin, il est possible d'afficher le `Toast` avec la méthode `void show ()`.
- Pour savoir si une `CheckBox` est sélectionnée, on utilisera `boolean isChecked()` qui renvoie `true` le cas échéant.
- Pour récupérer l'identifiant du `RadioButton` qui est sélectionné dans un `RadioGroup` il faut utiliser la méthode `int getCheckedRadioButtonId ()`.
- On peut récupérer le texte d'un `EditText` à l'aide de la fonction `Editable getText ()`. On peut ensuite vider le contenu de cet objet `Editable` à l'aide de la fonction `void clear()`.
- Parce que c'est déjà bien assez compliqué comme cela, on se simplifie la vie et on ne prend pas en compte les cas extrêmes (taille ou poids < 0 ou `null` par exemple).
- Pour détecter le moment où l'utilisateur écrit dans un `EditText`, on peut utiliser l'évènement `onKey`. Problème, cette technique ne fonctionne que sur les claviers virtuels, alors si l'utilisateur a un clavier physique, ce qu'il écrit n'enclenchera pas la méthode de *callback*... Je vais quand même vous présenter cette solution, mais pour faire ce genre de surveillance, on préférera utiliser un `TextWatcher`. C'est comme un listener, mais ça n'en porte pas le nom !

Ma solution

```
1  import android.app.Activity;
2  import android.os.Bundle;
3  import android.view.KeyEvent;
4  import android.view.MotionEvent;
5  import android.view.View;
6  import android.view.View.OnClickListener;
7  import android.view.View.OnKeyListener;
8  import android.widget.Button;
9  import android.widget.CheckBox;
10 import android.widget.EditText;
11 import android.widget.RadioGroup;
12 import android.widget.TextView;
13 import android.widget.Toast;
14
15 public class IMCActivity extends Activity {
16   // La chaîne de caractères par défaut
17   private final String defaut = "Vous devez cliquer sur le
         bouton « Calculer l'IMC » pour obtenir un résultat.";
18   // La chaîne de caractères de la megafonction
19   private final String megaString = "Vous faites un poids
         parfait ! Wahou ! Trop fort ! On dirait Brad Pitt (si vous
          êtes un homme)/Angelina Jolie (si vous êtes une femme)/
         Willy (si vous êtes un orque) !";
```

```
20
21    Button envoyer = null;
22    Button raz = null;
23
24    EditText poids = null;
25    EditText taille = null;
26
27    RadioGroup group = null;
28
29    TextView result = null;
30
31    CheckBox mega = null;
32
33    @Override
34    public void onCreate(Bundle savedInstanceState) {
35      super.onCreate(savedInstanceState);
36      setContentView(R.layout.activity_main);
37
38      // On récupère toutes les vues dont on a besoin
39      envoyer = (Button)findViewById(R.id.calcul);
40
41      raz = (Button)findViewById(R.id.raz);
42
43      taille = (EditText)findViewById(R.id.taille);
44      poids = (EditText)findViewById(R.id.poids);
45
46      mega = (CheckBox)findViewById(R.id.mega);
47
48      group = (RadioGroup)findViewById(R.id.group);
49
50      result = (TextView)findViewById(R.id.result);
51
52      // On attribue un listener adapté aux vues qui en ont
           besoin
53      envoyer.setOnClickListener(envoyerListener);
54      raz.setOnClickListener(razListener);
55      taille.addTextChangedListener(textWatcher);
56      poids.addTextChangedListener(textWatcher);
57
58      // Solution avec des onKey
59      //taille.setOnKeyListener(modificationListener);
60      //poids.setOnKeyListener(modificationListener);
61      mega.setOnClickListener(checkedListener);
62    }
63
64    /*
65    // Se lance à chaque fois qu'on appuie sur une touche en é
         tant sur un EditText
66    private OnKeyListener modificationListener = new
         OnKeyListener() {
```

```
67        @Override
68        public boolean onKey(View v, int keyCode, KeyEvent event) {
69          // On remet le texte à sa valeur par défaut pour ne pas
              avoir de résultat incohérent
70          result.setText(defaut);
71          return false;
72        }
73      };*/
74
75      private TextWatcher textWatcher = new TextWatcher() {
76
77        @Override
78        public void onTextChanged(CharSequence s, int start, int
            before, int count) {
79          result.setText(defaut);
80        }
81
82        @Override
83        public void beforeTextChanged(CharSequence s, int start,
            int count,
84          int after) {
85
86        }
87
88        @Override
89        public void afterTextChanged(Editable s) {
90
91        }
92      };
93
94      // Uniquement pour le bouton "envoyer"
95      private OnClickListener envoyerListener = new OnClickListener
          () {
96        @Override
97        public void onClick(View v) {
98          if(!mega.isChecked()) {
99            // Si la megafonction n'est pas activée
100           // On récupère la taille
101           String t = taille.getText().toString();
102           // On récupère le poids
103           String p = poids.getText().toString();
104
105           float tValue = Float.valueOf(t);
106
107           // Puis on vérifie que la taille est cohérente
108           if(tValue == 0)
109             Toast.makeText(IMCActivity.this, "Hého, tu es un
                  Minipouce ou quoi ?", Toast.LENGTH_SHORT).show();
110           else {
111             float pValue = Float.valueOf(p);
```

```
112             // Si l'utilisateur a indiqué que la taille était en
                    centimètres
113             // On vérifie que la Checkbox sélectionnée est la
                    deuxième à l'aide de son identifiant
114             if(group.getCheckedRadioButtonId() == R.id.radio2)
115               tValue = tValue / 100;
116
117             tValue = (float)Math.pow(tValue, 2);
118             float imc = pValue / tValue;
119             result.setText("Votre IMC est " + String.valueOf(imc)
                    );
120           }
121         } else
122           result.setText(megaString);
123       }
124     };
125
126     // Listener du bouton de remise à zéro
127     private OnClickListener razListener = new OnClickListener() {
128       @Override
129       public void onClick(View v) {
130         poids.getText().clear();
131         taille.getText().clear();
132         result.setText(defaut);
133       }
134     };
135
136     // Listener du bouton de la megafonction.
137     private OnClickListener checkedListener = new OnClickListener
          () {
138       @Override
139       public void onClick(View v) {
140         // On remet le texte par défaut si c'était le texte de la
                megafonction qui était écrit
141         if(!((CheckBox)v).isChecked() && result.getText().equals(
              megaString))
142           result.setText(defaut);
143       }
144     };
145   }
```

Vous pouvez copier ce code grâce au code web suivant :

▷ Copier ce code
Code web : 723331

Pourquoi on retourne `false` dans le `onKeyListener` ? Il se serait passer quoi si j'avais retourné `true` ?

Curieux va ! En fait l'évènement `onKey` sera lancé avant que l'écriture soit prise en compte par le système. Ainsi, si vous renvoyez `true`, Android considérera que l'évènement a été géré, et que vous avez vous-même écrit la lettre qui a été pressée. Si vous renvoyez `false`, alors le système comprendra que vous n'avez pas écrit la lettre et il le fera de lui-même. Alors vous auriez très bien pu renvoyer `true`, mais il faudrait écrire nous-même la lettre et c'est du travail en plus pour rien !

Vous avez vu ce qu'on a fait ? Sans toucher à l'interface graphique, on a pu effectuer toutes les modifications nécessaires au bon fonctionnement de notre application. C'est l'intérêt de définir l'interface dans un fichier XML et le côté interactif en Java : vous pouvez modifier l'un sans toucher l'autre !

En résumé

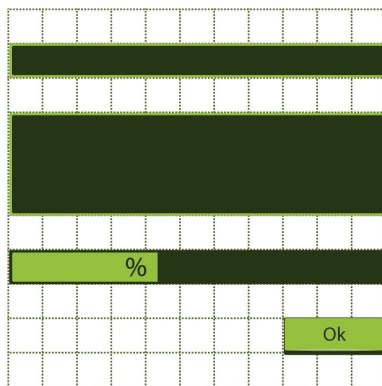
- Il existe un grand nombre de widgets différents. Parmi les plus utilisés, nous avons :
  - `TextView` destiné à afficher du texte sur l'écran.
  - `EditText` qui hérite des propriétés de `TextView` et qui permet à l'utilisateur d'écrire du texte.
  - `Button` qui hérite des propriétés de `TextView` et qui permet à l'utilisateur de cliquer sur du texte.
  - `CheckBox` qui hérite des propriétés de `Button` et qui permet à l'utilisateur de cocher une case.
  - `RadioButton` qui hérite des propriétés de `Button` et qui permet à l'utilisateur de choisir parmi plusieurs choix. De plus, `RadioGroup` est un layout spécifique aux `RadioButton`.
- N'oubliez pas que la documentation est l'unique endroit où vous pourrez trouver toutes les possibilités offertes pour chacun des widgets disponibles.
- Pour écouter les différents évènements qui pourraient se produire sur vos vues, on utilise des **listeners** qui enclenchent des méthodes de *callback* que vous pouvez redéfinir pour gérer leur implémentation.
- Android permet de lier des listeners à des vues de trois manières différentes :
  - Par héritage en implémentant l'interface au niveau de la classe, auquel cas il faudra réécrire les méthodes de *callback* directement dans votre classe.
  - Par classe anonyme en donnant directement une implémentation unique à la vue.
  - Par un attribut, si vous voulez réutiliser votre listener sur plusieurs vues.

Chapitre 7

Organiser son interface avec des layouts

Difficulté :

Pour l'instant, la racine de tous nos layouts a toujours été la même, ce qui fait que toutes nos applications avaient exactement le même squelette ! Mais il vous suffit de regarder n'importe quelle application Android pour réaliser que toutes les vues ne sont pas forcément organisées comme cela et qu'il existe une très grande variété d'architectures différentes. C'est pourquoi nous allons maintenant étudier les différents layouts, afin d'apprendre à placer nos vues comme nous le désirons. Nous pourrons ainsi concevoir une application plus attractive, plus esthétique et plus ergonomique !



LinearLayout : placer les éléments sur une ligne

Comme son nom l'indique, ce layout se charge de mettre les vues sur une même ligne, selon une certaine orientation. L'attribut pour préciser cette orientation est `android:orientation`. On peut lui donner deux valeurs :

- `vertical` pour que les composants soient placés de haut en bas (en colonne) ;
- `horizontal` pour que les composants soient placés de gauche à droite (en ligne).

On va faire quelques expériences pour s'amuser !

Premier exemple

```
1  <LinearLayout xmlns:android="http://schemas.android.com/apk/res
       /android"
2    android:orientation="vertical"
3    android:layout_width="fill_parent"
4    android:layout_height="fill_parent" >
5    <Button
6      android:id="@+id/premier"
7      android:layout_width="fill_parent"
8      android:layout_height="wrap_content"
9      android:text="Premier bouton" />
10
11   <Button
12     android:id="@+id/second"
13     android:layout_width="fill_parent"
14     android:layout_height="wrap_content"
15     android:text="Second bouton" />
16 </LinearLayout>
```

Le rendu de ce code se trouve à la figure 7.1.

- Le `LinearLayout` est vertical et prend toute la place de son parent (vous savez, l'invisible qui prend toute la place dans l'écran).
- Le premier bouton prend toute la place dans le parent en largeur et uniquement la taille nécessaire en hauteur (la taille du texte, donc !).
- Le second bouton fait de même.

Deuxième exemple

```
1  <LinearLayout xmlns:android="http://schemas.android.com/apk/res
       /android"
2    android:orientation="vertical"
3    android:layout_width="fill_parent"
4    android:layout_height="fill_parent" >
5
6    <Button
7      android:id="@+id/premier"
```

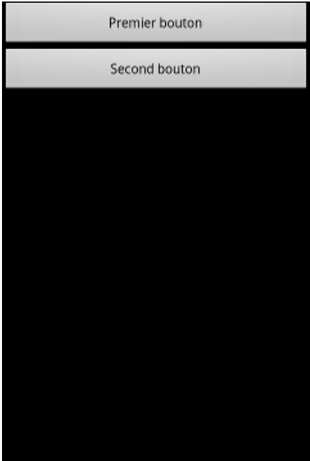


FIGURE 7.1 – Les deux boutons prennent toute la largeur

```
8        android:layout_width="wrap_content"
9        android:layout_height="fill_parent"
10       android:text="Premier bouton" />
11
12     <Button
13       android:id="@+id/second"
14       android:layout_width="wrap_content"
15       android:layout_height="fill_parent"
16       android:text="Second bouton" />
17   </LinearLayout>
```

Le rendu de ce code se trouve à la figure 7.2.

- Le `LinearLayout` est vertical et prend toute la place de son parent.
- Le premier bouton prend toute la place de son parent en hauteur et uniquement la taille nécessaire en largeur.
- Comme le premier bouton prend toute la place, alors le pauvre second bouton se fait écraser. C'est pour cela qu'on ne le voit pas.

Troisième exemple

```
1   <LinearLayout xmlns:android="http://schemas.android.com/apk/res
      /android"
2     android:orientation="vertical"
3     android:layout_width="wrap_content"
4     android:layout_height="wrap_content" >
5     <Button
6       android:id="@+id/premier"
7       android:layout_width="wrap_content"
8       android:layout_height="fill_parent"
```

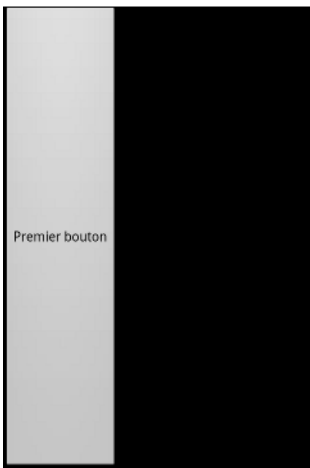


Figure 7.2 – Le premier bouton fait toute la hauteur, on ne voit donc pas le deuxième bouton

```
9       android:text="Premier bouton" />
10    <Button
11      android:id="@+id/second"
12      android:layout_width="wrap_content"
13      android:layout_height="fill_parent"
14      android:text="Second bouton" />
15  </LinearLayout>
```

Le rendu de ce code se trouve à la figure 7.3.

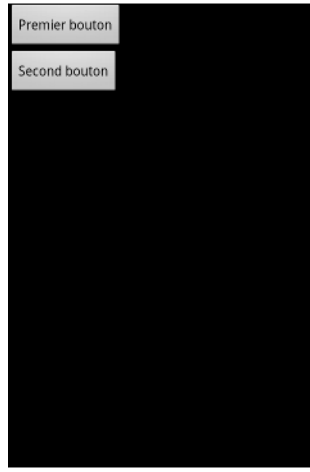


Figure 7.3 – Les deux boutons prennent uniquement la place nécessaire en hauteur et en largeur

- Le LinearLayout est vertical et prend toute la place en largeur mais uniquement la taille nécessaire en hauteur : dans ce cas précis, la taille nécessaire sera calculée en fonction de la taille des enfants.
- Le premier bouton prend toute la place possible dans le parent. Comme le parent prend le moins de place possible, il doit faire de même.
- Le second bouton fait de même.

Quatrième exemple

```
1  <LinearLayout xmlns:android="http://schemas.android.com/apk/res
       /android"
2    android:orientation="horizontal"
3    android:layout_width="fill_parent"
4    android:layout_height="fill_parent" >
5    <Button
6      android:id="@+id/premier"
7      android:layout_width="wrap_content"
8      android:layout_height="wrap_content"
9      android:text="Premier bouton" />
10   <Button
11     android:id="@+id/second"
12     android:layout_width="wrap_content"
13     android:layout_height="fill_parent"
14     android:text="Second bouton" />
15 </LinearLayout>
```

Le rendu de ce code se trouve à la figure 7.4.

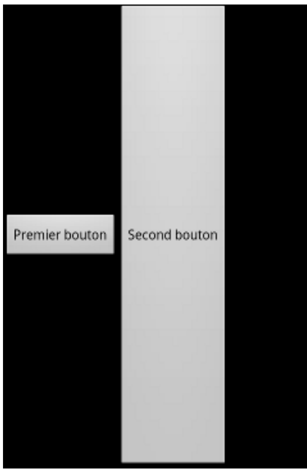


FIGURE 7.4 – Le premier bouton prend uniquement la place nécessaire et le deuxième toute la hauteur

- Le LinearLayout est horizontal et prend toute la place de son parent.

- Le premier bouton prend uniquement la place nécessaire.
- Le second bouton prend uniquement la place nécessaire en longueur et s'étend jusqu'aux bords du parent en hauteur.

Vous remarquerez que l'espace est toujours divisé entre les deux boutons, soit de manière égale, soit un bouton écrase complètement l'autre. Et si on voulait que le bouton de droite prenne deux fois plus de place que celui de gauche par exemple ?

Pour cela, il faut attribuer un poids au composant. Ce poids peut être défini grâce à l'attribut `android:layout_weight`. Pour faire en sorte que le bouton de droite prenne deux fois plus de place, on peut lui mettre `android:layout_weight="1"` et mettre au bouton de gauche `android:layout_weight="2"`. C'est alors le composant qui a la plus faible pondération qui a la priorité.

Et si, dans l'exemple précédent où un bouton en écrasait un autre, les deux boutons avaient eu un poids identique, par exemple `android:layout_weight="1"` pour les deux, ils auraient eu la même priorité et auraient pris la même place. Par défaut, ce poids est à 0.

Une astuce que j'utilise souvent consiste à faire en sorte que la somme des poids dans un même layout fasse 100. C'est une manière plus évidente pour répartir les poids.

Dernier attribut particulier pour les widgets de ce layout, `android:layout_gravity`, qu'il ne faut pas confondre avec `android:gravity`. `android:layout_gravity` vous permet de déterminer comment se placera la vue dans le parent, alors que `android:gravity` vous permet de déterminer comment se placera le contenu de la vue à l'intérieur même de la vue (par exemple, comment se placera le texte dans un `TextView` ? Au centre, en haut, à gauche ?).

Vous prendrez bien un petit exemple pour illustrer ces trois concepts ?

```
1  <LinearLayout xmlns:android="http://schemas.android.com/apk/res
       /android"
2    android:orientation="horizontal"
3    android:layout_width="fill_parent"
4    android:layout_height="fill_parent" >
5    <Button
6      android:id="@+id/bouton1"
7      android:layout_width="fill_parent"
8      android:layout_height="wrap_content"
9      android:layout_gravity="bottom"
10     android:layout_weight="40"
11     android:text="Bouton 1" />
12   <Button
13     android:id="@+id/bouton2"
14     android:layout_width="fill_parent"
15     android:layout_height="wrap_content"
16     android:layout_gravity="center"
17     android:layout_weight="20"
```

```
18        android:gravity="bottom|right"
19        android:text="Bouton 2" />
20      <Button
21        android:id="@+id/bouton3"
22        android:layout_width="fill_parent"
23        android:layout_height="wrap_content"
24        android:layout_gravity="top"
25        android:layout_weight="40"
26        android:text="Bouton 3" />
27    </LinearLayout>
```

Le rendu de ce code se trouve à la figure 7.5.

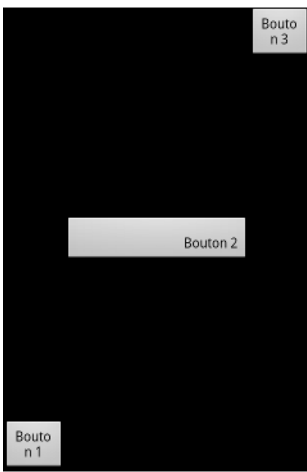


Figure 7.5 – Trois boutons placés différemment

Le bouton 2 a un poids deux fois inférieur aux boutons 1 et 3, alors il prend deux fois plus de place qu'eux. De plus, chaque bouton possède un attribut `android:layout_gravity` afin de que l'on détermine sa position dans le layout. Le deuxième bouton présente aussi l'attribut `android:gravity`, qui est un attribut de `TextView` et non `layout`, de façon à mettre le texte en bas (`bottom`) à droite (`right`).

Calcul de l'IMC - Partie 3.1

Énoncé

Récupérez le code de votre application de calcul de l'IMC et modifiez le layout pour obtenir quelque chose ressemblant à la figure 7.6.

Les `EditText` prennent le plus de place possible, mais comme ils ont un poids plus fort que les `TextView`, ils n'ont pas la priorité.

```
1  <?xml version="1.0" encoding="utf-8"?>
```

FIGURE 7.6 – Essayez d'obtenir la même interface

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical">
  <LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
  >
    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Poids : "
      android:textStyle="bold"
      android:textColor="#FF0000"
      android:gravity="center"
    />
    <EditText
      android:id="@+id/poids"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:hint="Poids"
      android:inputType="numberDecimal"
      android:layout_weight="1"
    />
  </LinearLayout>
  <LinearLayout
    android:layout_width="fill_parent"
```

```
30        android:layout_height="wrap_content"
31        android:orientation="horizontal"
32      >
33        <TextView
34          android:layout_width="wrap_content"
35          android:layout_height="wrap_content"
36          android:text="Taille : "
37          android:textStyle="bold"
38          android:textColor="#FF0000"
39          android:gravity="center"
40        />
41        <EditText
42          android:id="@+id/taille"
43          android:layout_width="fill_parent"
44          android:layout_height="wrap_content"
45          android:hint="Taille"
46          android:inputType="numberDecimal"
47          android:layout_weight="1"
48        />
49      </LinearLayout>
50      <RadioGroup
51        android:id="@+id/group"
52        android:layout_width="wrap_content"
53        android:layout_height="wrap_content"
54        android:checkedButton="@+id/radio2"
55        android:orientation="horizontal"
56      >
57        <RadioButton
58          android:id="@+id/radio1"
59          android:layout_width="wrap_content"
60          android:layout_height="wrap_content"
61          android:text="Mètre"
62        />
63        <RadioButton
64          android:id="@+id/radio2"
65          android:layout_width="wrap_content"
66          android:layout_height="wrap_content"
67          android:text="Centimètre"
68        />
69      </RadioGroup>
70      <CheckBox
71        android:id="@+id/mega"
72        android:layout_width="wrap_content"
73        android:layout_height="wrap_content"
74        android:text="Mega fonction !"
75      />
76      <LinearLayout
77        android:layout_width="fill_parent"
78        android:layout_height="wrap_content"
79        android:orientation="horizontal"
```

```
80        >
81          <Button
82            android:id="@+id/calcul"
83            android:layout_width="wrap_content"
84            android:layout_height="wrap_content"
85            android:text="Calculer l'IMC"
86            android:layout_weight="1"
87            android:layout_marginLeft="25dip"
88            android:layout_marginRight="25dip"
89          />
90          <Button
91            android:id="@+id/raz"
92            android:layout_width="wrap_content"
93            android:layout_height="wrap_content"
94            android:text="RAZ"
95            android:layout_weight="1"
96            android:layout_marginLeft="25dip"
97            android:layout_marginRight="25dip"
98          />
99        </LinearLayout>
100       <TextView
101         android:layout_width="wrap_content"
102         android:layout_height="wrap_content"
103         android:text="Résultat:"
104       />
105       <TextView
106         android:id="@+id/result"
107         android:layout_width="fill_parent"
108         android:layout_height="fill_parent"
109         android:text="Vous devez cliquer sur le bouton « Calculer l
              'IMC » pour obtenir un résultat."
110       />
111     </LinearLayout>
```

Vous pouvez copier ce code grâce au code web suivant :

▷ Copier ce code
Code web : 989289

De manière générale, on évite d'empiler les LinearLayout (avoir un LinearLayout dans un LinearLayout, dans un LinearLayout, etc.), c'est mauvais pour les performances d'une application.

RelativeLayout : placer les éléments les uns en fonction des autres

De manière totalement différente, ce layout propose plutôt de placer les composants les uns par rapport aux autres. Il est même possible de les placer par rapport au `RelativeLayout` parent.

Si on veut par exemple placer une vue au centre d'un `RelativeLayout`, on peut passer à cette vue l'attribut `android:layout_centerInParent="true"`. Il est aussi possible d'utiliser `android:layout_centerHorizontal="true"` pour centrer, mais uniquement sur l'axe horizontal, de même avec `android:layout_centerVertical="true"` pour centrer sur l'axe vertical.

Premier exemple

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Centré dans le parent"
    android:layout_centerInParent="true" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Centré verticalement"
    android:layout_centerVertical="true" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Centré horizontalement"
    android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Le rendu de ce code se trouve à la figure 7.7.

On observe ici une différence majeure avec le `LinearLayout` : il est possible d'empiler les vues. Ainsi, le `TextView` centré verticalement s'entremêle avec celui centré verticalement et horizontalement.

Il existe d'autres contrôles pour situer une vue par rapport à un `RelativeLayout`. On peut utiliser :

- `android:layout_alignParentBottom="true"` pour aligner le plancher d'une vue au plancher du `RelativeLayout` ;

FIGURE 7.7 – Deux vues sont empilées

- `android:layout_alignParentTop="true"` pour coller le plafond d'une vue au plafond du `RelativeLayout` ;
- `android:layout_alignParentLeft="true"` pour coller le bord gauche d'une vue avec le bord gauche du `RelativeLayout` ;
- `android:layout_alignParentRight="true"` pour coller le bord droit d'une vue avec le bord droit du `RelativeLayout`.

Deuxième exemple

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/
     res/android"
2    android:layout_width="fill_parent"
3    android:layout_height="fill_parent" >
4    <TextView
5      android:layout_width="wrap_content"
6      android:layout_height="wrap_content"
7      android:text="En haut !"
8      android:layout_alignParentTop="true" />
9    <TextView
10     android:layout_width="wrap_content"
11     android:layout_height="wrap_content"
12     android:text="En bas !"
13     android:layout_alignParentBottom="true" />
14   <TextView
15     android:layout_width="wrap_content"
16     android:layout_height="wrap_content"
17     android:text="A gauche !"
18     android:layout_alignParentLeft="true" />
19   <TextView
```

```
20        android:layout_width="wrap_content"
21        android:layout_height="wrap_content"
22        android:text="A droite !"
23        android:layout_alignParentRight="true" />
24      <TextView
25        android:layout_width="wrap_content"
26        android:layout_height="wrap_content"
27        android:text="Ces soirées là !"
28        android:layout_centerInParent="true" />
29    </RelativeLayout>
```

Le rendu de ce code se trouve à la figure 7.8.

FIGURE 7.8 – En haut à gauche, deux `TextView` se superposent

On remarque tout de suite que les `TextView` censés se situer à gauche et en haut s'entremêlent, mais c'est logique puisque par défaut une vue se place en haut à gauche dans un `RelativeLayout`. Donc, quand on lui dit « Place-toi à gauche » ou « Place-toi en haut », c'est comme si on ne lui donnait pas d'instructions au final.

Enfin, il ne faut pas oublier que le principal intérêt de ce layout est de pouvoir placer les éléments les uns par rapport aux autres. Pour cela il existe deux catégories d'attributs :

- Ceux qui permettent de positionner deux bords opposés de deux vues différentes ensemble. On y trouve `android:layout_below` (pour aligner le plafond d'une vue sous le plancher d'une autre), `android:layout_above` (pour aligner le plancher d'une vue sur le plafond d'une autre), `android:layout_toRightOf` (pour aligner le bord gauche d'une vue au bord droit d'une autre) et `android:layout_toLeftOf` (pour aligner le bord droit d'une vue au bord gauche d'une autre).
- Ceux qui permettent de coller deux bords similaires ensemble. On trouve :
  - `android:layout_alignBottom` (pour aligner le plancher de la vue avec le plancher d'une autre).
  - `android:layout_alignTop` (pour aligner le plafond de la vue avec le plafond d'une

autre).
- `android:layout_alignLeft` (pour aligner le bord gauche d'une vue avec le bord gauche d'une autre).
- `android:layout_alignRight` (pour aligner le bord droit de la vue avec le bord droit d'une autre).

Troisième exemple

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
  <TextView
    android:id="@+id/premier"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="[I] En haut à gauche par défaut" />
  <TextView
    android:id="@+id/deuxieme"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="[II] En dessous de (I)"
    android:layout_below="@id/premier" />
  <TextView
    android:id="@+id/troisieme"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="[III] En dessous et à droite de (I)"
    android:layout_below="@id/premier"
    android:layout_toRightOf="@id/premier" />
  <TextView
    android:id="@+id/quatrieme"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="[IV] Au dessus de (V), bord gauche aligné avec le bord gauche de (II)"
    android:layout_above="@+id/cinquieme"
    android:layout_alignLeft ="@id/deuxieme" />
  <TextView
    android:id="@+id/cinquieme"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="[V] En bas à gauche"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true" />
</RelativeLayout>
```

Le rendu de ce code se trouve à la figure 7.9.

Je vous demande maintenant de regarder l'avant dernier `TextView`, en particulier son

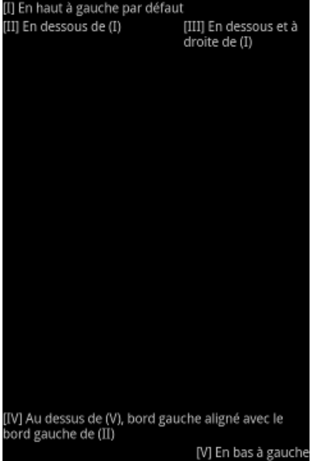


FIGURE 7.9 – Les TextView sont bien placés

attribut `android:layout_above`. On ne fait pas référence au dernier `TextView` comme aux autres, il faut préciser un +! Eh oui, rappelez-vous, je vous avais dit il y a quelques chapitres déjà que, si nous voulions faire référence à une vue qui n'était définie que plus tard dans le fichier XML, alors il fallait ajouter un + dans l'identifiant, sinon Android pensera qu'il s'agit d'une faute et non d'un identifiant qui sera déclaré après.

Calcul de l'IMC - Partie 3.2

Même chose pour un layout différent ! Moi, je vise le même résultat que précédemment.

Ma solution

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/
       res/android"
3    android:layout_width="fill_parent"
4    android:layout_height="fill_parent">
5    <TextView
6      android:id="@+id/textPoids"
7      android:layout_width="wrap_content"
8      android:layout_height="wrap_content"
9      android:text="Poids : "
10     android:textStyle="bold"
11     android:textColor="#FF0000"
12   />
13   <EditText
14     android:id="@+id/poids"
15     android:layout_width="wrap_content"
16     android:layout_height="wrap_content"
```

```
17      android:hint="Poids"
18      android:inputType="numberDecimal"
19      android:layout_toRightOf="@id/textPoids"
20      android:layout_alignParentRight="true"
21    />
22    <TextView
23      android:id="@+id/textTaille"
24      android:layout_width="wrap_content"
25      android:layout_height="wrap_content"
26      android:text="Taille : "
27      android:textStyle="bold"
28      android:textColor="#FF0000"
29      android:gravity="left"
30      android:layout_below="@id/poids"
31    />
32    <EditText
33      android:id="@+id/taille"
34      android:layout_width="wrap_content"
35      android:layout_height="wrap_content"
36      android:hint="Taille"
37      android:inputType="numberDecimal"
38      android:layout_below="@id/poids"
39      android:layout_toRightOf="@id/textTaille"
40      android:layout_alignParentRight="true"
41    />
42    <RadioGroup
43      android:id="@+id/group"
44      android:layout_width="wrap_content"
45      android:layout_height="wrap_content"
46      android:checkedButton="@+id/radio2"
47      android:orientation="horizontal"
48      android:layout_below="@id/taille"
49    >
50      <RadioButton
51        android:id="@+id/radio1"
52        android:layout_width="wrap_content"
53        android:layout_height="wrap_content"
54        android:text="Mètre"
55      />
56      <RadioButton
57        android:id="@+id/radio2"
58        android:layout_width="wrap_content"
59        android:layout_height="wrap_content"
60        android:text="Centimètre"
61      />
62    </RadioGroup>
63    <CheckBox
64      android:id="@+id/mega"
65      android:layout_width="wrap_content"
66      android:layout_height="wrap_content"
```

```
67        android:text="Mega fonction !"
68        android:layout_below="@id/group"
69      />
70      <Button
71        android:id="@+id/calcul"
72        android:layout_width="wrap_content"
73        android:layout_height="wrap_content"
74        android:text="Calculer l'IMC"
75        android:layout_below="@id/mega"
76        android:layout_marginLeft="25dip"
77      />
78      <Button
79        android:id="@+id/raz"
80        android:layout_width="wrap_content"
81        android:layout_height="wrap_content"
82        android:text="RAZ"
83        android:layout_below="@id/mega"
84        android:layout_alignRight="@id/taille"
85        android:layout_marginRight="25dip"
86      />
87      <TextView
88        android:id="@+id/resultPre"
89        android:layout_width="wrap_content"
90        android:layout_height="wrap_content"
91        android:text="Résultat :"
92        android:layout_below="@id/calcul"
93      />
94      <TextView
95        android:id="@+id/result"
96        android:layout_width="fill_parent"
97        android:layout_height="fill_parent"
98        android:text="Vous devez cliquer sur le bouton « Calculer l
              'IMC » pour obtenir un résultat."
99        android:layout_below="@id/resultPre"
100     />
101   </RelativeLayout>
```

Vous pouvez copier ce code grâce au code web suivant :

▷ Copier ce code
Code web : 565746

Le problème de ce layout, c'est qu'une petite modification dans l'interface graphique peut provoquer de grosses modifications dans tout le fichier XML, il faut donc savoir par avance très précisément ce qu'on veut faire.

Il s'agit du layout le plus compliqué à maîtriser, et pourtant le plus puissant tout en étant l'un des moins gourmands en ressources. Je vous encourage fortement à vous entraîner à l'utiliser.

TableLayout : placer les éléments comme dans un tableau

Dernier layout de base, il permet d'organiser les éléments en tableau, comme en HTML, mais sans les bordures. Voici un exemple d'utilisation de ce layout :

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:stretchColumns="1">
  <TextView
    android:text="Les items précédés d'un V ouvrent un sous-
      menu"
  />
  <View
    android:layout_height="2dip"
    android:background="#FF909090"
  />
  <TableRow>
    <TextView
      android:text="N'ouvre pas un sous-menu"
      android:layout_column="1"
      android:padding="3dip"
    />
    <TextView
      android:text="Non !"
      android:gravity="right"
      android:padding="3dip"
    />
  </TableRow>
  <TableRow>
    <TextView
      android:text="V"
    />
    <TextView
      android:text="Ouvre un sous-menu"
      android:layout_column="1"
      android:padding="3dip"
    />
    <TextView
      android:text="Là si !"
      android:gravity="right"
      android:padding="3dip"
    />
  </TableRow>
  <View
    android:layout_height="2dip"
    android:background="#FF909090"
```

```
43      />
44      <TableRow>
45        <TextView
46          android:text="V"
47        />
48        <TextView
49          android:text="Ouvre un sous-menu"
50          android:padding="3dip"
51        />
52      </TableRow>
53      <View
54        android:layout_height="2dip"
55        android:background="#FF909090"
56      />
57      <TableRow>
58        <TextView
59          android:layout_column="1"
60          android:layout_span="2"
61          android:text="Cet item s'étend sur deux colonnes, cool
              hein ?"
62          android:padding="3dip"
63        />
64      </TableRow>
65    </TableLayout>
```

Ce qui donne la figure 7.10.

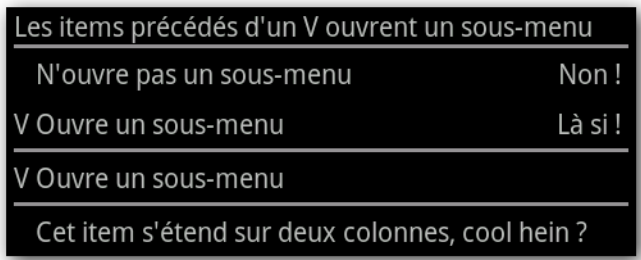


FIGURE 7.10 – Le contenu est organisé en tableau

On observe tout d'abord qu'il est possible de mettre des vues directement dans le tableau, auquel cas elles prendront toute la place possible en longueur. En fait, elles s'étendront sur toutes les colonnes du tableau. Cependant, si on veut un contrôle plus complet ou avoir plusieurs éléments sur une même ligne, alors il faut passer par un objet `<TableRow>`.

```
1   <TextView
2     android:text="Les items précédés d'un V ouvrent un sous-menu"
          />
```

Cet élément s'étend sur toute la ligne puisqu'il ne se trouve pas dans un `<TableRow>`

```
<View
  android:layout_height="2dip"
  android:background="#FF909090" />
```

Moyen efficace pour dessiner un séparateur — n'essayez pas de le faire en dehors d'un `<TableLayout>` ou votre application plantera.

Une ligne est composée de cellules. Chaque cellule peut contenir une vue, ou être vide. La taille du tableau en colonnes est celle de la ligne qui contient le plus de cellules. Dans notre exemple, nous avons trois colonnes pour tout le tableau, puisque la ligne avec le plus cellules est celle qui contient « V » et se termine par « Là si ! ».

```
<TableRow>
  <TextView
    android:text="V"
  />
  <TextView
    android:text="Ouvre un sous-menu"
    android:layout_column="1"
    android:padding="3dip"
  />
  <TextView
    android:text="Là si !"
    android:gravity="right"
    android:padding="3dip"
  />
</TableRow>
```

Cette ligne a trois éléments, c'est la plus longue du tableau, ce dernier est donc constitué de trois colonnes.

On peut choisir dans quelle colonne se situe un item avec l'attribut `android:layout_column`. Attention, l'index des colonnes commence à 0. Dans notre exemple, le dernier item se place directement à la deuxième colonne grâce à `android:layout_column="1"`.

```
<TableRow>
  <TextView
    android:text="N'ouvre pas un sous-menu"
    android:layout_column="1"
    android:padding="3dip"
  />
  <TextView
    android:text="Non !"
    android:gravity="right"
    android:padding="3dip"
  />
</TableRow>
```

On veut laisser vide l'espace pour la première colonne, on place alors les deux `TextView` dans les colonnes 1 et 2.

La taille d'une cellule dépend de la cellule la plus large sur une même colonne. Dans notre exemple, la seconde colonne fait la largeur de la cellule qui contient le texte « N'ouvre pas un sous-menu », puisqu'il se trouve dans la deuxième colonne et qu'il n'y a pas d'autres éléments dans cette colonne qui soit plus grand.

Enfin, il est possible d'étendre un item sur plusieurs colonnes à l'aide de l'attribut `android:layout_span`. Dans notre exemple, le dernier item s'étend de la deuxième colonne à la troisième. Il est possible de faire de même sur les lignes avec l'attribut `android:layout_column`.

```
1 <TableRow>
2   <TextView
3     android:layout_column="1"
4     android:layout_span="2"
5     android:text="Cet item s'étend sur deux colonnes, cool hein
        ?"
6     android:padding="3dip"
7   />
8 </TableRow>
```

Ce `TextView` débute à la deuxième colonne et s'étend sur deux colonnes, donc jusqu'à la troisième.

Sur le nœud `TableLayout`, on peut jouer avec trois attributs (attention, les rangs débutent à 0) :

- `android:stretchColumns` pour que la longueur de tous les éléments de cette colonne passe en `fill_parent`, donc pour prendre le plus de place possible. Il faut préciser le rang de la colonne à cibler, ou plusieurs rangs séparés par des virgules.
- `android:shrinkColumns` pour que la longueur de tous les éléments de cette colonne passe en `wrap_content`, donc pour prendre le moins de place possible. Il faut préciser le rang de la colonne à cibler, ou plusieurs rangs séparés par des virgules.
- `android:collapseColumns` pour faire purement et simplement disparaître des colonnes du tableau. Il faut préciser le rang de la colonne à cibler, ou plusieurs rangs séparés par des virgules.

Calcul de l'IMC - Partie 3.3

Énoncé

Réitérons l'expérience, essayez encore une fois d'obtenir le même rendu, mais cette fois avec un `TableLayout`. L'exercice est intéressant puisqu'on n'est pas vraiment en présence d'un tableau, il va donc falloir réfléchir beaucoup et exploiter au maximum vos connaissances pour obtenir un rendu acceptable.

Ma solution

```
1 <?xml version="1.0" encoding="utf-8"?>
```

```
2  <TableLayout xmlns:android="http://schemas.android.com/apk/res/
     android"
3    android:layout_width="fill_parent"
4    android:layout_height="fill_parent"
5    android:stretchColumns="1">
6    <TableRow>
7      <TextView
8        android:text="Poids : "
9        android:textStyle="bold"
10       android:textColor="#FF0000"
11       android:gravity="center"
12     />
13     <EditText
14       android:id="@+id/poids"
15       android:hint="Poids"
16       android:inputType="numberDecimal"
17       android:layout_span="2"
18     />
19   </TableRow>
20   <TableRow>
21     <TextView
22       android:layout_width="fill_parent"
23       android:layout_height="wrap_content"
24       android:text="Taille : "
25       android:textStyle="bold"
26       android:textColor="#FF0000"
27       android:gravity="center"
28     />
29     <EditText
30       android:id="@+id/taille"
31       android:layout_width="fill_parent"
32       android:layout_height="wrap_content"
33       android:hint="Taille"
34       android:inputType="numberDecimal"
35       android:layout_span="2"
36     />
37   </TableRow>
38   <RadioGroup
39     android:id="@+id/group"
40     android:layout_width="wrap_content"
41     android:layout_height="wrap_content"
42     android:checkedButton="@+id/radio2"
43     android:orientation="horizontal">
44     <RadioButton
45       android:id="@+id/radio1"
46       android:layout_width="wrap_content"
47       android:layout_height="wrap_content"
48       android:text="Mètre"
49     />
50     <RadioButton
```

```
51          android:id="@+id/radio2"
52          android:layout_width="wrap_content"
53          android:layout_height="wrap_content"
54          android:text="Centimètre"
55        />
56      </RadioGroup>
57      <CheckBox
58        android:id="@+id/mega"
59        android:layout_width="wrap_content"
60        android:layout_height="wrap_content"
61        android:text="Mega fonction !"
62      />
63      <TableRow>
64        <Button
65          android:id="@+id/calcul"
66          android:layout_width="wrap_content"
67          android:layout_height="wrap_content"
68          android:text="Calculer l'IMC"
69        />
70        <Button
71          android:id="@+id/raz"
72          android:layout_width="wrap_content"
73          android:layout_height="wrap_content"
74          android:text="RAZ"
75          android:layout_column="2"
76        />
77      </TableRow>
78      <TextView
79        android:layout_width="wrap_content"
80        android:layout_height="wrap_content"
81        android:text="Résultat:"
82      />
83      <TextView
84        android:id="@+id/result"
85        android:layout_width="fill_parent"
86        android:layout_height="fill_parent"
87        android:text="Vous devez cliquer sur le bouton « Calculer l
            'IMC » pour obtenir un résultat."
88      />
89    </TableLayout>
```

Vous pouvez copier ce code grâce au code web suivant :

▷ Copier ce code
Code web : 419144

FrameLayout : un layout un peu spécial

Ce layout est plutôt utilisé pour afficher une unique vue. Il peut sembler inutile comme ça, mais ne l'est pas du tout ! Il n'est destiné à afficher qu'un élément, mais il est

possible d'en mettre plusieurs dedans puisqu'il s'agit d'un `ViewGroup`. Si par exemple vous souhaitez faire un album photo, il vous suffit de mettre plusieurs éléments dans le `FrameLayout` et de ne laisser qu'une seule photo visible, en laissant les autres invisibles grâce à l'attribut `android:visibility` (cet attribut est disponible pour toutes les vues). Pareil pour un lecteur de PDF, il suffit d'empiler toutes les pages dans le `FrameLayout` et de n'afficher que la page actuelle, celle du dessus de la pile, à l'utilisateur. Cet attribut peut prendre trois valeurs :

- `visible` (`View.VISIBLE`), la valeur par défaut.
- `invisible` (`View.INVISIBLE`) n'affiche rien, mais est pris en compte pour l'affichage du layout niveau spatial (on lui réserve de la place).
- `gone` (`View.GONE`) n'affiche rien et ne prend pas de place, un peu comme s'il n'était pas là.

L'équivalent Java de cet attribut est `public void setVisibility (int)` avec comme paramètre une des valeurs entre parenthèses dans la liste ci-dessus. Quand il y a plusieurs éléments dans un `FrameLayout`, celui-ci les empile les uns au-dessus des autres, le premier élément du XML se trouvant en dernière position et le dernier ajouté tout au-dessus.

ScrollView : faire défiler le contenu d'une vue

Ne vous laissez pas bernez par son nom, cette vue est bel et bien un layout. Elle est par ailleurs un peu particulière puisqu'elle fait juste en sorte d'ajouter une barre de défilement verticale à un autre layout. En effet, si le contenu de votre layout dépasse la taille de l'écran, une partie du contenu sera invisible à l'utilisateur. De façon à rendre ce contenu visible, on peut préciser que la vue est englobée dans une `ScrollView`, et une barre de défilement s'ajoutera automatiquement.

Ce layout hérite de `FrameLayout`, par conséquent il vaut mieux envisager de ne mettre qu'une seule vue dedans. Il s'utilise en général avec `LinearLayout`, mais peut être utilisé avec tous les layouts… ou bien des widgets ! Par exemple :

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <ScrollView
3  xmlns:android="http://schemas.android.com/apk/res/android"
4  android:layout_width="fill_parent"
5  android:layout_height="wrap_content">
6    <LinearLayout>
7      <!-- contenu du layout -->
8    </LinearLayout>
9  </ScrollView>
```

Attention cependant, il ne faut pas mettre de widgets qui peuvent déjà défiler dans une `ScrollView`, sinon il y aura conflit entre les deux contrôleurs et le résultat sera médiocre. Nous n'avons pas encore vu de widgets de ce type, mais cela ne saurait tarder.

En résumé

- `LinearLayout` permet d'afficher plusieurs vues sur une même ligne de manière horizontale ou verticale. Il est possible d'attribuer un poids aux vues pour effectuer des placements précis.
- `RelativeLayout` permet d'afficher des vues les unes en fonction des autres.
- `TableLayout` permet d'organiser les éléments en tableau.
- `FrameLayout` permet d'afficher une vue à l'écran ou d'en superposer plusieurs les unes au-dessus des autres.
- `ScrollView` permet de rendre « scrollable » la vue qu'elle contient. Ne lui donnez qu'un fils et de ne fournissez pas de vues déjà « scrollable » sinon il y aura des conflits.

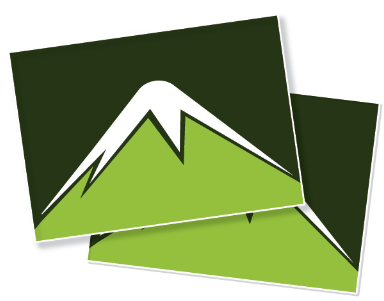
Chapitre 8

Les autres ressources

Difficulté :

Maintenant que vous avez parfaitement compris ce qu'étaient les ressources, pourquoi et comment les utiliser, je vous propose de voir… comment les créer. Il existe une grande variété ressources différences, c'est pourquoi on ne les verra pas toutes. Je vous présenterai ici uniquement les plus utiles et plus compliquées à utiliser.

Un dernier conseil avant d'entrer dans le vif du sujet : créez le plus de ressources possible, dès que vous le pouvez. Ainsi, vos applications seront plus flexibles, et le développement sera plus évident.



Aspect général des fichiers de ressources

Nous allons voir comment sont constitués les fichiers de ressources qui contiennent des *values* (je les appellerai « données » désormais). C'est encore une fois un fichier XML, mais qui revêt cette forme-ci :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  ...
</resources>
```

Afin d'avoir un petit aperçu de ce à quoi elles peuvent ressembler, on va d'abord observer les fichiers que crée Android à la création d'un nouveau projet. Double-cliquez sur le fichier `res/values/strings.xml` pour ouvrir une nouvelle fenêtre (voir figure 8.1).

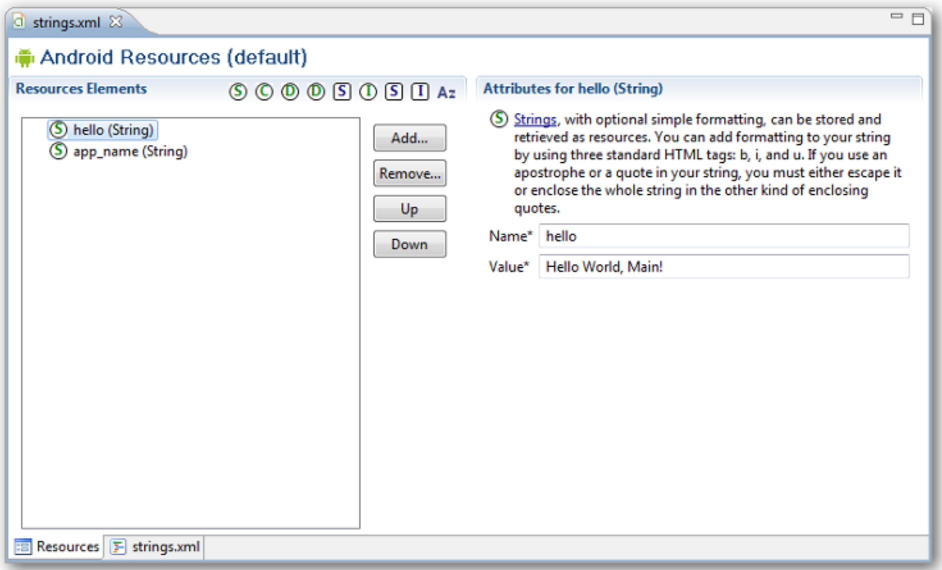


FIGURE 8.1 – Fenêtre d'édition des données

On retrouve à gauche toutes les ressources qui sont contenues dans ce fichier. Là il y en a deux, c'est plutôt facile de s'y retrouver, mais imaginez un gros projet avec une cinquantaine voire une centaine de données, vous risquez de vite vous y perdre. Si vous voulez éviter ce type de désagréments, vous pouvez envisager deux manières de vous organiser :

- Réunir les données d'un même type pour une activité dans un seul fichier. Par exemple `strings.xml` pour toutes les chaines de caractères. Le problème est qu'il vous faudra créer beaucoup de fichiers, ce qui peut être long.
- Ou alors mettre toutes les données d'une activité dans un fichier, ce qui demande

moins de travail, mais nécessite une meilleure organisation afin de pouvoir s'y retrouver.

N'oubliez pas qu'Android est capable de retrouver automatiquement des ressources parce qu'elles se situent dans un fichier précis à un emplacement précis. Ainsi, quelle que soit l'organisation pour laquelle vous optez, il faudra la répercuter à tous les répertoires `values`, tous différenciés par des quantificateurs, pour que les données se retrouvent dans des fichiers au nom identique mais dans des répertoires différents.

Si vous souhaitez opter pour la seconde organisation, alors le meilleur moyen de s'y retrouver est de savoir trier les différentes ressources à l'aide du menu qui se trouve en haut de la fenêtre. Il vous permet de filtrer la liste des données en fonction de leur type. Voici la signification de tous les boutons :

- Afficher uniquement les chaînes de caractères (`String`)
- Afficher uniquement les couleurs (`Color`)
- Afficher uniquement les dimensions (`Dimension`)
- Afficher uniquement les drawables (`Drawable`)
- Afficher uniquement les styles et thèmes (`Style`)
- Afficher uniquement les éléments qui appartiennent à un ensemble (à un tableau par exemple) (`Item`)
- Afficher uniquement les tableaux de chaînes de caractères (`String Array`)
- Afficher uniquement les tableaux d'entiers (`Int Array`)
- Ranger la liste dans l'ordre alphabétique du nom de la donnée. Un second clic range dans l'ordre alphabétique inverse

De plus, le menu du milieu (voir figure 8.2) vous permet de créer ou supprimer des données.

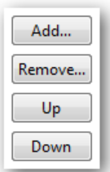


FIGURE 8.2 – Menu du milieu

- Le bouton `Add...` permet d'ajouter une nouvelle donnée.
- Le bouton `Remove...` permet de supprimer une donnée.
- Le bouton `Up` permet d'augmenter d'un cran la position de la donnée dans le tableau central.
- Le bouton `Down` permet de diminuer d'un cran la position de la donnée dans le tableau central.

Personnellement, je n'utilise cette fenêtre que pour avoir un aperçu rapide de mes données. Cependant, dès qu'il me faut effectuer des manipulations, je préfère utiliser l'éditeur XML. D'ailleurs je ne vous apprendrai ici qu'à travailler avec un fichier XML, de manière à ce que vous ne soyez pas totalement déboussolés si vous souhaitiez utiliser une autre extension que l'ADT. Vous pouvez naviguer entre les deux interfaces à l'aide des deux boutons en bas de la fenêtre, visibles à la figure 8.3.

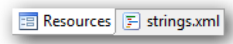


FIGURE 8.3 – Vous pouvez naviguer entre les deux interfaces

Référence à une ressource

Nous avons déjà vu que quand une ressource avait un identifiant, Android s'occupait d'ajouter au fichier `R.java` une référence à l'identifiant de cette ressource, de façon à ce que nous puissions la récupérer en l'inflatant. La syntaxe de la référence était :

```
1 R.type_de_ressource.nom_de_la_ressource
```

Ce que je ne vous ai pas dit, c'est qu'il était aussi possible d'y accéder en XML. Ce n'est pas tellement plus compliqué qu'en Java puisqu'il suffit de respecter la syntaxe suivante :

```
1 @type_de_ressource/nom_de_la_ressource
```

Par exemple pour une chaîne de caractères qui s'appellerait `salut`, on y ferait référence en Java à l'aide de `R.strings.salut` et en XML avec `@string/salut`.

Enfin, si la ressource à laquelle on essaie d'accéder est une ressource fournie par Android dans son SDK, il suffit de respecter la syntaxe suivante :

```
// Java
Android.R.type_de_ressource.nom_de_la_ressource

// XML
@android:type_de_ressource/nom_de_la_ressource
```

Les chaînes de caractères

Vous connaissez les chaînes de caractères, c'est le mot compliqué pour désigner un texte. La syntaxe est évidente à maîtriser, par exemple si nous voulions créer une chaîne de caractères de nom « nomDeLExemple » et de valeur `Texte de la chaîne qui s'appelle "nomDeLExemple"` :

```
<string name="nomDeLExemple">Texte de la chaîne qui s appelle
    nomDeLExemple</string>
```

Et ils ont disparu où, les guillemets et l'apostrophe ?

Commençons par l'évidence, s'il n'y a ni espace, ni apostrophe dans le nom, c'est parce qu'il s'agit du nom d'une variable comme nous l'avons vu précédemment, par conséquent il faut respecter les règles de nommage d'une variable standard.

Pour ce qui est du texte, il est interdit d'insérer des apostrophes ou des guillemets. Sinon, comment Android peut-il détecter que vous avez fini d'écrire une phrase ? Afin de contourner cette limitation, vous pouvez très bien échapper les caractères gênants, c'est-à-dire les faire précéder d'un antislash (\).

```
<string name="nomDeLExemple">Texte de la chaîne qui s\'appelle
    \"nomDeLExemple\"</string>
```

Vous pouvez aussi encadrer votre chaîne de guillemets afin de ne pas avoir à échapper les apostrophes ; en revanche vous aurez toujours à échapper les guillemets.

```
<string name="nomDeLExemple">"Texte de la chaîne qui s'appelle
    \"nomDeLExemple\""</string>
```

Application

Je vous propose de créer un bouton et de lui associer une chaîne de caractères qui contient des balises HTML (<b>, <u> et <i>) ainsi que des guillemets et des apos-

trophes. Si vous ne connaissez pas de balises HTML, vous allez créer la chaîne suivante : « `Vous connaissez l’histoire de <b>"Tom Sawyer"</b> ?` ». Les balises `<b></b>` vous permettent de mettre du texte en gras.

Instructions

- On peut convertir notre `String` en `Spanned`. `Spanned` est une classe particulière qui représente les chaînes de caractères qui contiennent des balises HTML et qui peut les interpréter pour les afficher comme le ferait un navigateur internet. Cette transformation se fait à l’aide de la méthode statique `Spanned Html.fromHtml (String source)`.
- On mettra ce `Spanned` comme texte sur le bouton avec la méthode `void setText (CharSequence text)`.

Les caractères spéciaux < et > doivent être écrits en code HTML. Au lieu d’écrire < vous devez marquer `«` et à la place de > il faut insérer `»`. Si vous utilisez l’interface graphique pour la création de `String`, il convertira automatiquement les caractères ! Mais il convertira aussi les guillemets en code HTML, ce qu’il ne devrait pas faire...

Ma correction

Le fichier `strings.xml` :

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3    <string name="hello">Hello World, TroimsActivity!</string>
4    <string name="histoire">Vous connaissez l\'histoire de <b>\"
       Tom Sawyer\"</b> ?</string>
5    <string name="app_name">Troims</string>
6  </resources>
```

Et le code Java associé :

```
1  import android.app.Activity;
2  import android.os.Bundle;
3  import android.text.Html;
4  import android.text.Spanned;
5  import android.widget.Button;
6
7  public class StringExampleActivity extends Activity {
8    Button button = null;
9    String hist = null;
10
11   @Override
12   public void onCreate(Bundle savedInstanceState) {
13     super.onCreate(savedInstanceState);
14
```

```
15        // On récupère notre ressource au format String
16        hist = getResources().getString(R.string.histoire);
17        // On le convertit en Spanned
18        Spanned marked_up = Html.fromHtml(hist);
19
20        button = new Button(this);
21        // Et on attribue le Spanned au bouton
22        button.setText(marked_up);
23
24        setContentView(button);
25    }
26 }
```

Formater des chaînes de caractères

Le problème avec nos chaînes de caractères en tant que ressources, c'est qu'elles sont statiques. Elles ne sont pas destinées à être modifiées et par conséquent elles ne peuvent pas s'adapter.

Imaginons une application qui salue quelqu'un, qui lui donne son âge, et qui s'adapte à la langue de l'utilisateur. Il faudrait qu'elle dise : « Bonjour Anaïs, vous avez 22 ans » en français et « Hello Anaïs, you are 22 » en anglais. Cette technique est par exemple utilisée dans le jeu *Civilization IV* pour traduire le texte en plusieurs langues. Pour indiquer dans une chaîne de caractères à quel endroit se situe la partie dynamique, on va utiliser un code. Dans l'exemple précédent, on pourrait avoir `Bonjour %1$s, vous avez %2$d ans` en français et `Hello %1$s, you are %2$d` en anglais. L'astuce est que la première partie du code correspond à une position dans une liste d'arguments (qu'il faudra fournir) et la seconde partie à un type de texte (`int`, `float`, `string`, `bool`, ...). En d'autres termes, un code se décompose en deux parties :

- `%n` avec « n » étant un entier naturel (nombre sans virgule et supérieur à 0) qui sert à indiquer le rang de l'argument à insérer (`%1` correspond au premier argument, `%2` au deuxième argument, etc.) ;
- `$x`, qui indique quel type d'information on veut ajouter (`$s` pour une chaîne de caractères et `$d` pour un entier — vous pourrez trouver la liste complète des possibilités sur la documentation).

▷ Voir les possibilités
Code web : 841732

On va maintenant voir comment insérer les arguments. Il existe au moins deux manières de faire.

On peut le faire en récupérant la ressource :

```
1 Resources res = getResources();
2 // Anaïs ira en %1 et 22 ira en %2
3 String chaine = res.getString(R.string.hello, "Anaïs", 22);
```

Ou alors sur n'importe quelle chaîne avec une fonction statique de `String` :

```
// On n'est pas obligé de préciser la position puisqu'on n'a qu
    'un argument !
String iLike = String.format("J'aime les $s", "pâtes");
```

Application

C'est simple, je vais vous demander d'arriver au résultat visible à la figure 8.4.

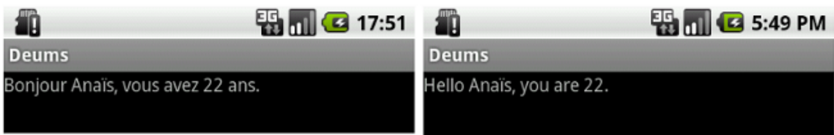


FIGURE 8.4 – L'exemple à reproduire

Ma solution

On aura besoin de deux fichiers `strings.xml` : un dans le répertoire `values` et un dans le répertoire `values-en` qui contiendra le texte en anglais :

`values/strings.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">Bonjour %1$s, vous avez %2$d ans.</
      string>
  <string name="app_name">Deums</string>
</resources>
```

`values-en/strings.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">Hello %1$s, you are %2$d.</string>
  <string name="app_name">Deums</string>
</resources>
```

De plus on va donner un identifiant à notre `TextView` pour récupérer la chaîne :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <TextView
    android:id="@+id/vue"
```

```
9       android:layout_width="fill_parent"
10      android:layout_height="wrap_content" />
11
12  </LinearLayout>
```

Et enfin, on va récupérer notre `TextView` et afficher le texte correct pour une femme s'appelant Anaïs et qui aurait 22 ans :

```
1   import android.app.Activity;
2   import android.content.res.Resources;
3   import android.os.Bundle;
4   import android.widget.TextView;
5
6   public class DeumsActivity extends Activity {
7     @Override
8     public void onCreate(Bundle savedInstanceState) {
9       super.onCreate(savedInstanceState);
10
11      setContentView(R.layout.main);
12
13      Resources res = getResources();
14      // Anaïs se mettra dans %1 et 22 ira dans %2, mais le reste
            changera en fonction de la langue du terminal !
15      String chaine = res.getString(R.string.hello, "Anaïs", 22);
16      TextView vue = (TextView)findViewById(R.id.vue);
17      vue.setText(chaine);
18    }
19  }
```

Et voilà, en fonction de la langue de l'émulateur, le texte sera différent !

Les drawables

La dénomination « drawable » rassemble tous les fichiers « dessinables » (oui, ce mot n'existe pas en français, mais « drawable » n'existe pas non plus en anglais après tout), c'est-à-dire les dessins ou les images. Je ne parlerai que des images puisque ce sont les drawables les plus utilisés et les plus indispensables.

Les images matricielles

Android supporte trois types d'images : les PNG, les GIF et les JPEG. Sachez que ces trois formats n'ont pas les mêmes usages :

- Les GIF sont peu recommandés. On les utilise sur internet pour les images de moindre qualité ou les petites animations. On va donc les éviter le plus souvent.
- Les JPEG sont surtout utilisés en photographie ou pour les images dont on veut conserver la haute qualité. Ce format ne gère pas la transparence, donc toutes vos images seront rectangulaires.

– Les PNG sont un bon compromis entre compression et qualité d'image. De plus, ils gèrent la transparence. Si le choix se présente, optez pour ce format-là.

Il n'y a rien de plus simple que d'ajouter une image dans les ressources, puisqu'il suffit de faire glisser le fichier à l'emplacement voulu dans Eclipse (ou mettre le fichier dans le répertoire voulu dans les sources de votre projet), comme à la figure 8.5, et le drawable sera créé automatiquement.

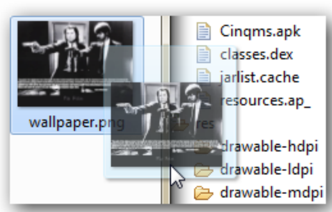


FIGURE 8.5 – On se contente de glisser-déposer l'image dans le répertoire voulu et Android fera le reste

Le nom du fichier déterminera l'identifiant du drawable, et il pourra contenir toutes les lettres minuscules, tous les chiffres et des underscores (_), mais attention, pas de majuscules. Puis, on pourra récupérer le drawable à l'aide de `R.drawable.nom_du_fichier_sans_l_extension`.

Les images extensibles

Utiliser une image permet d'agrémenter son application, mais, si on veut qu'elle soit de qualité pour tous les écrans, il faudrait une image pour chaque résolution, ce qui est long. La solution la plus pratique serait une image qui s'étire sans jamais perdre en qualité ! Dans les faits, c'est difficile à obtenir, mais certaines images sont assez simples pour qu'Android puisse déterminer comment étirer l'image en perdant le moins de qualité possible. Je fais ici référence à la technique **9-Patch**. Un exemple sera plus parlant qu'un long discours : on va utiliser l'image visible à la figure 8.6, qui est aimablement prêtée par Richard Lalancette, qui nous autorise à utiliser ses images, même pour des projets professionnels ; un grand merci à lui.

FIGURE 8.6 – Nous allons utiliser cette image pour l'exemple

Cette image ne paye pas de mine, mais elle pourra être étendue jusqu'à former une image immense sans pour autant être toute pixellisée. L'astuce consiste à indiquer

quelles parties de l'image peuvent être étendues, et le SDK d'Android contient un outil pour vous aider dans votre démarche. Par rapport à l'endroit où vous avez installé le SDK, il se trouve dans `\Android\tools\draw9patch.bat`. Vous pouvez directement glisser l'image dans l'application pour l'ouvrir ou bien aller dans `File > Open 9-patch...` (voir figure 8.7).

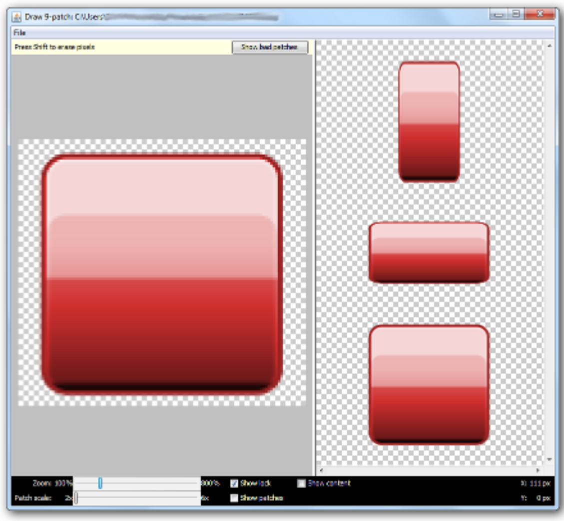


Figure 8.7 – Le logiciel Draw 9-patch

Ce logiciel contient trois zones différentes :

- La zone de gauche représente l'image et c'est dans cette zone que vous pouvez dessiner. Si, si, essayez de dessiner un gros cœur au milieu de l'image. Je vous ai eus ! Vous ne pouvez en fait dessiner que sur la partie la plus extérieure de l'image, la bordure qui fait un pixel de largeur et qui entoure l'image.
- Celle de droite est un aperçu de l'image élargie de plusieurs façons. Vous pouvez voir qu'actuellement les images agrandies sont grossières, les coins déformés et de gros pixels sont visibles.
- Et en bas on trouve plusieurs outils pour vous aider dans votre tâche.

Si vous passez votre curseur à l'intérieur de l'image, un filtre rouge s'interposera de façon à vous indiquer que vous ne devez pas dessiner à cet endroit (mais vous pouvez désactiver ce filtre avec l'option `Show lock`). En effet, l'espèce de quadrillage à côté de votre image indique les zones de transparence, celles qui ne contiennent pas de dessin. Votre rôle sera d'indiquer quels bords de l'image sont extensibles et dans quelle zone de l'objet on pourra insérer du contenu. Pour indiquer les bords extensibles on va tracer un trait d'une largeur d'un pixel sur les bords haut et gauche de l'image, alors que des traits sur les bords bas et droite déterminent où peut se placer le contenu. Par exemple pour cette image, on pourrait avoir (il n'y a pas qu'une façon de faire, faites en fonction de ce que vous souhaitez obtenir) le résultat visible à la figure 8.8.

Vous voyez la différence ? Les images étirées montrent beaucoup moins de pixels et les

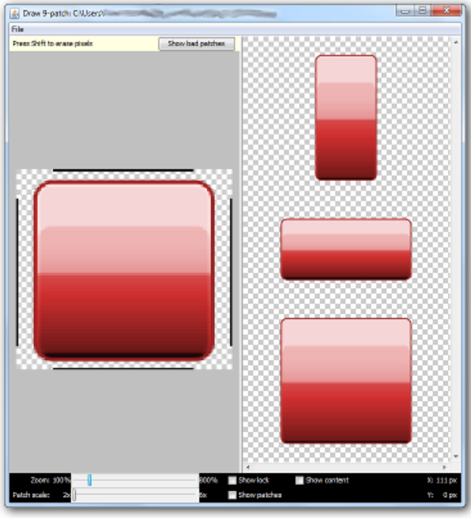


Figure 8.8 – Indiquez les zones extensibles ainsi que l'emplacement du contenu

transitions entre les couleurs sont bien plus esthétiques ! Enfin pour ajouter cette image à votre projet, il vous suffit de l'enregistrer au format `.9.png`, puis de l'ajouter à votre projet comme un drawable standard.

L'image 8.9 vous montre plus clairement à quoi correspondent les bords :

Figure 8.9 – À gauche, la zone qui peut être agrandie, à droite la zone dans laquelle on peut écrire

Les commandes

– Le slider `Zoom` vous permet de vous rapprocher ou vous éloigner de l'image originale.

- Le slider `Patch scale` vous permet de vous rapprocher ou vous éloigner des agrandissements.
- `Show patches` montre les zones qui peuvent être étendue dans la zone de dessin.
- `Show content` vous montre la zone où vous pourrez insérer du contenu (image ou texte) dans Android.
- Enfin, vous voyez un bouton en haut de la zone de dessin, `Show bad patches`, qui une fois coché vous montre les zones qui pourraient provoquer des désagréments une fois l'image agrandie ; l'objectif sera donc d'en avoir le moins possible (voire aucune).

Les styles

Souvent quand on fait une application, on adopte un certain parti pris en ce qui concerne la charte graphique. Par exemple, des tons plutôt clairs avec des boutons blancs qui font une taille de 20 pixels et dont la police du texte serait en cyan. Et pour dire qu'on veut que tous les boutons soient blancs, avec une taille de 20 pixels et le texte en cyan, il va falloir indiquer pour chaque bouton qu'on veut qu'il soit blanc, avec une taille de 20 pixels et le texte en cyan, ce qui est très vite un problème si on a beaucoup de boutons !

Afin d'éviter d'avoir à se répéter autant, il est possible de définir ce qu'on appelle un *style*. Un style est un ensemble de critères esthétiques dont l'objectif est de pouvoir définir plusieurs règles à différents éléments graphiques distincts. Ainsi, il est plus évident de créer un style « Boutons persos », qui précise que la cible est « blanche, avec une taille de 20 pixels et le texte en cyan » et d'indiquer à tous les boutons qu'on veut qu'ils soient des « Boutons persos ». Et si vous voulez mettre tous vos boutons en jaune, il suffit simplement de changer l'attribut blanc du style « Bouton persos » en jaune .

Les styles sont des *values*, on doit donc les définir au même endroit que les chaînes de caractères.

Voici la forme standard d'un style :

```
<resources>
  <style name="nom_du_style" parent="nom_du_parent">
    <item name="propriete_1">valeur_de_la_propriete_1</item>
    <item name="propriete_2">valeur_de_la_propriete_2</item>
    <item name="propriete_3">valeur_de_la_propriete_3</item>
    ...
    <item name="propriete_n">valeur_de_la_propriete_n</item>
  </style>
</resources>
```

Voici les règles à respecter :

- Comme d'habitude, on va définir un nom unique pour le style, puisqu'il y aura une variable pour y accéder.

- Il est possible d'ajouter des propriétés physiques à l'aide d'`<item>`. Le nom de l'`<item>` correspond à un des attributs destinés aux `Vues`, qu'on a déjà étudiés. On va par exemple utiliser l'attribut `android:textColor` pour changer la couleur d'un texte.
- Enfin, on peut faire hériter notre style d'un autre style — qu'il ait été défini par Android ou par vous-mêmes — et ainsi récupérer ou écraser les attributs d'un parent.

Le style suivant permet de mettre du texte en cyan :

```
1 <style name="texte_cyan">
2   <item name="android:textColor">#00FFFF</item>
3 </style>
```

Les deux styles suivants héritent du style précédent en rajoutant d'autres attributs :

```
1 <style name="texte_cyan_grand" parent="texte_cyan">
2   <!-- On récupère la couleur du texte définie par le parent --
      >
3   <item name="android:textSize">20sp</item>
4 </style>
```

```
1 <style name="texte_rouge_grand" parent="texte_cyan_grand">
2   <!-- On écrase la couleur du texte définie par le parent,
      mais on garde la taille -->
3   <item name="android:textColor">#FF0000</item>
4 </style>
```

Il est possible de n'avoir qu'un seul parent pour un style, ce qui peut être très vite pénible, alors organisez-vous à l'avance !

Il est ensuite possible d'attribuer un style à une vue en XML avec l'attribut `style="identifiant_du_style"`. Cependant, un style ne s'applique pas de manière dynamique en Java, il faut alors préciser le style à utiliser dans le constructeur. Regardez : `public View (Context contexte, AttributeSet attrs)`. Le paramètre `attrs` est facultatif, et c'est lui qui permet d'attribuer un style à une vue. Par exemple :

```
1 Button bouton = new Button (this, R.style.texte_rouge_grand);
```

Les animations

Pour donner un peu de dynamisme à notre interface graphique, on peut faire en sorte de bouger, faire tourner, agrandir ou faire disparaître une vue ou un ensemble de vues. Mais au préalable sachez qu'il est possible de placer un système de coordonnées sur notre écran de manière à pouvoir y situer les éléments. Comme à la figure 8.10, l'axe qui va de gauche à droite s'appelle l'axe X et l'axe qui va de haut en bas s'appelle l'axe Y.

Voici quelques informations utiles :

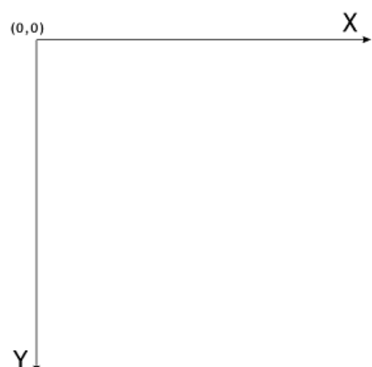


FIGURE 8.10 – L'axe horizontal est X, l'axe vertical est Y

- Sur l'axe X, plus on se déplace vers la droite, plus on s'éloigne de 0.
- Sur l'axe Y, plus on se déplace vers le bas, plus on s'éloigne de 0.
- Pour exprimer une coordonnée, on utilise la notation (X, Y).
- L'unité est le pixel.
- Le point en haut à gauche a pour coordonnées (0, 0).
- Le point en bas à droite a pour coordonnées (largeur de l'écran, hauteur de l'écran).

Définition en XML

Contrairement aux chaînes de caractères et aux styles, les animations ne sont pas des données mais des ressources indépendantes, comme l'étaient les drawables. Elles doivent être définies dans le répertoire `res/anim/`.

Pour un widget

Il existe quatre animations de base qu'il est possible d'effectuer sur une vue (que ce soit un widget ou un layout !). Une animation est décrite par un état de départ pour une vue et un état d'arrivée : par exemple on part d'une vue visible pour qu'elle devienne invisible.

Transparence

`<alpha>` permet de faire apparaître ou disparaître une vue.

- `android:fromAlpha` est la transparence de départ avec 0.0 pour une vue totalement transparente et 1.0 pour une vue totalement visible.
- `android:toAlpha` est la transparence finale voulue avec 0.0 pour une vue totalement transparente et 1.0 pour une vue totalement visible.

Rotation

<rotate> permet de faire tourner une vue autour d'un axe.

- android:fromDegrees est l'angle de départ.
- android:pivotX est la coordonnée du centre de rotation sur l'axe X (en pourcentages par rapport à la gauche de la vue, par exemple 50% correspond au milieu de la vue et 100% au bord droit).
- android:pivotY est la coordonnée du centre de rotation sur l'axe Y (en pourcentages par rapport au plafond de la vue).
- android:toDegrees est l'angle voulu à la fin.

Taille

<scale> permet d'agrandir ou de réduire une vue.

- android:fromXScale est la taille de départ sur l'axe X (1.0 pour la valeur actuelle).
- android:fromYScale est la taille de départ sur l'axe Y (1.0 pour la valeur actuelle).
- android:pivotX (identique à <rotate>).
- android:pivotY (identique à <rotate>).
- android:toXScale est la taille voulue sur l'axe X (1.0 pour la valeur de départ).
- android:toYScale est la taille voulue sur l'axe Y (1.0 pour la valeur de départ).

Mouvement

<translate> permet de faire subir une translation à une vue (mouvement rectiligne).

- android:fromXDelta est le point de départ sur l'axe X (en pourcentages).
- android:fromYDelta est le point de départ sur l'axe Y (en pourcentages).
- android:toXDelta est le point d'arrivée sur l'axe X (en pourcentages).
- android:toYDelta est le point d'arrivée sur l'axe Y (en pourcentages).

Sachez qu'il est en plus possible de regrouper les animations en un ensemble et de définir un horaire de début et un horaire de fin. Le nœud qui représente cet ensemble est de type <set>. Tous les attributs qui sont passés à ce nœud se répercuteront sur les animations qu'il contient. Par exemple :

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <set xmlns:android="http://schemas.android.com/apk/res/android"
       >
3    <scale
4      android:fromXScale="1.0"
5      android:fromYScale="1.0"
6      android:toXScale="2.0"
7      android:toYScale="0.5"
8      android:pivotX="50%"
9      android:pivotY="50%" />
10   <alpha
11     android:fromAlpha="1.0"
12     android:toAlpha="0.0" />
```

```
13 </set>
```

android:pivotX="50%" et android:pivotY="50%" permettent de placer le centre d'application de l'animation au milieu de la vue.

Dans ce code, le `scale` et l'`alpha` se feront en même temps ; cependant notre objectif va être d'effectuer d'abord le `scale`, et seulement après l'`alpha`. Pour cela, on va dire au `scale` qu'il démarrera exactement au lancement de l'animation, qu'il durera 0,3 seconde et on dira à l'`alpha` de démarrer à partir de 0,3 seconde, juste après le `scale`. Pour qu'une animation débute immédiatement, il ne faut rien faire, c'est la propriété par défaut. En revanche pour qu'elle dure 0,3 seconde, il faut utiliser l'attribut `android:duration` qui prend comme valeur la durée en millisecondes (ça veut dire qu'il vous faut multiplier le temps en secondes par 1000). Enfin, pour définir à quel moment l'`alpha` débute, c'est-à-dire avec quel retard, on utilise l'attribut `android:startOffset` (toujours en millisecondes). Par exemple, pour que le `scale` démarre immédiatement, dure 0,3 seconde et soit suivi par un `alpha` qui dure 2 secondes, voici ce qu'on écrira :

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <set xmlns:android="http://schemas.android.com/apk/res/android"
       >
3    <scale
4      android:fromXScale="1.0"
5      android:fromYScale="1.0"
6      android:toXScale="2.0"
7      android:toYScale="0.5"
8      android:pivotX="50%"
9      android:pivotY="50%"
10     android:duration="300"/>
11   <alpha
12     android:fromAlpha="1.0"
13     android:toAlpha="0.0"
14     android:startOffset="300"
15     android:duration="2000"/>
16 </set>
```

Un dernier détail. Une animation permet de donner du dynamisme à une vue, mais elle n'effectuera pas de changements réels sur l'animation : l'animation effectuera l'action, mais uniquement sur le plan visuel. Ainsi, si vous essayez ce code, Android affichera un mouvement, mais une fois l'animation finie, les vues redeviendront exactement comme elles étaient avant le début de l'animation. Heureusement, il est possible de demander à votre animation de changer les vues pour qu'elles correspondent à leur état final à la fin de l'animation. Il suffit de rajouter les deux attributs `android:fillAfter="true"` et `android:fillEnabled="true"`.

Enfin je ne vais pas abuser de votre patience, je comprendrais que vous ayez envie d'essayer votre nouveau joujou. Pour ce faire, c'est très simple, utilisez la classe `AnimationUtils`.

```
1 // On crée un utilitaire de configuration pour cette animation
2 Animation animation = AnimationUtils.loadAnimation(
      contexte_dans_lequel_se_situe_la_vue,
      identifiant_de_l_animation);
3 // On l'affecte au widget désiré, et on démarre l'animation
4 le_widget.startAnimation(animation);
```

Pour un layout

Si vous effectuez l'animation sur un layout, alors vous aurez une petite manipulation à faire. En fait, on peut très bien appliquer une animation normale à un layout avec la méthode que nous venons de voir, mais il se trouve qu'on voudra parfois faire en sorte que l'animation se propage parmi les enfants du layout pour donner un joli effet.

Tout d'abord, il vous faut créer un nouveau fichier XML, toujours dans le répertoire `res/anim`, mais la racine de celui-ci sera un nœud de type `<layoutAnimation>` (attention au « l » minuscule !). Ce nœud peut prendre trois attributs. Le plus important est `android:animation` puisqu'il faut y mettre l'identifiant de l'animation qu'on veut passer au layout. On peut ensuite définir le délai de propagation de l'animation entre les enfants à l'aide de l'attribut `android:delay`. Le mieux est d'utiliser un pourcentage, par exemple 100% pour attendre que l'animation soit finie ou 0% pour ne pas attendre. Enfin, on peut définir l'ordre dans lequel l'animation s'effectuera parmi les enfants avec `android:animationOrder`, qui peut prendre les valeurs : `normal` pour l'ordre dans lequel les vues ont été ajoutées au layout, `reverse` pour l'ordre inverse et `random` pour une distribution aléatoire entre les enfants.

On obtient alors :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <layoutAnimation xmlns:android="http://schemas.android.com/apk/
     res/android"
3   android:delay="10%"
4   android:animationOrder="random"
5   android:animation="@anim/animation_standard"
6 />
```

Puis on peut l'utiliser dans le code Java avec :

```
1 LayoutAnimationController animation = AnimationUtils.
      loadLayoutAnimation(contexte_dans_lequel_se_situe_la_vue,
      identifiant_de_l_animation);
2 layout.setLayoutAnimation(animation);
```

On aurait aussi pu passer l'animation directement au layout en XML avec l'attribut `android:layoutAnimation="identifiant_de_l_animation"`.

Un dernier raffinement : l'interpolation

Nos animations sont super, mais il manque un petit quelque chose qui pourrait les rendre encore plus impressionnantes. Si vous testez les animations, vous verrez qu'elles sont constantes, elles ne montrent pas d'effets d'accélération ou de décélération par exemple. On va utiliser ce qu'on appelle un **agent d'interpolation**, c'est-à-dire une fonction mathématique qui va calculer dans quel état doit se trouver notre animation à un moment donné pour simuler un effet particulier.

Regardez la figure 8.11 : en rouge, sans interpolation, la vitesse de votre animation reste identique pendant toute la durée de l'animation. En bleu, avec interpolation, votre animation démarrera très lentement et accélérera avec le temps. Heureusement, vous n'avez pas besoin d'être bons en maths pour utiliser les interpolateurs.

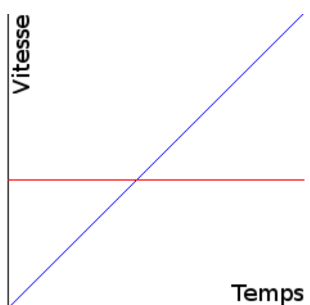


FIGURE 8.11 – La vitesse de l'animation s'accélère avec le temps

Vous pouvez rajouter un interpolateur à l'aide de l'attribut `android:interpolator`, puis vous pouvez préciser quel type d'effet vous souhaitez obtenir à l'aide d'une des valeurs suivantes :

- `@android:anim/accelerate_decelerate_interpolator` : la vitesse est identique au début et à la fin de l'animation, mais accélère au milieu.
- `@android:anim/accelerate_interpolator` : pour une animation lente au début et plus rapide par la suite.
- `@android:anim/anticipate_interpolator` : pour que l'animation commence à l'envers, puis revienne dans le bon sens.
- `@android:anim/anticipate_overshoot_interpolator` : pour que l'animation commence à l'envers, puis revienne dans le bon sens, dépasse la valeur finale puis fasse marche arrière pour l'atteindre.
- `@android:anim/bounce_interpolator` : pour un effet de rebond très sympathique.
- `@android:anim/decelerate_interpolator` : pour que l'animation démarre brutalement et se termine lentement.
- `@android:anim/overshoot_interpolator` : pour une animation qui démarre normalement, dépasse la valeur finale, puis fasse marche arrière pour l'atteindre.

Enfin, si on place un interpolateur dans un `<set>`, il est probable qu'on veuille le partager à tous les enfants de ce `<set>`. Pour propager une interpolation à tous les enfants d'un ensemble, il faut utiliser l'attribut `android:shareInterpolator="true"`.

Concernant les répétitions, il existe aussi un interpolateur, mais il y a plus pratique. Préférez plutôt la combinaison des attributs `android:repeatCount` et `android:repeatMode`. Le premier définit le nombre de répétitions de l'animation qu'on veut effectuer (-1 pour un nombre infini, 0 pour aucune répétition, et n'importe quel autre nombre entier positif pour fixer un nombre précis de répétitions), tandis que le second s'occupe de la façon dont les répétitions s'effectuent. On peut lui affecter la valeur `restart` (répétition normale) ou alors `reverse` (à la fin de l'animation, on effectue la même animation mais à l'envers).

L'évènementiel dans les animations

Il y a trois évènements qui peuvent être gérés dans le code : le lancement de l'animation, la fin de l'animation, et chaque début d'une répétition. C'est aussi simple que :

```
animation.setAnimationListener(new AnimationListener() {
  public void onAnimationEnd(Animation _animation) {
    // Que faire quand l'animation se termine ? (n'est pas lancé à la fin d'une répétition)
  }

  public void onAnimationRepeat(Animation _animation) {
    // Que faire quand l'animation se répète ?
  }

  public void onAnimationStart(Animation _animation) {
    // Que faire au premier lancement de l'animation ?
  }
});
```

En résumé

- Chaque type de ressources aura comme racine un élement `resources` qui contiendra d'autres éléments hiérarchisant les ressources. Elles peuvent être accessibles soit par la partie Java `R.type_de_ressource.nom_de_la_ressource` soit par d'autres fichiers XML `@type_de_ressource/nom_de_la_ressource`.
- Les chaînes de caractères sont déclarées par des éléments `string`.
- Android supporte 3 types d'images : PNG, JPEG et GIF, dans l'ordre du plus conseillé au moins conseillé.
- 9-Patch est une technologie permettant de rendre des images extensibles en gardant un rendu net.
- Les styles permettent de définir ou redéfinir des propriétés visuelles existantes pour les utiliser sur plusieurs vues différentes. Ils se déclarent par un élément `style` et contiennent une liste d'`item`.
- Les animations se définissent par un ensemble d'éléments :
  - `<alpha>` pour la transparence d'une vue.
  - `<rotate>` pour la rotation d'une vue autour d'un axe.

  - `<scale>` pour la modification de l'échelle d'une vue.
  - `<translate>` pour le déplacement d'une vue.
- L'animation sur un layout se fait grâce à la déclaration d'un élément `LayoutAnimation`.
- Une interpolation peut être appliquée à une animation pour modifier les variations de vitesse de l'animation.

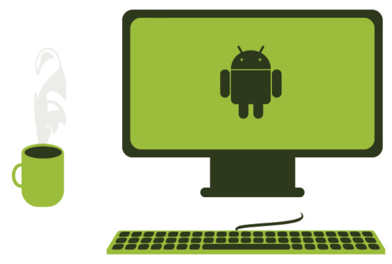
Chapitre 9

TP : un bloc-notes

Difficulté :

Notre premier TP ! Nous avions bien sûr déjà fait un petit programme avec le calculateur d'IMC, mais cette fois nous allons réfléchir à tous les détails pour faire une application qui plaira à d'éventuels utilisateurs : un bloc-notes.

En théorie, vous verrez à peu près tout ce qui a été abordé jusque là, donc s'il vous manque une information, pas de panique, on respire un bon coup et on regarde dans les chapitres précédents, en quête d'informations. Je vous donnerai évidemment la solution à ce TP, mais ce sera bien plus motivant pour vous si vous réussissez seuls. Une dernière chose : il n'existe pas *une* solution mais *des* solutions. Si vous parvenez à réaliser cette application en n'ayant pas le même code que moi, ce n'est pas grave, l'important c'est que cela fonctionne.



Objectif

L'objectif ici va être de réaliser un programme qui mettra en forme ce que vous écrivez. Cela ne sera pas très poussé : mise en gras, en italique, souligné, changement de couleur du texte et quelques smileys. Il y aura une visualisation de la mise en forme en temps réel. Le seul hic c'est que... vous ne pourrez pas enregistrer le texte, étant donné que nous n'avons pas encore vu comment faire.

Ici, on va surtout se concentrer sur l'aspect visuel du TP. C'est pourquoi nous allons essayer d'utiliser le plus de widgets et de layouts possible. Mais en plus, on va exploiter des ressources pour nous simplifier la vie sur le long terme. La figure 9.1 vous montre ce que j'obtiens. Ce n'est pas très joli, mais ça fonctionne.

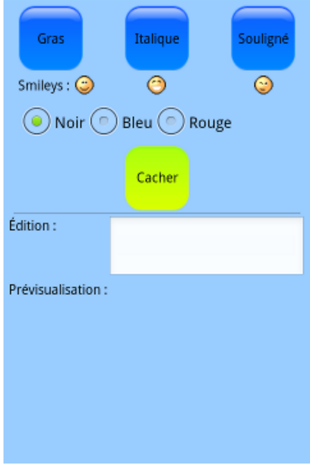


FIGURE 9.1 – Voici à quoi va ressembler l'application

Vous pouvez voir que l'écran se divise en deux zones :

- Celle en haut avec les boutons constituera le menu ;
- Celle du bas avec l'`EditText` et les `TextView`.

Le menu

Chaque bouton permet d'effectuer une des commandes de base d'un éditeur de texte. Par exemple, le bouton `Gras` met une portion du texte en gras, appuyer sur n'importe lequel des smileys permet d'insérer cette image dans le texte et les trois couleurs permettent de choisir la couleur de l'*ensemble* du texte (enfin vous pouvez le faire pour une portion du texte si vous le désirez, c'est juste plus compliqué).

Ce menu est mouvant. En appuyant sur le bouton `Cacher`, le menu se rétracte vers le haut jusqu'à disparaître. Puis, le texte sur le bouton devient « Afficher » et cliquer dessus fait redescendre le menu (voir figure 9.2).

FIGURE 9.2 – Le bouton « Afficher »

L'éditeur

Je vous en parlais précédemment, nous allons mettre en place une zone de prévisualisation qui permettra de voir le texte mis en forme en temps réel, comme sur l'image 9.3.

FIGURE 9.3 – Le texte est mis en forme en temps téel dans la zone de prévisualisation

Spécifications techniques

Fichiers à utiliser

On va d'abord utiliser les smileys du Site du Zéro, visibles à la figure 9.4.

FIGURE 9.4 – Quelques smileys du Site du Zéro

▷ Télécharger les smileys
Code web : 707148

Pour les boutons, j'ai utilisé les 9-patches visibles à la figure 9.5.

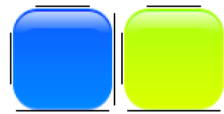


FIGURE 9.5 – Deux 9-patches

▷ Télécharger les 9-patches
Code web : 491356

Le HTML

Les balises

Comme vous avez pu le constater, nos textes seront formatés à l'aide du langage de balisage HTML. Rappelez-vous, je vous avais déjà dit qu'il était possible d'interpréter du HTML dans un `TextView` ; cependant, on va procéder un peu différemment ici comme je vous l'indiquerai plus tard.

Heureusement, vous n'avez pas à connaître le HTML, juste certaines balises de base que voici :

Effet désiré	**Balise**
Écrire en gras	`<b>Le texte</b>`
Écrire en italique	`<i>Le texte</i>`
Souligner du texte	`<u>Le texte</u>`
Insérer une image	`<img src="Nom de l'image">`
Changer la couleur de la police	`<font color="Code couleur">Le texte</font>`

L'évènementiel

Ensuite, on a dit qu'il fallait que le `TextView` interprète en temps réel le contenu de l'`EditText`. Pour cela, il suffit de faire en sorte que chaque modification de l'`EditText` provoque aussi une modification du `TextView` : c'est ce qu'on appelle un évènement. Comme nous l'avons déjà vu, pour gérer les évènements, nous allons utiliser un `Listener`. Dans ce cas précis, ce sera un objet de type `TextWatcher` qui fera l'affaire. On peut l'utiliser de cette manière :

```
editText.addTextChangedListener(new TextWatcher() {
  @Override
  /**
   * s est la chaîne de caractères qui est en train de changer
  */
  public void onTextChanged(CharSequence s, int start, int
      before, int count) {
```

```
        // Que faire au moment où le texte change ?
    }

    @Override
    /**
     * @param s La chaîne qui a été modifiée
     * @param count Le nombre de caractères concernés
     * @param start L'endroit où commence la modification dans la
         chaîne
     * @param after La nouvelle taille du texte
    */
    public void beforeTextChanged(CharSequence s, int start, int
        count, int after) {
        // Que faire juste avant que le changement de texte soit
           pris en compte ?
    }

    @Override
    /**
     * @param s L'endroit où le changement a été effectué
    */
    public void afterTextChanged(Editable s) {
        // Que faire juste après que le changement de texte a été
           pris en compte ?
    }
});
```

De plus, il nous faut penser à autre chose. L'utilisateur va vouloir appuyer sur `Entrée` pour revenir à la ligne quand il sera dans l'éditeur. Le problème est qu'en HTML il faut préciser avec une balise qu'on veut faire un retour à la ligne ! S'il appuie sur `Entrée`, aucun retour à la ligne ne sera pris en compte dans le `TextView`, alors que dans l'`EditText`, si. C'est pourquoi il va falloir faire attention aux touches que presse l'utilisateur et réagir en fonction du type de touche. Cette détection est encore un évènement, il s'agit donc encore d'un rôle pour un `Listener` : cette fois, le `OnKeyListener`. Il se présente ainsi :

```
editText.setOnKeyListener(new View.OnKeyListener() {
    /**
     * Que faire quand on appuie sur une touche ?
     * @param v La vue sur laquelle s'est effectué l'évènement
     * @param keyCode Le code qui correspond à la touche
     * @param event L'évènement en lui-même
    */
    public boolean onKey(View v, int keyCode, KeyEvent event) {
        // ...
    }
});
```

Le code pour la touche `Entrée` est 66. Le code HTML du retour à la ligne est `<br />`.

Les images

Pour pouvoir récupérer les images en HTML, il va falloir préciser à Android comment les récupérer. On utilise pour cela l'interface `Html.ImageGetter`. On va donc faire implémenter cette interface à une classe et devoir implémenter la seule méthode à implémenter : `public Drawable getDrawable (String source)`. À chaque fois que l'interpréteur HTML rencontrera une balise pour afficher une image de ce style `<img src="source">`, alors l'interpréteur donnera à la fonction `getDrawable` la source précisée dans l'attribut `src`, puis l'interpréteur affichera l'image que renvoie `getDrawable`. On a par exemple :

```
1  public class Exemple implements ImageGetter {
2    @Override
3    public Drawable getDrawable(String smiley) {
4      Drawable retour = null;
5
6      Resources resources = context.getResources();
7
8      retour = resources.getDrawable(R.drawable.ic_launcher);
9
10     // On délimite l'image (elle va de son coin en haut à
           gauche à son coin en bas à droite)
11     retour.setBounds(0, 0, retour.getIntrinsicWidth(), retour.
           getIntrinsicHeight());
12     return retour;
13   }
14 }
```

Enfin, pour interpréter le code HTML, utilisez la fonction `public Spanned Html.fromHtml(String source, Html.ImageGetter imageGetter, null)` (nous n'utiliserons pas le dernier paramètre). L'objet `Spanned` retourné est celui qui doit être inséré dans le `TextView`.

Les codes pour chaque couleur

La balise `<font color="couleur">` a besoin qu'on lui précise un code pour savoir quelle couleur afficher. Vous devez savoir que :

- Le code pour le noir est `#000000`.
- Le code pour le bleu est `#0000FF`.
- Le code pour le rouge est `#FF0000`.

L'animation

On souhaite faire en sorte que le menu se rétracte et ressorte à volonté. Le problème, c'est qu'on a besoin de la hauteur du menu pour pouvoir faire cette animation, et cette mesure n'est bien sûr pas disponible en XML. On va donc devoir faire une animation de manière programmatique.

Comme on cherche uniquement à déplacer linéairement le menu, on utilisera la classe `TranslateAnimation`, en particulier son constructeur `public TranslateAnimation (float fromXDelta, float toXDelta, float fromYDelta, float toYDelta)`. Chacun de ces paramètres permet de définir sur les deux axes (X et Y) d'où part l'animation (`from`) et jusqu'où elle va (`to`). Dans notre cas, on aura besoin de deux animations : une pour faire remonter le menu, une autre pour le faire descendre.

Pour faire remonter le menu, on va partir de sa position de départ (donc `fromXDelta = 0` et `fromYDelta = 0`, c'est-à-dire qu'on ne bouge pas le menu sur aucun des deux axes au début) et on va le déplacer sur l'axe Y jusqu'à ce qu'il sorte de l'écran (donc `toXDelta = 0` puisqu'on ne bouge pas et `toYDelta = -tailleDuMenu` puisque, rappelez-vous, l'axe Y part du haut pour aller vers le bas). Une fois l'animation terminée, on dissimule le menu avec la méthode `setVisibility(VIEW.Gone)`.

Avec un raisonnement similaire, on va d'abord remettre la visibilité à une valeur normale (`setVisibility(VIEW.Visible)`) et on déplacera la vue de son emplacement hors cadre jusqu'à son emplacement normal (donc `fromXDelta = 0`, `fromYDelta = -tailleDuMenu`, `toXDelta = 0` et `toYDelta = 0`).

Il est possible d'ajuster la vitesse avec la fonction `public void setDuration (long durationMillis)`.

Pour rajouter un interpolateur, on peut utiliser la fonction `public void setInterpolator (Interpolator i)` ; j'ai par exemple utilisé un `AccelerateInterpolator`.

Enfin, je vous conseille de créer un layout personnalisé pour des raisons pratiques. Je vous laisse imaginer un peu comment vous débrouiller ; cependant, sachez que pour utiliser une vue personnalisée dans un fichier XML, il vous faut préciser le package dans lequel elle se trouve, suivi du nom de la classe. Par exemple :

```
1 <nom.du.package.NomDeLaClasse>
```

Déboguer des applications Android

Quand on veut déboguer en Java, sans passer par le débogueur, on utilise souvent `System.out.println` afin d'afficher des valeurs et des messages dans la console. Cependant, on est bien embêté avec Android, puisqu'il n'est pas possible de faire de `System.out.println`. En effet, si vous faites un `System.out.println`, vous envoyez un message dans la console du terminal sur lequel s'exécute le programme, c'est-à-dire la console du téléphone, de la tablette ou de l'émulateur ! Et vous n'y avez pas accès avec Eclipse. Alors, qu'est-ce qui existe pour la remplacer ?

Laissez-moi vous présenter le **Logcat**. C'est un outil de l'ADT, une sorte de journal qui permet de lire des entrées, mais surtout d'en écrire. Voyons d'abord comment l'ouvrir. Dans Eclipse, allez dans `Window > Show View > Logcat`. Normalement, il s'affichera en bas de la fenêtre, dans la partie visible à la figure 9.6.

La première chose à faire, c'est de cliquer sur le troisième bouton en haut à droite (voir figure 9.7).

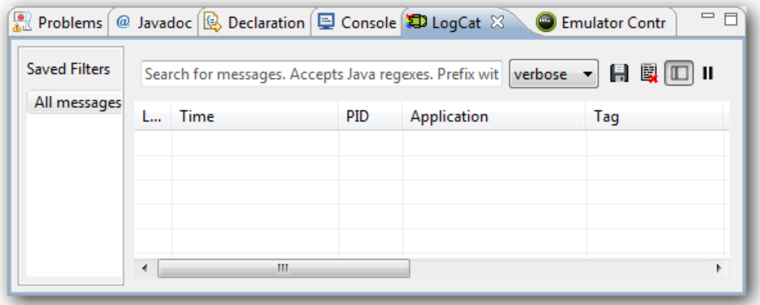


FIGURE 9.6 – Le Logcast est ouvert

FIGURE 9.7 – Cliquez sur le troisième bouton

Félicitations, vous venez de vous débarrasser d'un nombre incalculable de bugs laissés dans le Logcat ! En ce qui concerne les autres boutons, celui de gauche permet d'enregistrer le journal dans un fichier externe, le deuxième, d'effacer toutes les entrées actuelles du journal afin d'obtenir un journal vierge, et le dernier bouton permet de mettre en pause pour ne plus voir le journal défiler sans cesse.

Pour ajouter des entrées manuellement dans le Logcat, vous devez tout d'abord importer `android.util.Log` dans votre code. Vous pouvez ensuite écrire des messages à l'aide de plusieurs méthodes. Chaque message est accompagné d'une étiquette, qui permet de le retrouver facilement dans le Logcat.

- `Log.v("Étiquette", "Message à envoyer")` pour vos messages communs.
- `Log.d("Étiquette", "Message à envoyer")` pour vos messages de *debug*.
- `Log.i("Étiquette", "Message à envoyer")` pour vos messages à caractère informatif.
- `Log.w("Étiquette", "Message à envoyer")` pour vos avertissements.
- `Log.e("Étiquette", "Message à envoyer")` pour vos erreurs.

Vous pouvez ensuite filtrer les messages que vous souhaitez afficher dans le Logcat à l'aide de la liste déroulante visible à la figure 9.8.

Vous voyez, la première lettre utilisée dans le code indique un type de message : `v` pour `Verbose`, `d` pour `Debug`, etc.

Sachez aussi que, si votre programme lance une exception non catchée, c'est dans le Logcat que vous verrez ce qu'on appelle le « *stack trace* », c'est-à-dire les différents appels à des méthodes qui ont amené au lancement de l'exception.

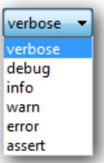


FIGURE 9.8 – Cette liste déroulante permet d'afficher dans le Logcat les messages que vous souhaitez

Par exemple avec le code :

```
Log.d("Essai", "Coucou les Zéros !");
TextView x = null;
x.setText("Va planter");
```

On obtient la figure 9.9.

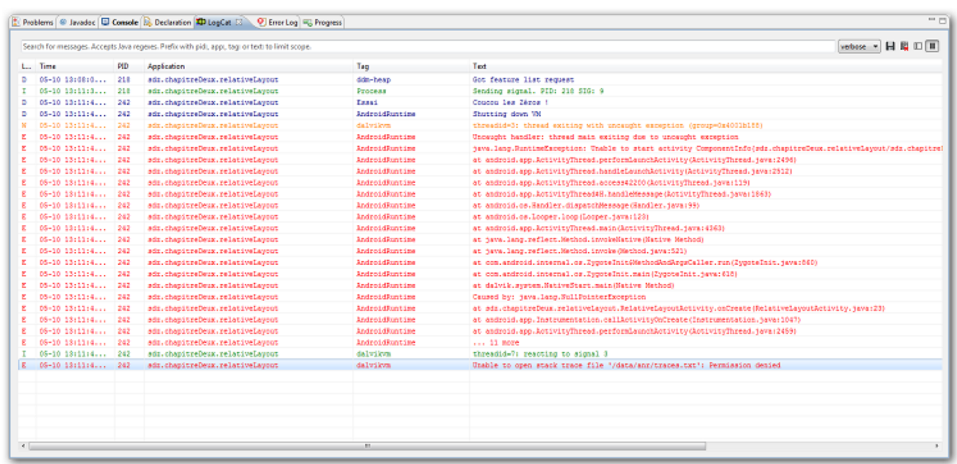


FIGURE 9.9 – Une liste d'erreurs s'affiche

À la figure 9.10, on peut voir le message que j'avais inséré.

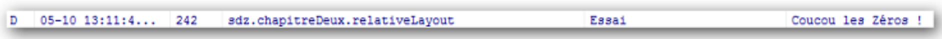


FIGURE 9.10 – le message que j'avais inséré s'affiche bien

Avec, dans les colonnes (de gauche à droite) :

- Le type de message (D pour Debug) ;
- La date et l'heure du message ;
- Le numéro unique de l'application qui a lancé le message ;
- Le package de l'application ;

– L'étiquette du message ;
– Le contenu du message.

On peut aussi voir à la figure 9.11 que mon étourderie a provoqué un plantage de l'application.

FIGURE 9.11 – L'application a planté, il suffit de regarder le message pour savoir où

Ce message signifie qu'il y a eu une exception de type `NullPointerException` (provoquée quand on veut utiliser un objet qui vaut `null`). Vous pouvez voir à la deuxième ligne que cette erreur est intervenue dans ma classe `RelativeLayoutActivity` qui appartient au package `sdz.chapitreDeux.relativeLayout`. L'erreur s'est produite dans la méthode `onCreate`, à la ligne 23 de mon code pour être précis. Enfin, pas besoin de fouiller, puisqu'un double-clic sur l'une de ces lignes permet d'y accéder directement.

Ma solution

Les ressources

Couleurs utilisées

J'ai défini une ressource de type `values` qui contient toutes mes couleurs. Elle contient :

```
1 <resources>
2   <color name="background">#99CCFF</color>
3   <color name="black">#000000</color>
4   <color name="translucide">#00000000</color>
5 </resources>
```

La couleur translucide est un peu différente des autres qui sont des nombres hexadécimaux sur 8 bits : elle est sur 8 + 2 bits. En fait, les deux bits supplémentaires expriment la transparence. Je l'ai mise à 00, comme ça elle représente les objets transparents.

Styles utilisés

Parce qu'ils sont bien pratiques, j'ai utilisé des styles, par exemple pour tous les textes qui doivent prendre la couleur noire :

```
1 <resources>
2   <style name="blueBackground">
3     <item name="android:background">@color/background</item>
4   </style>
5 
6   <style name="blackText">
```

```
      <item name="android:textColor">@color/black</item>
    </style>

    <style name="optionButton">
      <item name="android:background">@drawable/option_button</
          item>
    </style>

    <style name="hideButton">
      <item name="android:background">@drawable/hide_button</item
          >
    </style>

    <style name="translucide">
      <item name="android:background">@color/translucide</item>
    </style>
</resources>
```

Rien de très étonnant encore une fois. Notez bien que le style appelé `translucide` me permettra de mettre en transparence le fond des boutons qui affichent des smileys.

Les chaînes de caractères

Sans surprise, j'utilise des ressources pour contenir mes `string` :

```
<resources>
  <string name="app_name">Notepad</string>
  <string name="hide">Cacher</string>
  <string name="show">Afficher</string>
  <string name="bold">Gras</string>
  <string name="italic">Italique</string>
  <string name="underline">Souligné</string>
  <string name="blue">Bleu</string>
  <string name="red">Rouge</string>
  <string name="black">Noir</string>
  <string name="smileys">Smileys :</string>
  <string name="divider">Séparateur</string>
  <string name="edit">Édition :</string>
  <string name="preview">Prévisualisation : </string>
  <string name="smile">Smiley content</string>
  <string name="clin">Smiley qui fait un clin d\oeil</string>
  <string name="heureux">Smiley avec un gros sourire</string>
</resources>
```

Le `Slider`

J'ai construit une classe qui dérive de `LinearLayout` pour contenir toutes mes vues et qui s'appelle `Slider`. De cette manière, pour faire glisser le menu, je fais glisser toute l'activité et l'effet est plus saisissant. Mon `Slider` possède plusieurs attributs :

- `boolean isOpen`, pour retenir l'état de mon menu (ouvert ou fermé) ;
- `RelativeLayout toHide`, qui est le menu à dissimuler ou à afficher ;
- `final static int SPEED`, afin de définir la vitesse désirée pour mon animation.

Finalement, cette classe ne possède qu'une grosse méthode, qui permet d'ouvrir ou de fermer le menu :

```
/**
 * Utilisée pour ouvrir ou fermer le menu.
 * @return true si le menu est désormais ouvert.
*/
public boolean toggle() {
  //Animation de transition.
  TranslateAnimation animation = null;

  // On passe de ouvert à fermé (ou vice versa)
  isOpen = !isOpen;

  // Si le menu est déjà ouvert
  if (isOpen)
  {
    // Animation de translation du bas vers le haut
    animation = new TranslateAnimation(0.0f, 0.0f, -toHide.
        getHeight(), 0.0f);
    animation.setAnimationListener(openListener);
  } else
  {
    // Sinon, animation de translation du haut vers le bas
    animation = new TranslateAnimation(0.0f, 0.0f, 0.0f, -
        toHide.getHeight());
    animation.setAnimationListener(closeListener);
  }

  // On détermine la durée de l'animation
  animation.setDuration(SPEED);
  // On ajoute un effet d'accélération
  animation.setInterpolator(new AccelerateInterpolator());
  // Enfin, on lance l'animation
  startAnimation(animation);

  return isOpen;
}
```

Le layout

Tout d'abord, je rajoute un fond d'écran et un padding au layout pour des raisons esthétiques. Comme mon `Slider` se trouve dans le package `sdz.chapitreDeux.notepad`, je l'appelle avec la syntaxe `sdz.chapitreDeux.notepad.Slider` :

```
<sdz.chapitreDeux.notepad.Slider xmlns:android="http://schemas.
    android.com/apk/res/android"
  android:id="@+id/slider"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical"
  android:padding="5dip"
  style="@style/blueBackground" >
  <!-- Restant du code -->
</sdz.chapitreDeux.notepad.Slider>
```

Ensuite, comme je vous l'ai dit dans le chapitre consacré aux layouts, on va éviter de cumuler les LinearLayout, c'est pourquoi j'ai opté pour le très puissant RelativeLayout à la place :

```
<RelativeLayout
  android:id="@+id/toHide"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:layoutAnimation="@anim/main_appear"
  android:paddingLeft="10dip"
  android:paddingRight="10dip" >

  <Button
    android:id="@+id/bold"
    style="@style/optionButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:text="@string/bold"
  />

  <TextView
    android:id="@+id/smiley"
    style="@style/blackText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@id/bold"
    android:paddingTop="5dip"
    android:text="@string/smileys"
  />

  <ImageButton
    android:id="@+id/smile"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/bold"
    android:layout_toRightOf="@id/smiley"
    android:contentDescription="@string/smile"
```

```
37        android:padding="5dip"
38        android:src="@drawable/smile"
39        style="@style/translucide"
40      />
41
42      <ImageButton
43        android:id="@+id/heureux"
44        android:layout_width="wrap_content"
45        android:layout_height="wrap_content"
46        android:layout_alignTop="@id/smile"
47        android:layout_centerHorizontal="true"
48        android:contentDescription="@string/heureux"
49        android:padding="5dip"
50        android:src="@drawable/heureux"
51        style="@style/translucide"
52      />
53
54      <ImageButton
55        android:id="@+id/clin"
56        android:layout_width="wrap_content"
57        android:layout_height="wrap_content"
58        android:layout_alignTop="@id/smile"
59        android:layout_alignLeft="@+id/underline"
60        android:layout_alignRight="@+id/underline"
61        android:contentDescription="@string/clin"
62        android:padding="5dip"
63        android:src="@drawable/clin"
64        style="@style/translucide"
65      />
66
67      <Button
68        android:id="@+id/italic"
69        style="@style/optionButton"
70        android:layout_width="wrap_content"
71        android:layout_height="wrap_content"
72        android:layout_alignParentTop="true"
73        android:layout_centerHorizontal="true"
74        android:text="@string/italic"
75      />
76
77      <Button
78        android:id="@+id/underline"
79        style="@style/optionButton"
80        android:layout_width="wrap_content"
81        android:layout_height="wrap_content"
82        android:layout_alignParentTop="true"
83        android:layout_alignParentRight="true"
84        android:text="@string/underline"
85      />
86
```

```
87    <RadioGroup
88      android:id="@+id/colors"
89      android:layout_width="wrap_content"
90      android:layout_height="wrap_content"
91      android:layout_alignParentLeft="true"
92      android:layout_alignParentRight="true"
93      android:layout_below="@id/heureux"
94      android:orientation="horizontal" >
95
96      <RadioButton
97        android:id="@+id/black"
98        style="@style/blackText"
99        android:layout_width="wrap_content"
100       android:layout_height="wrap_content"
101       android:checked="true"
102       android:text="@string/black"
103     />
104     <RadioButton
105       android:id="@+id/blue"
106       style="@style/blackText"
107       android:layout_width="wrap_content"
108       android:layout_height="wrap_content"
109       android:text="@string/blue"
110     />
111     <RadioButton
112       android:id="@+id/red"
113       style="@style/blackText"
114       android:layout_width="wrap_content"
115       android:layout_height="wrap_content"
116       android:text="@string/red"
117     />
118   </RadioGroup>
119 </RelativeLayout>
```

On trouve ensuite le bouton pour actionner l'animation. On parle de l'objet au centre du layout parent (sur l'axe horizontal) avec l'attribut `android:layout_gravity="center_horizontal"`.

```
1 <Button
2   android:id="@+id/hideShow"
3   style="@style/hideButton"
4   android:layout_width="wrap_content"
5   android:layout_height="wrap_content"
6   android:paddingBottom="5dip"
7   android:layout_gravity="center_horizontal"
8   android:text="@string/hide" />
```

J'ai ensuite rajouté un séparateur pour des raisons esthétiques. C'est une `ImageView` qui affiche une image qui est présente dans le système Android ; faites de même quand vous désirez faire un séparateur facilement !

```
<ImageView
  android:src="@android:drawable/divider_horizontal_textfield"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:scaleType="fitXY"
  android:paddingLeft="5dp"
  android:paddingRight="5dp"
  android:paddingBottom="2dp"
  android:paddingTop="2dp"
  android:contentDescription="@string/divider" />
```

La seconde partie de l'écran est représentée par un `TableLayout` — plus par intérêt pédagogique qu'autre chose. Cependant, j'ai rencontré un comportement étrange (mais qui est voulu, d'après Google...). Si on veut que notre `EditText` prenne le plus de place possible dans le `TableLayout`, on doit utiliser `android:stretchColumns`, comme nous l'avons déjà vu. Cependant, avec ce comportement, le `TextView` ne fera pas de retour à la ligne automatique, ce qui fait que le texte dépasse le cadre de l'activité. Pour contrer ce désagrément, au lieu d'étendre la colonne, on la rétrécit avec `android:shrinkColumns` et on ajoute un élément invisible qui prend le plus de place possible en largeur. Regardez vous-mêmes :

```
<TableLayout
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:shrinkColumns="1" >

  <TableRow
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <TextView
      android:text="@string/edit"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      style="@style/blackText" />

    <EditText
       android:id="@+id/edit"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:gravity="top"
       android:inputType="textMultiLine"
       android:lines="5"
       android:textSize="8sp" />

  </TableRow>

  <TableRow
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
```

```
30
31      <TextView
32        android:layout_width="fill_parent"
33        android:layout_height="fill_parent"
34        android:text="@string/preview"
35        style="@style/blackText" />
36
37      <TextView
38        android:id="@+id/text"
39        android:layout_width="fill_parent"
40        android:layout_height="fill_parent"
41        android:textSize="8sp"
42        android:text=""
43        android:scrollbars="vertical"
44        android:maxLines = "100"
45        android:paddingLeft="5dip"
46        android:paddingTop="5dip"
47        style="@style/blackText" />
48
49    </TableRow>
50
51    <TableRow
52      android:layout_width="fill_parent"
53      android:layout_height="fill_parent" >
54      <TextView
55        android:layout_width="fill_parent"
56        android:layout_height="fill_parent"
57        android:text="" />
58
59      <TextView
60        android:layout_width="fill_parent"
61        android:layout_height="fill_parent"
62        android:text="                    " />
63
64    </TableRow>
65
66  </TableLayout>
```

Le code

Le SmileyGetter

On commence par la classe que j'utilise pour récupérer mes smileys dans mes drawables. On lui donne le `Context` de l'application en attribut :

```
1  /**
2   * Récupère une image depuis les ressources
3   * pour les ajouter dans l'interpréteur HTML
4  */
```

```
5  public class SmileyGetter implements ImageGetter {
6    /* Context de notre activité */
7    protected Context context = null;
8
9    public SmileyGetter(Context c) {
10     context = c;
11   }
12
13   public void setContext(Context context) {
14     this.context = context;
15   }
16
17   @Override
18   /**
19    * Donne un smiley en fonction du paramètre d'entrée
20    * @param smiley Le nom du smiley à afficher
21    */
22   public Drawable getDrawable(String smiley) {
23     Drawable retour = null;
24
25     // On récupère le gestionnaire de ressources
26     Resources resources = context.getResources();
27
28     // Si on désire le clin d'œil...
29     if(smiley.compareTo("clin") == 0)
30       // ... alors on récupère le drawable correspondant
31       retour = resources.getDrawable(R.drawable.clin);
32     else if(smiley.compareTo("smile") == 0)
33       retour = resources.getDrawable(R.drawable.smile);
34     else
35       retour = resources.getDrawable(R.drawable.heureux);
36     // On délimite l'image (elle va de son coin en haut à
         gauche à son coin en bas à droite)
37     retour.setBounds(0, 0, retour.getIntrinsicWidth(), retour.
         getIntrinsicHeight());
38     return retour;
39   }
40 }
```

L'activité

Enfin, le principal, le code de l'activité :

```
1  public class NotepadActivity extends Activity {
2    /* Récupération des éléments du GUI */
3    private Button hideShow = null;
4    private Slider slider = null;
5    private RelativeLayout toHide = null;
6    private EditText editer = null;
```

```
7     private TextView text = null;
8     private RadioGroup colorChooser = null;
9
10    private Button bold = null;
11    private Button italic = null;
12    private Button underline = null;
13
14    private ImageButton smile = null;
15    private ImageButton heureux = null;
16    private ImageButton clin = null;
17
18    /* Utilisé pour planter les smileys dans le texte */
19    private SmileyGetter getter = null;
20
21    /* Couleur actuelle du texte */
22    private String currentColor = "#000000";
23
24    @Override
25    public void onCreate(Bundle savedInstanceState) {
26      super.onCreate(savedInstanceState);
27      setContentView(R.layout.main);
28
29      getter = new SmileyGetter(this);
30
31      // On récupère le bouton pour cacher/afficher le menu
32      hideShow = (Button) findViewById(R.id.hideShow);
33      // Puis on récupère la vue racine de l'application et on
          change sa couleur
34      hideShow.getRootView().setBackgroundColor(R.color.
          background);
35      // On rajoute un Listener sur le clic du bouton...
36      hideShow.setOnClickListener(new View.OnClickListener() {
37        @Override
38        public void onClick(View vue) {
39          // ... pour afficher ou cacher le menu
40          if(slider.toggle())
41          {
42            // Si le Slider est ouvert...
43            // ... on change le texte en "Cacher"
44            hideShow.setText(R.string.hide);
45          }else
46          {
47            // Sinon on met "Afficher"
48            hideShow.setText(R.string.show);
49          }
50        }
51      });
52
53      // On récupère le menu
54      toHide = (RelativeLayout) findViewById(R.id.toHide);
```

```
55      // On récupère le layout principal
56      slider = (Slider) findViewById(R.id.slider);
57      // On donne le menu au layout principal
58      slider.setToHide(toHide);
59
60      // On récupère le TextView qui affiche le texte final
61      text = (TextView) findViewById(R.id.text);
62      // On permet au TextView de défiler
63      text.setMovementMethod(new ScrollingMovementMethod());
64
65      // On récupère l'éditeur de texte
66      editer = (EditText) findViewById(R.id.edit);
67      // On ajoute un Listener sur l'appui de touches
68      editer.setOnKeyListener(new View.OnKeyListener() {
69        @Override
70        public boolean onKey(View v, int keyCode, KeyEvent event)
              {
71          // On récupère la position du début de la sélection
              dans le texte
72          int cursorIndex = editer.getSelectionStart();
73          // Ne réagir qu'à l'appui sur une touche (et pas au rel
              âchement)
74          if(event.getAction() == 0)
75            // S'il s'agit d'un appui sur la touche « entrée »
76            if(keyCode == 66)
77              // On insère une balise de retour à la ligne
78              editer.getText().insert(cursorIndex, "<br />");
79          return true;
80        }
81      });
82      // On ajoute un autre Listener sur le changement, dans le
          texte cette fois
83      editer.addTextChangedListener(new TextWatcher() {
84        @Override
85        public void onTextChanged(CharSequence s, int start, int
            before, int count) {
86          // Le Textview interprète le texte dans l'éditeur en
              une certaine couleur
87          text.setText(Html.fromHtml("<font color=\"" +
              currentColor + "\">" + editer.getText().toString() +
              "</font>", getter, null));
88        }
89
90        @Override
91        public void beforeTextChanged(CharSequence s, int start,
            int count, int after) {
92
93        }
94
95        @Override
```

```
96          public void afterTextChanged(Editable s) {
97
98          }
99        });
100
101
102       // On récupère le RadioGroup qui gère la couleur du texte
103       colorChooser = (RadioGroup) findViewById(R.id.colors);
104       // On rajoute un Listener sur le changement de RadioButton
            sélectionné
105       colorChooser.setOnCheckedChangeListener(new RadioGroup.
            OnCheckedChangeListener() {
106         @Override
107         public void onCheckedChanged(RadioGroup group, int
              checkedId) {
108           // En fonction de l'identifiant du RadioButton sé
                lectionné...
109           switch(checkedId)
110           {
111             // On change la couleur actuelle pour noir
112             case R.id.black:
113               currentColor = "#000000";
114               break;
115             // On change la couleur actuelle pour bleu
116             case R.id.blue:
117               currentColor = "#0022FF";
118               break;
119             // On change la couleur actuelle pour rouge
120             case R.id.red:
121               currentColor = "#FF0000";
122           }
123           /*
124            * On met dans l'éditeur son texte actuel
125            * pour activer le Listener de changement de texte
126           */
127           editer.setText(editer.getText().toString());
128         }
129       });
130
131       smile = (ImageButton) findViewById(R.id.smile);
132       smile.setOnClickListener(new View.OnClickListener() {
133         @Override
134         public void onClick(View v) {
135           // On récupère la position du début de la sélection
                dans le texte
136           int selectionStart = editer.getSelectionStart();
137           // Et on insère à cette position une balise pour
                afficher l'image du smiley
138           editer.getText().insert(selectionStart, "<img src=\"
                smile\" >");
```

```
139          }
140        });
141
142        heureux =(ImageButton) findViewById(R.id.heureux);
143        heureux.setOnClickListener(new View.OnClickListener() {
144          @Override
145          public void onClick(View v) {
146            // On récupère la position du début de la sélection
147            int selectionStart = editer.getSelectionStart();
148            editer.getText().insert(selectionStart, "<img src=\"
                  heureux\" >");
149          }
150        });
151
152        clin = (ImageButton) findViewById(R.id.clin);
153        clin.setOnClickListener(new View.OnClickListener() {
154          @Override
155          public void onClick(View v) {
156            //On récupère la position du début de la sélection
157            int selectionStart = editer.getSelectionStart();
158            editer.getText().insert(selectionStart, "<img src=\"
                  clin\" >");
159          }
160        });
161
162        bold = (Button) findViewById(R.id.bold);
163        bold.setOnClickListener(new View.OnClickListener() {
164          @Override
165          public void onClick(View vue) {
166            // On récupère la position du début de la sélection
167            int selectionStart = editer.getSelectionStart();
168            // On récupère la position de la fin de la sélection
169            int selectionEnd = editer.getSelectionEnd();
170
171            Editable editable = editer.getText();
172
173            // Si les deux positions sont identiques (pas de sé
                  lection de plusieurs caractères)
174            if(selectionStart == selectionEnd)
175              //On insère les balises ouvrante et fermante avec
                    rien dedans
176              editable.insert(selectionStart, "<b></b>");
177            else
178            {
179              // On met la balise avant la sélection
180              editable.insert(selectionStart, "<b>");
181              // On rajoute la balise après la sélection (et après
                    les 3 caractères de la balise <b>)
182              editable.insert(selectionEnd + 3, "</b>");
183            }
```

```
184         }
185       });
186 
187       italic = (Button) findViewById(R.id.italic);
188       italic.setOnClickListener(new View.OnClickListener() {
189         @Override
190         public void onClick(View vue) {
191           // On récupère la position du début de la sélection
192           int selectionStart = editer.getSelectionStart();
193           // On récupère la position de la fin de la sélection
194           int selectionEnd = editer.getSelectionEnd();
195 
196           Editable editable = editer.getText();
197 
198           // Si les deux positions sont identiques (pas de sé
                  lection de plusieurs caractères)
199           if(selectionStart == selectionEnd)
200             //On insère les balises ouvrante et fermante avec
                    rien dedans
201             editable.insert(selectionStart, "<i></i>");
202           else
203           {
204             // On met la balise avant la sélection
205             editable.insert(selectionStart, "<i>");
206             // On rajoute la balise après la sélection (et après
                    les 3 caractères de la balise <b>)
207             editable.insert(selectionEnd + 3, "</i>");
208           }
209         }
210       });
211 
212        underline = (Button) findViewById(R.id.underline);
213        underline.setOnClickListener(new View.OnClickListener() {
214         @Override
215         public void onClick(View vue) {
216           // On récupère la position du début de la sélection
217           int selectionStart = editer.getSelectionStart();
218           // On récupère la position de la fin de la sélection
219           int selectionEnd = editer.getSelectionEnd();
220 
221           Editable editable = editer.getText();
222 
223           // Si les deux positions sont identiques (pas de sé
                  lection de plusieurs caractères)
224           if(selectionStart == selectionEnd)
225             // On insère les balises ouvrante et fermante avec
                    rien dedans
226             editable.insert(selectionStart, "<u></u>");
227           else
228           {
```

```
229             // On met la balise avant la sélection
230             editable.insert(selectionStart, "<u>");
231             // On rajoute la balise après la sélection (et après
                    les 3 caractères de la balise <b>)
232             editable.insert(selectionEnd + 3, "</u>");
233           }
234         }
235       });
236     }
237 }
```

▷ Télécharger le projet
Code web : 229796

Objectifs secondaires

Boutons à plusieurs états

En testant votre application, vous verrez qu'en cliquant sur un bouton, il conserve sa couleur et ne passe pas orange, comme les vrais boutons Android. Le problème est que l'utilisateur risque d'avoir l'impression que son clic ne fait rien, il faut donc lui fournir un moyen d'avoir un retour. On va faire en sorte que nos boutons changent de couleur quand on clique dessus. Pour cela, on va avoir besoin du **9-Patch** visible à la figure 9.12.

FIGURE 9.12 – Ce bouton va nous permettre de modifier la couleur d'un bouton appuyé

Comment faire pour que le bouton prenne ce fond quand on clique dessus ? On va utiliser un type de drawable que vous ne connaissez pas, les *state lists*. Voici ce qu'on peut obtenir à la fin :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <selector xmlns:android="http://schemas.android.com/apk/res/
      android" >
3   <item android:state_pressed="true"
4     android:drawable="@drawable/pressed" />
5   <item android:drawable="@drawable/number" />
6 </selector>
```

On a une racine `<selector>` qui englobe des `<item>`, et chaque `<item>` correspond à un état. Le principe est qu'on va associer chaque état à une image différente. Ainsi, le premier état `<item android:state_pressed="true" android:drawable="@drawable/pressed" />` indique que, quand le bouton est dans l'état « pressé », on utilise le drawable d'identifiant `pressed` (qui correspond à une image qui s'appelle `pressed.9.png`).

Le second item, `<item android:drawable="@drawable/number" />`, n'a pas d'état associé, c'est donc l'état par défaut. Si Android ne trouve pas d'état qui correspond à l'état actuel du bouton, alors il utilisera celui-là.

En parcourant le XML, Android s'arrêtera dès qu'il trouvera un attribut qui correspond à l'état actuel, et, comme je vous l'ai déjà dit, il n'existe que deux attributs qui peuvent correspondre à un état : soit l'attribut qui correspond à l'état, soit l'état par défaut, celui qui n'a pas d'attribut. Il faut donc que l'état par défaut soit le dernier de la liste, sinon Android s'arrêtera à chaque fois qu'il tombe dessus, et ne cherchera pas dans les `<item>` suivants.

Internationalisation

Pour toucher le plus de gens possible, il vous est toujours possible de traduire votre application en anglais ! Même si, je l'avoue, il n'y a rien de bien compliqué à comprendre.

Gérer correctement le mode paysage

Et si vous tournez votre téléphone en mode paysage (`Ctrl` + `F11` avec l'émulateur) ? Eh oui, ça ne passe pas très bien. Mais vous savez comment procéder, n'est-ce pas ?

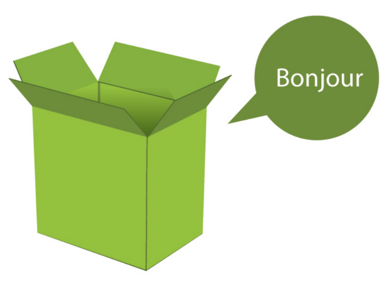
Chapitre 10

Des widgets plus avancés et des boîtes de dialogue

Difficulté :

On a vu dans un chapitre précédent les vues les plus courantes et les plus importantes. Mais le problème est que vous ne pourrez pas tout faire avec les éléments précédemment présentés. Je pense en particulier à une structure de données fondamentale pour représenter un ensemble de données... je parle bien entendu des listes.

On verra aussi les boîtes de dialogue, qui sont utilisées dans énormément d'applications. Enfin, je vous présenterai de manière un peu moins détaillée d'autres éléments, moins répandus mais qui pourraient éventuellement vous intéresser.



Les listes et les adaptateurs

N'oubliez pas que le Java est un langage orienté objet et que par conséquent il pourrait vous arriver d'avoir à afficher une liste d'un type d'objet particulier, des livres par exemple. Il existe plusieurs paramètres à prendre en compte dans ce cas-là. Tout d'abord, quelle est l'information à afficher pour chaque livre ? Le titre ? L'auteur ? Le genre littéraire ? Et que faire quand on clique sur un élément de la liste ? Et l'esthétique dans tout ça, c'est-à-dire comment sont représentés les livres ? Affiche-t-on leur couverture avec leur titre ? Ce sont autant d'éléments à prendre en compte quand on veut afficher une liste.

La gestion des listes se divise en deux parties distinctes. Tout d'abord les `Adapter` (que j'appellerai **adaptateurs**), qui sont les objets qui gèrent les données, mais pas leur affichage ou leur comportement en cas d'interaction avec l'utilisateur. On peut considérer un adaptateur comme un intermédiaire entre les données et la vue qui représente ces données. De l'autre côté, on trouve les `AdapterView`, qui, eux, vont gérer l'affichage et l'interaction avec l'utilisateur, mais sur lesquels on ne peut pas effectuer d'opération de modification des données.

Le comportement typique pour afficher une liste depuis un ensemble de données est celui-ci : on donne à l'adaptateur une liste d'éléments à traiter et la manière dont ils doivent l'être, puis on passe cet adaptateur à un `AdapterView`, comme schématisé à la figure 10.1. Dans ce dernier, l'adaptateur va créer un widget pour chaque élément en fonction des informations fournies en amont.

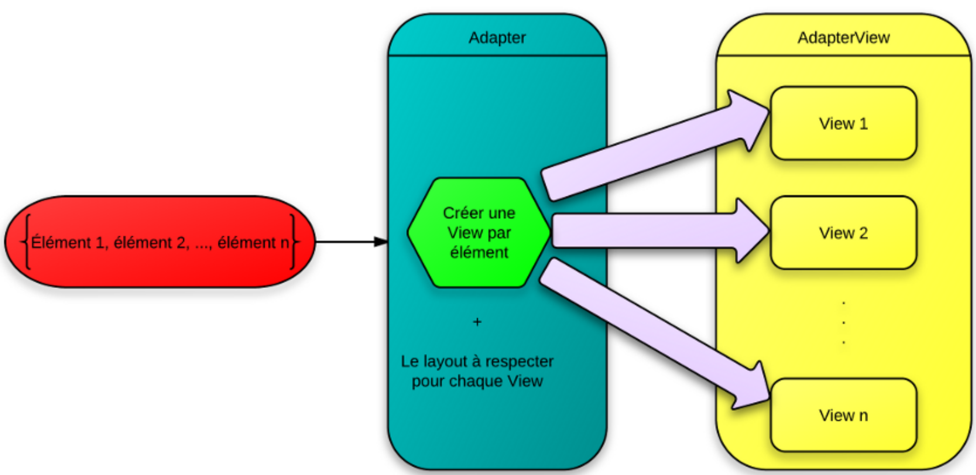


FIGURE 10.1 – Schéma du fonctionnement des `Adapter` et `AdapterView`

L'ovale rouge représente la liste des éléments. On la donne à l'adaptateur, qui se charge de créer une vue pour chaque élément, avec le layout à respecter. Puis, les vues sont fournies à un `AdapterView` (toutes au même instant, bien entendu), où elles seront affichées dans l'ordre fourni et avec le layout correspondant. L'`AdapterView` possède

lui aussi un layout afin de le personnaliser.

Savez-vous ce qu'est une fonction *callback* (vous trouverez peut-être aussi l'expression « fonction de rappel ») ? Pour simplifier les choses, c'est une fonction qu'on n'appelle pas directement, c'est une autre fonction qui y fera appel. On a déjà vu une fonction de *callback* dans la section qui parlait de l'évènementiel chez les widgets : quand vous cliquez sur un bouton, la fonction `onTouch` est appelée, alors qu'on n'y fait pas appel nous-mêmes. Dans cette prochaine section figure aussi une fonction de *callback*, je tiens juste à être certain que vous connaissiez bien le terme.

Les adaptateurs

`Adapter` n'est en fait qu'une interface qui définit les comportements généraux des adaptateurs. Cependant, si vous voulez un jour construire un adaptateur, faites le dériver de `BaseAdapter`.

Si on veut construire un widget simple, on retiendra trois principaux adaptateurs :

1. `ArrayAdapter`, qui permet d'afficher les informations simples ;
2. `SimpleAdapter` est quant à lui utile dès qu'il s'agit d'écrire plusieurs informations pour chaque élément (s'il y a deux textes dans l'élément par exemple) ;
3. `CursorAdapter`, pour adapter le contenu qui provient d'une base de données. On y reviendra dès qu'on abordera l'accès à une base de données.

Les listes simples : `ArrayAdapter`

La classe `ArrayAdapter` se trouve dans le package `android.widget.ArrayAdapter`.

On va considérer le constructeur suivant : `public ArrayAdapter (Context contexte, int id, T[] objects)` ou encore `public ArrayAdapter (Context contexte, int id, List<T> objects)`. Pour vous aider, voici la signification de chaque paramètre :

- Vous savez déjà ce qu'est le `contexte`, ce sont des informations sur l'activité, on passe donc l'activité.
- Quant à `id`, il s'agira d'une référence à un layout. C'est donc elle qui déterminera la mise en page de l'élément. Vous pouvez bien entendu créer une ressource de layout par vous-mêmes, mais Android met à disposition certains layouts, qui dépendent beaucoup de la liste dans laquelle vont se trouver les widgets.
- `objects` est la liste ou le tableau des éléments à afficher.

T[] signifie qu'il peut s'agir d'un tableau de n'importe quel type d'objet ; de manière similaire List<T> signifie que les objets de la liste peuvent être de n'importe quel type. Attention, j'ai dit « les objets », donc pas de primitives (comme int ou float par exemple) auquel cas vous devrez passer par des objets équivalents (comme Integer ou Float).

Des listes plus complexes : SimpleAdapter

On peut utiliser la classe SimpleAdapter à partir du package android.widget.SimpleAdapter.

Le SimpleAdapter est utile pour afficher simplement plusieurs informations par élément. En réalité, pour chaque information de l'élément on aura une vue dédiée qui affichera l'information voulue. Ainsi, on peut avoir du texte, une image... ou même une autre liste si l'envie vous en prend. Mieux qu'une longue explication, voici l'exemple d'un répertoire téléphonique :

```
1  import java.util.ArrayList;
2  import java.util.HashMap;
3  import java.util.List;
4  import android.app.Activity;
5  import android.os.Bundle;
6  import android.widget.ListAdapter;
7  import android.widget.ListView;
8  import android.widget.SimpleAdapter;
9
10 public class ListesActivity extends Activity {
11   ListView vue;
12
13   @Override
14   public void onCreate(Bundle savedInstanceState) {
15     super.onCreate(savedInstanceState);
16     setContentView(R.layout.activity_main);
17
18     //On récupère une ListView de notre layout en XML, c'est la
          vue qui représente la liste
19     vue = (ListView) findViewById(R.id.listView);
20
21     /*
22      * On entrepose nos données dans un tableau qui contient
          deux colonnes :
23      *  - la première contiendra le nom de l'utilisateur
24      *  - la seconde contiendra le numéro de téléphone de l'
          utilisateur
25     */
26     String[][] repertoire = new String[][]{
27       {"Bill Gates", "06 06 06 06 06"},
28       {"Niels Bohr", "05 05 05 05 05"},
29       {"Alexandre III de Macédoine", "04 04 04 04 04"}};
```

```
30
31      /*
32       * On doit donner à notre adaptateur une liste du type «
             List<Map<String, ?> » :
33       *  - la clé doit forcément être une chaîne de caractères
34       *  - en revanche, la valeur peut être n'importe quoi, un
             objet ou un entier par exemple,
35       *  si c'est un objet, on affichera son contenu avec la mé
             thode « toString() »
36       *
37       * Dans notre cas, la valeur sera une chaîne de caractères,
              puisque le nom et le numéro de téléphone
38       * sont entreposés dans des chaînes de caractères
39      */
40      List<HashMap<String, String>> liste = new ArrayList<HashMap
           <String, String>>();
41
42      HashMap<String, String> element;
43      //Pour chaque personne dans notre répertoire...
44      for(int i = 0 ; i < repertoire.length ; i++) {
45        //... on crée un élément pour la liste...
46        element = new HashMap<String, String>();
47        /*
48         * ... on déclare que la clé est « text1 » (j'ai choisi
               ce mot au hasard, sans sens technique particulier)
49         * pour le nom de la personne (première dimension du
               tableau de valeurs)...
50        */
51        element.put("text1", repertoire[i][0]);
52        /*
53         * ... on déclare que la clé est « text2 »
54         * pour le numéro de cette personne (seconde dimension du
                tableau de valeurs)
55        */
56        element.put("text2", repertoire[i][1]);
57        liste.add(element);
58      }
59
60      ListAdapter adapter = new SimpleAdapter(this,
61        //Valeurs à insérer
62        liste,
63        /*
64         * Layout de chaque élément (là, il s'agit d'un layout
               par défaut
65         * pour avoir deux textes l'un au-dessus de l'autre, c'
               est pourquoi on
66         * n'affiche que le nom et le numéro d'une personne)
67        */
68        android.R.layout.simple_list_item_2,
69        /*
```

```
         * Les clés des informations à afficher pour chaque élément :
         *  - la valeur associée à la clé « text1 » sera la première information
         *  - la valeur associée à la clé « text2 » sera la seconde information
        */
        new String[] {"text1", "text2"},
        /*
         * Enfin, les layouts à appliquer à chaque widget de notre élément
         * (ce sont des layouts fournis par défaut) :
         *  - la première information appliquera le layout « android.R.id.text1 »
         *  - la seconde information appliquera le layout « android.R.id.text2 »
        */
        new int[] {android.R.id.text1, android.R.id.text2 });
      //Pour finir, on donne à la ListView le SimpleAdapter
      vue.setAdapter(adapter);
    }
}
```

Ce qui donne la figure 10.2.

FIGURE 10.2 – Le résultat en image

On a utilisé le constructeur `public SimpleAdapter(Context context, List<? extends Map<String,?» data, int ressource, String[] from, int[] to)`.

Quelques méthodes communes à tous les adaptateurs

Tout d'abord, pour ajouter un objet à un adaptateur, on peut utiliser la méthode `void add (T object)` ou l'insérer à une position particulière avec `void insert (T object, int position)`. Il est possible de récupérer un objet dont on connaît la position avec la

méthode `T getItem (int position)`, ou bien récupérer la position d'un objet précis avec la méthode `int getPosition (T object)`.

On peut supprimer un objet avec la méthode `void remove (T object)` ou vider complètement l'adaptateur avec `void clear()`.

Par défaut, un `ArrayAdapter` affichera pour chaque objet de la liste le résultat de la méthode `String toString()` associée et l'insérera dans une `TextView`.

Voici un exemple de la manière d'utiliser ces codes :

```
1  // On crée un adaptateur qui fonctionne avec des chaînes de
       caractères
2  ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
       android.R.layout.simple_list_item_1);
3  // On rajoute la chaîne de caractères "Pommes"
4  adapter.add("Pommes");
5  // On récupère la position de la chaîne dans l'adaptateur.
       Comme il n'y a pas d'autres chaînes dans l'adaptateur,
       position vaudra 0
6  int position = adapter.getPosition("Pommes");
7  // On affiche la valeur et la position de la chaîne de caractè
       res
8  Toast.makeText(this, "Les " + adapter.getItem(position) + " se
       trouvent à la position " + position + ".", Toast.LENGTH_LONG
       ).show();
9  // Puis on la supprime, n'en n'ayant plus besoin
10 adapter.remove("Pommes");
```

Les vues responsables de l'affichage des listes : les `AdapterView`

On trouve la classe `AdapterView` dans le package `android.widget.AdapterView`.

Alors que l'adaptateur se chargera de construire les sous-éléments, c'est l'`AdapterView` qui liera ces sous-éléments et qui fera en sorte de les afficher en une liste. De plus, c'est l'`AdapterView` qui gérera les interactions avec les utilisateurs : l'adaptateur s'occupe des éléments en tant que données, alors que l'`AdapterView` s'occupe de les afficher et veille aux interactions avec un utilisateur.

On observe trois principaux `AdapterView` :

1. `ListView`, pour simplement afficher des éléments les uns après les autres ;
2. `GridView`, afin d'organiser les éléments sous la forme d'une grille ;
3. `Spinner`, qui est une liste défilante.

Pour associer un adaptateur à une `AdapterView`, on utilise la méthode `void setAdapter (Adapter adapter)`, qui se chargera de peupler la vue, comme vous le verrez dans quelques instants.

Les listes standards : `ListView`

On les trouve dans le package `android.widget.ListView`. Elles affichent les éléments les uns après les autres, comme à la figure 10.3.

Le layout de base est `android.R.layout.simple_list_item_1`.

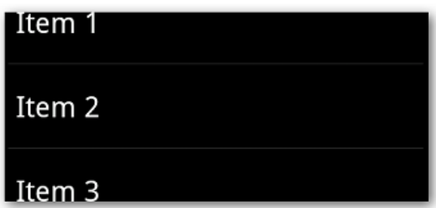


FIGURE 10.3 – Une liste simple

L'exemple précédent est obtenu à l'aide de ce code :

```
import java.util.ArrayList;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class TutoListesActivity extends Activity {
  ListView liste = null;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    liste = (ListView) findViewById(R.id.listView);
    List<String> exemple = new ArrayList<String>();
    exemple.add("Item 1");
    exemple.add("Item 2");
    exemple.add("Item 3");

    ArrayAdapter<String> adapter = new ArrayAdapter<String>(
        this, android.R.layout.simple_list_item_1, exemple);
    liste.setAdapter(adapter);
  }
}
```

Au niveau évènementiel, il est toujours possible de gérer plusieurs types de clic, comme par exemple :

```
void setOnItemClickListener (AdapterView.OnItemClickListener
    listener)
```

```
2 // Pour un clic simple sur un élément de la liste.
3 // La fonction de callback associée est :
4 // void onItemClick (AdapterView<?> adapter, View view, int
     position, long id) // Avec adapter l'AdapterView qui
     contient la vue sur laquelle le clic a été effectué, view
     qui est la vue en elle-même, position qui est la position de
      la vue dans la liste et enfin id qui est l'identifiant de
     la vue.
5
6 void setOnItemLongClickListener (AdapterView.
     OnItemLongClickListener listener)
7 // Pour un clic prolongé sur un élément de la liste. La
     fonction de callback associée est boolean onItemLongClick (
     AdapterView<?> adapter, View view, int position, long id)
```

Ce qui donne :

```
1 listView.setOnItemClickListener(new AdapterView.
     OnItemClickListener() {
2   @Override
3   public void onItemClick(AdapterView<?> adapterView,
4     View view,
5     int position,
6     long id) {
7       // Que faire quand on clique sur un élément de la liste ?
8   }
9 });
```

En revanche il peut arriver qu'on ait besoin de sélectionner un ou plusieurs éléments. Tout d'abord, il faut indiquer à la liste quel mode de sélection elle accepte. On peut le préciser en XML à l'aide de l'attribut `android:choiceMode` qui peut prendre les valeurs `singleChoice` (sélectionner un seul élément) ou `multipleChoice` (sélectionner plusieurs éléments). En Java, il suffit d'utiliser la méthode `void setChoiceMode(int mode)` avec `mode` qui peut valoir `ListView.CHOICE_MODE_SINGLE` (sélectionner un seul élément) ou `ListView.CHOICE_MODE_MULTIPLE` (sélectionner plusieurs éléments).

À nouveau, il nous faut choisir un layout adapté. Pour les sélections uniques, on peut utiliser `android.R.layout.simple_list_item_single_choice`, ce qui donnera la figure 10.4.

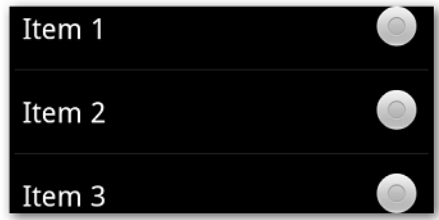


FIGURE 10.4 – Une liste de sélection unique

Pour les sélections multiples, on peut utiliser android.R.layout.simple_list_item_multiple_choice, ce qui donnera la figure 10.5.

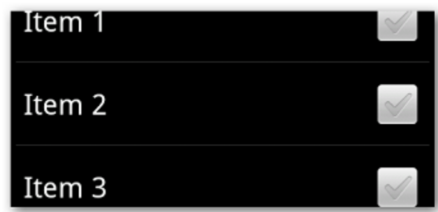


FIGURE 10.5 – Une liste de sélection multiple

Enfin, pour récupérer le rang de l'élément sélectionné dans le cas d'une sélection unique, on peut utiliser la méthode int getCheckedItemPosition() et dans le cas d'une sélection multiple, SparseBooleanArray getCheckedItemPositions().

Un SparseBooleanArray est un tableau associatif dans lequel on associe un entier à un booléen, c'est-à-dire que c'est un équivalent à la structure Java standard Hashmap<Integer, Boolean>, mais en plus optimisé. Vous vous rappelez ce que sont les *hashmaps*, les tableaux associatifs ? Ils permettent d'associer une clé (dans notre cas un Integer) à une valeur (dans ce cas-ci un Boolean) afin de retrouver facilement cette valeur. La clé n'est pas forcément un entier, on peut par exemple associer un nom à une liste de prénoms avec Hashmap<String, ArrayList<String» afin de retrouver les prénoms des gens qui portent un nom en commun.

En ce qui concerne les SparseBooleanArray, il est possible de vérifier la valeur associée à une clé entière avec la méthode boolean get(int key). Par exemple dans notre cas de la sélection multiple, on peut savoir si le troisième élément de la liste est sélectionné en faisant liste.getCheckedItemPositions().get(3), et, si le résultat vaut true, alors l'élément est bien sélectionné dans la liste.

Application

Voici un petit exemple qui vous montre comment utiliser correctement tous ces attributs. Il s'agit d'une application qui réalise un sondage. L'utilisateur doit indiquer son sexe et les langages de programmation qu'il maîtrise. Notez que, comme l'application est destinée aux Zéros qui suivent ce tuto, par défaut on sélectionne le sexe masculin et on déclare que l'utilisateur connaît le Java !

Dès que l'utilisateur a fini d'entrer ses informations, il peut appuyer sur un bouton pour confirmer sa sélection. Ce faisant, on empêche l'utilisateur de changer ses informations en enlevant les boutons de sélection et en l'empêchant d'appuyer à nouveau sur le bouton, comme le montre la figure 10.6.

Solution

Le layout :

FIGURE 10.6 – À gauche, au démarrage de l'application ; à droite, après avoir appuyé sur le bouton « Envoyer »

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <TextView
    android:id="@+id/textSexe"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Quel est votre sexe :" />

    <!-- On choisit le mode de sélection avec android:
        choiceMode -->
    <ListView
      android:id="@+id/listSexe"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:choiceMode="singleChoice" >
    </ListView>

    <TextView
      android:id="@+id/textProg"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Quel(s) langage(s) maîtrisez-vous :" />

    <ListView
```

```
28        android:id="@+id/listProg"
29        android:layout_width="fill_parent"
30        android:layout_height="wrap_content"
31        android:choiceMode="multipleChoice" >
32      </ListView>
33
34      <Button
35        android:id="@+id/send"
36        android:layout_width="wrap_content"
37        android:layout_height="wrap_content"
38        android:layout_gravity="center"
39        android:text="Envoyer" />
40
41  </LinearLayout>
```

Et le code :

```
1   package sdz.exemple.selectionMultiple;
2
3   import android.app.Activity;
4   import android.os.Bundle;
5   import android.view.View;
6   import android.widget.ArrayAdapter;
7   import android.widget.Button;
8   import android.widget.ListView;
9   import android.widget.Toast;
10
11  public class SelectionMultipleActivity extends Activity {
12    /** Affichage de la liste des sexes **/
13    private ListView mListSexe = null;
14    /** Affichage de la liste des langages connus **/
15    private ListView mListProg = null;
16    /** Bouton pour envoyer le sondage **/
17    private Button mSend = null;
18
19    /** Contient les deux sexes **/
20    private String[] mSexes = {"Masculin", "Feminin"};
21    /** Contient différents langages de programmation **/
22    private String[] mLangages = null;
23
24    @Override
25    public void onCreate(Bundle savedInstanceState) {
26      super.onCreate(savedInstanceState);
27      setContentView(R.layout.main);
28
29      //On récupère les trois vues définies dans notre layout
30      mListSexe = (ListView) findViewById(R.id.listSexe);
31      mListProg = (ListView) findViewById(R.id.listProg);
32      mSend = (Button) findViewById(R.id.send);
33
```

```
34        //Une autre manière de créer un tableau de chaînes de
              caractères
35        mLangages = new String[]{"C", "Java", "COBOL", "Perl"};
36
37        //On ajoute un adaptateur qui affiche des boutons radio (c'
              est l'affichage à considérer quand on ne peut
38        //sélectionner qu'un élément d'une liste)
39        mListSexe.setAdapter(new ArrayAdapter<String>(this, android
              .R.layout.simple_list_item_single_choice, mSexes));
40        //On déclare qu'on sélectionne de base le premier élément (
              Masculin)
41        mListSexe.setItemChecked(0, true);
42
43        //On ajoute un adaptateur qui affiche des cases à cocher (c
              'est l'affichage à considérer quand on peut sélectionner
44        //autant d'éléments qu'on veut dans une liste)
45        mListProg.setAdapter(new ArrayAdapter<String>(this, android
              .R.layout.simple_list_item_multiple_choice, mLangages));
46        //On déclare qu'on sélectionne de base le second élément (F
              éminin)
47        mListProg.setItemChecked(1, true);
48
49        //Que se passe-t-il dès qu'on clique sur le bouton ?
50        mSend.setOnClickListener(new View.OnClickListener() {
51
52          @Override
53          public void onClick(View v) {
54            Toast.makeText(SelectionMultipleActivity.this, "Merci !
                  Les données ont été envoyées !", Toast.LENGTH_LONG)
                  .show();
55
56            //On déclare qu'on ne peut plus sélectionner d'élément
57            mListSexe.setChoiceMode(ListView.CHOICE_MODE_NONE);
58            //On affiche un layout qui ne permet pas de sélection
59            mListSexe.setAdapter(new ArrayAdapter<String>(
                  SelectionMultipleActivity.this, android.R.layout.
                  simple_list_item_1,
60                            mSexes));
61
62            //On déclare qu'on ne peut plus sélectionner d'élément
63            mListProg.setChoiceMode(ListView.CHOICE_MODE_NONE);
64            //On affiche un layout qui ne permet pas de sélection
65            mListProg.setAdapter(new ArrayAdapter<String>(
                  SelectionMultipleActivity.this, android.R.layout.
                  simple_list_item_1, mLangages));
66
67            //On désactive le bouton
68            mSend.setEnabled(false);
69          }
70        });
```

```
71        }
72    }
```

Dans un tableau : `GridView`

On peut utiliser la classe `GridView` à partir du package `android.widget.GridView`.

Ce type de liste fonctionne presque comme le précédent ; cependant, il met les éléments dans une grille dont il détermine automatiquement le nombre d'éléments par ligne, comme le montre la figure 10.7.

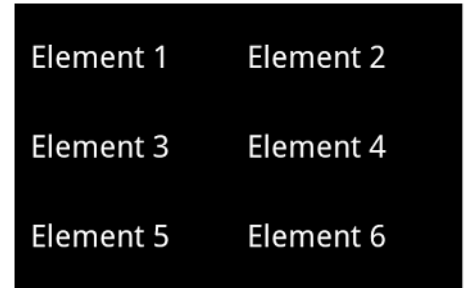


Figure 10.7 – Les éléments sont placés sur une grille

Il est cependant possible d'imposer ce nombre d'éléments par ligne à l'aide de `android:numColumns` en XML et `void setNumColumns (int column)` en Java.

Les listes défilantes : `Spinner`

La classe `Spinner` se trouve dans le package `android.widget.Spinner`.

Encore une fois, cet `AdapterView` ne réinvente pas l'eau chaude. Cependant, on utilisera deux vues. Une pour l'élément sélectionné qui est affiché, et une pour la liste d'éléments sélectionnables. La figure 10.8 montre ce qui arrive si on ne définit pas de mise en page pour la liste d'éléments.

La première vue affiche uniquement « Element 2 », l'élément actuellement sélectionné. La seconde vue affiche la liste de tous les éléments qu'il est possible de sélectionner.

Heureusement, on peut personnaliser l'affichage de la seconde vue, celle qui affiche une liste, avec la fonction `void setDropDownViewResource (int id)`. D'ailleurs, il existe déjà un layout par défaut pour cela. Voici un exemple :

```
1  import java.util.ArrayList;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.widget.ArrayAdapter;
6  import android.widget.Spinner;
7
```

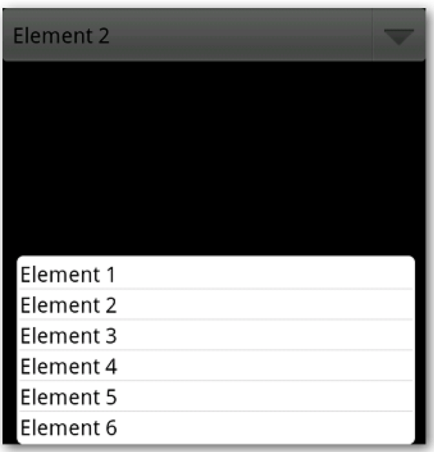


FIGURE 10.8 – Aucune mise en page pour la liste d'éléments n'a été définie

```
8   public class TutoListesActivity extends Activity {
9     private Spinner liste = null;
10
11    @Override
12    public void onCreate(Bundle savedInstanceState) {
13      super.onCreate(savedInstanceState);
14      setContentView(R.layout.main);
15
16      liste = (Spinner) findViewById(R.id.spinner1);
17      List<String> exemple = new ArrayList<String>();
18      exemple.add("Element 1");
19      exemple.add("Element 2");
20      exemple.add("Element 3");
21      exemple.add("Element 4");
22      exemple.add("Element 5");
23      exemple.add("Element 6");
24
25      ArrayAdapter<String> adapter = new ArrayAdapter<String>(
          this, android.R.layout.simple_spinner_item, exemple);
26      //Le layout par défaut est android.R.layout.
          simple_spinner_dropdown_item
27      adapter.setDropDownViewResource(android.R.layout.
          simple_spinner_dropdown_item);
28      liste.setAdapter(adapter);
29    }
30  }
```

Ce code donnera la figure 10.9.

FIGURE 10.9 – Un style a été défini

Plus complexe : les adaptateurs personnalisés

Imaginez que vous vouliez faire un répertoire téléphonique. Il consisterait donc en une liste, et chaque élément de la liste aurait une photo de l'utilisateur, son nom et prénom ainsi que son numéro de téléphone. Ainsi, on peut déduire que les items de notre liste auront un layout qui utilisera deux `TextView` et une `ImageView`. Je vous vois vous trémousser sur votre chaise en vous disant qu'on va utiliser un `SimpleAdapter` pour faire l'intermédiaire entre les données (complexes) et les vues, mais comme nous sommes des Zéros d'exception, nous allons plutôt créer notre propre adaptateur.

Je vous ai dit qu'un adaptateur implémentait l'interface `Adapter`, ce qui est vrai ; cependant, quand on crée notre propre adaptateur, il est plus sage de partir de `BaseAdapter` afin de nous simplifier l'existence.

Un adaptateur est le conteneur des informations d'une liste, au contraire de l'`AdapterView`, qui affiche les informations et régit ses interactions avec l'utilisateur. C'est donc dans l'adaptateur que se trouve la structure de données qui détermine comment sont rangées les données. Ainsi, dans notre adaptateur se trouvera une liste de contacts sous forme de `ArrayList`.

Dès qu'une classe hérite de `BaseAdapter`, il faut implémenter obligatoirement trois méthodes :

```java
import android.widget.BaseAdapter;

public class RepertoireAdapter extends BaseAdapter {
  /**
   * Récupérer un item de la liste en fonction de sa position
   * @param position - Position de l'item à récupérer
   * @return l'item récupéré
  */
  public Object getItem(int position) {
    // ...
  }

  /**
   * Récupérer l'identifiant d'un item de la liste en fonction
       de sa position (plutôt utilisé dans le cas d'une
   * base de données, mais on va l'utiliser aussi)
   * @param position - Position de l'item à récupérer
   * @return l'identifiant de l'item
  */
  public long getItemId(int position) {
    // ...
  }

  /**
   * Explication juste en dessous.
  */
  public View getView(int position, View convertView, ViewGroup
       parent) {
    //...
  }
}
```

La méthode `View getView(int position, View convertView, ViewGroup parent)` est la plus délicate à utiliser. En fait, cette méthode est appelée à chaque fois qu'un item est affiché à l'écran, comme à la figure 10.10.

En ce qui concerne les trois paramètres :

- `position` est la position de l'item dans la liste (et donc dans l'adaptateur).
- `parent` est le layout auquel rattacher la vue.
- Et `convertView` vaut `null`… ou pas, mais une meilleure explication s'impose.

`convertView` vaut `null` uniquement les premières fois qu'on affiche la liste. Dans notre exemple, `convertView` vaudra `null` aux sept premiers appels de `getView` (donc les sept premières créations de vues), c'est-à-dire pour tous les éléments affichés à l'écran au démarrage. Toutefois, dès qu'on fait défiler la liste jusqu'à afficher un élément qui n'était pas à l'écran à l'instant d'avant, `convertView` ne vaut plus `null`, mais plutôt la valeur de la vue qui vient de disparaître de l'écran. Ce qui se passe en interne, c'est que la vue qu'on n'affiche plus est recyclée, puisqu'on a plus besoin de la voir.

Il nous faut alors un moyen d'inflater une vue, mais sans l'associer à notre activité. Il

FIGURE 10.10 – Dans cet exemple, la méthode « getView » a été appelée sur les sept lignes visibles, mais pas sur les autres lignes de la liste

existe au moins trois méthodes pour cela :

- `LayoutInflater getSystemService (LAYOUT_INFLATER_SERVICE)` sur une activité.
- `LayoutInflater getLayoutInflater ()` sur une activité.
- `LayoutInflater LayoutInflater.from(Context contexte)`, sachant que `Activity` dérive de `Context`.

Puis vous pouvez inflater une vue à partir de ce `LayoutInflater` à l'aide de la méthode `View inflate (int id, ViewGroup root)`, avec `root` la racine à laquelle attacher la hiérarchie désérialisée. Si vous indiquez `null`, c'est la racine actuelle de la hiérarchie qui sera renvoyée, sinon la hiérarchie s'attachera à la racine indiquée.

Pourquoi ce mécanisme me demanderez-vous ? C'est encore une histoire d'optimisation. En effet, si vous avez un layout personnalisé pour votre liste, à chaque appel de `getView` vous allez peupler votre rangée avec le layout à inflater depuis son fichier XML :

```
1  LayoutInflater mInflater;
2  String[] mListe;
3
4  public View getView(int position, View convertView, ViewGroup
       parent) {
5    TextView vue = (TextView) mInflater.inflate(R.layout.ligne,
         null);
6
7    vue.setText(mListe[position]);
8
9    return vue;
10 }
```

Cependant, je vous l'ai déjà dit plein de fois, la désérialisation est un processus lent ! C'est pourquoi il *faut* utiliser `convertView` pour vérifier si cette vue n'est pas déjà peuplée et ainsi ne pas désérialiser à chaque construction d'une vue :

```
LayoutInflater mInflater;
String[] mListe;

public View getView(int position, View convertView, ViewGroup
    parent) {
  TextView vue = null;
  // Si la vue est recyclée, elle contient déjà le bon layout
  if(convertView != null)
    // On n'a plus qu'à la récupérer
    vue  = (TextView) convertView;
  else
    // Sinon, il faut en effet utiliser le LayoutInflater
    vue = mInflater.inflate(R.layout.ligne, null);

  vue.setText(mListe[position]);

  return vue;
}
```

En faisant cela, votre liste devient au moins deux fois plus fluide.

Quand vous utilisez votre propre adaptateur et que vous souhaitez pouvoir sélectionner des éléments dans votre liste, je vous conseille d'ignorer les solutions de sélection présentées dans le chapitre sur les listes (vous savez, `void setChoiceMode (int mode)`) et de développer votre propre méthode, vous aurez moins de soucis. Ici, j'ai ajouté un booléen dans chaque contact pour savoir s'il est sélectionné ou pas.

Amélioration : le *pattern* `ViewHolder`

Dans notre adaptateur, on remarque qu'on a optimisé le layout de chaque contact en ne l'inflatant que quand c'est nécessaire... mais on inflate quand même les trois vues qui ont le même layout ! C'est moins grave, parce que les vues inflatées par `findViewById` le sont plus rapidement, mais quand même. Il existe une alternative pour améliorer encore le rendu. Il faut utiliser une classe interne statique, qu'on appelle `ViewHolder` d'habitude. Cette classe devra contenir toutes les vues de notre layout :

```
static class ViewHolder {
  public TextView mNom;
  public TextView mNumero;
  public ImageView mPhoto;
}
```

Ensuite, la première fois qu'on inflate le layout, on récupère chaque vue pour les mettre dans le `ViewHolder`, puis on *insère* le `ViewHolder` dans le layout à l'aide de la méthode `void setTag (Object tag)`, qui peut être utilisée sur n'importe quel `View`. Cette technique permet d'insérer dans notre vue des objets afin de les récupérer plus tard

avec la méthode `Object getTag ()`. On récupérera le `ViewHolder` si le `convertView` n'est pas `null`, comme ça on n'aura inflaté les vues qu'une fois chacune.

```
public View getView(int r, View convertView, ViewGroup parent)
    {
  ViewHolder holder = null;
  // Si la vue n'est pas recyclée
  if(convertView == null) {
    // On récupère le layout
    convertView  = mInflater.inflate(R.layout.item, null);

    holder = new ViewHolder();
    // On place les widgets de notre layout dans le holder
    holder.mNom = (TextView) convertView.findViewById(R.id.nom)
        ;
    holder.mNumero = (TextView) convertView.findViewById(R.id.
        numero);
    holder.mPhoto = (ImageView) convertView.findViewById(R.id.
        photo);

    // puis on insère le holder en tant que tag dans le layout
    convertView.setTag(holder);
  } else {
    // Si on recycle la vue, on récupère son holder en tag
    holder = (ViewHolder)convertView.getTag();
  }

  // Dans tous les cas, on récupère le contact téléphonique
      concerné
  Contact c = (Contact)getItem(r);
  // Si cet élément existe vraiment...
  if(c != null) {
    // On place dans le holder les informations sur le contact
    holder.mNom.setText(c.getNom());
    holder.mNumero.setText(c.getNumero());
  }
  return convertView;
}
```

Les boîtes de dialogue

Une boîte de dialogue est une petite fenêtre qui passe au premier plan pour informer l'utilisateur ou lui demander ce qu'il souhaite faire. Par exemple, si je compte quitter mon navigateur internet alors que j'ai plusieurs onglets ouverts, une boîte de dialogue s'ouvrira pour me demander confirmation, comme le montre la figure 10.11.

On les utilise souvent pour annoncer des erreurs, donner une information ou indiquer un état d'avancement d'une tâche à l'aide d'une barre de progression par exemple.

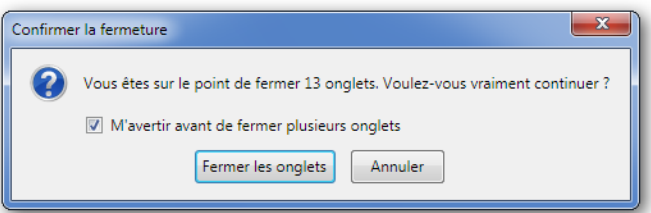


FIGURE 10.11 – Firefox demande confirmation avant de se fermer si plusieurs onglets sont ouverts

Généralités

Les boîtes de dialogue d'Android sont dites *modales*, c'est-à-dire qu'elles bloquent l'interaction avec l'activité sous-jacente. Dès qu'elles apparaissent, elles passent au premier plan en surbrillance devant notre activité et, comme on l'a vu dans le chapitre introduisant les activités, une activité qu'on ne voit plus que partiellement est suspendue.

Les boîtes de dialogue héritent de la classe `Dialog` et on les trouve dans le package `android.app.Dialog`.

On verra ici les boîtes de dialogue les plus communes, celles que vous utiliserez certainement un jour ou l'autre. Il en existe d'autres, et il vous est même possible de faire votre propre boîte de dialogue. Mais chaque chose en son temps.

Dans un souci d'optimisation, les développeurs d'Android ont envisagé un système très astucieux. En effet, on fera en sorte de ne pas avoir à créer de nouvelle boîte de dialogue à chaque occasion, mais plutôt de recycler les anciennes.

La classe `Activity` possède la méthode de *callback* `Dialog onCreateDialog (int id)`, qui sera appelée quand on instancie pour la première fois une boîte de dialogue. Elle prend en argument un entier qui sera l'identifiant de la boîte. Mais un exemple vaut mieux qu'un long discours :

```
private final static int IDENTIFIANT_BOITE_UN   = 0;
private final static int IDENTIFIANT_BOITE_DEUX   = 1;

@Override
public Dialog onCreateDialog(int identifiant) {
  Dialog box = null;
  //En fonction de l'identifiant de la boîte qu'on veut créer
  switch(identifiant) {
    case IDENTIFIANT_BOITE_UN :
      // On construit la première boîte de dialogue, que l'on
         insère dans « box »
      break;

```

```
13      case IDENTIFIANT_BOITE_DEUX :
14        // On construit la seconde boîte de dialogue, que l'on
              insère dans « box »
15        break;
16    }
17    return box;
18  }
```

Bien sûr, comme il s'agit d'une méthode de *callback*, on ne fait pas appel directement à `onCreateDialog`. Pour appeler une boîte de dialogue, on utilise la méthode `void showDialog (int id)`, qui se chargera d'appeler `onCreateDialog(id)` en lui passant le même identifiant.

Quand on utilise la méthode `showDialog` pour un certain identifiant la première fois, elle se charge d'appeler `onCreateDialog` comme nous l'avons vu, mais aussi la méthode `void onPrepareDialog (int id, Dialog dialog)`, avec le paramètre `id` qui est encore une fois l'identifiant de la boîte de dialogue, alors que le paramètre `dialog` est tout simplement la boîte de dialogue en elle-même. La seconde fois qu'on utilise `showDialog` avec un identifiant, `onCreateDialog` ne sera pas appelée (puisqu'on ne crée pas une boîte de dialogue deux fois), mais `onPrepareDialog` sera en revanche appelée.

Autrement dit, `onPrepareDialog` est appelée à chaque fois qu'on veut montrer la boîte de dialogue. Cette méthode est donc à redéfinir uniquement si on veut afficher un contenu différent pour la boîte de dialogue à chaque appel, mais, si le contenu est toujours le même à chaque appel, il suffit de définir le contenu dans `onCreateDialog`, qui n'est appelée qu'à la création. Et cela tombe bien, c'est le sujet du prochain exercice !

Application

Énoncé

L'activité consistera en un gros bouton. Cliquer sur ce bouton lancera une boîte de dialogue dont le texte indiquera le nombre de fois que la boîte a été lancée. Cependant une autre boîte de dialogue devient jalouse au bout de 5 appels et souhaite être sollicitée plus souvent, comme à la figure 10.12.

Instructions

Pour créer une boîte de dialogue, on va passer par le constructeur `Dialog (Context context)`. On pourra ensuite lui donner un texte à afficher à l'aide de la méthode `void setTitle (CharSequence text)`.

Ma solution

```
1  import android.app.Activity;
2  import android.app.Dialog;
```

Figure 10.12 – Après le cinquième clic

```
3  import android.os.Bundle;
4  import android.view.View;
5  import android.view.View.OnClickListener;
6  import android.widget.Button;
7
8  public class StringExampleActivity extends Activity {
9    private Button bouton;
10   //Variable globale, au-dessus de cette valeur c'est l'autre
       boîte de dialogue qui s'exprime
11   private final static int ENERVEMENT = 4;
12   private int compteur = 0;
13
14   private final static int ID_NORMAL_DIALOG = 0;
15   private final static int ID_ENERVEE_DIALOG = 1;
16
17   @Override
18   public void onCreate(Bundle savedInstanceState) {
19     super.onCreate(savedInstanceState);
20     setContentView(R.layout.activity_main);
21
22     bouton = (Button) findViewById(R.id.bouton);
23     bouton.setOnClickListener(boutonClik);
24   }
25
26   private OnClickListener boutonClik = new OnClickListener() {
27     @Override
28     public void onClick(View v) {
29       // Tant qu'on n'a pas invoqué la première boîte de
           dialogue 5 fois
30       if(compteur < ENERVEMENT) {
31         //on appelle la boîte normale
32         compteur ++;
33         showDialog(ID_NORMAL_DIALOG);
34       } else
35         showDialog(ID_ENERVEE_DIALOG);
36     }
37   };
38
39   /*
40    * Appelée qu'à la première création d'une boîte de dialogue
41    * Les fois suivantes, on se contente de récupérer la boîte
        de dialogue déjà créée...
42    * Sauf si la méthode « onPrepareDialog » modifie la boîte de
         dialogue.
43   */
44   @Override
45   public Dialog onCreateDialog (int id) {
46     Dialog box = null;
47     switch(id) {
48     // Quand on appelle avec l'identifiant de la boîte normale
```

```java
      case ID_NORMAL_DIALOG:
        box = new Dialog(this);
        box.setTitle("Je viens tout juste de naître.");
        break;

      // Quand on appelle avec l'identifiant de la boîte qui s'énerve
      case ID_ENERVEE_DIALOG:
        box = new Dialog(this);
        box.setTitle("ET MOI ALORS ???");
      }
      return box;
    }

    @Override
    public void onPrepareDialog (int id, Dialog box) {
      if(id == ID_NORMAL_DIALOG && compteur > 1)
        box.setTitle("On est au " + compteur + "ème lancement !");
       //On ne s'intéresse pas au cas où l'identifiant vaut 1, puisque cette boîte affiche le même texte à chaque lancement
    }
}
```

On va maintenant discuter des types de boîte de dialogue les plus courantes.

La boîte de dialogue de base

On sait déjà qu'une boîte de dialogue provient de la classe `Dialog`. Cependant, vous avez bien vu qu'on ne pouvait mettre qu'un titre de manière programmatique. Alors, de la même façon qu'on fait une interface graphique pour une activité, on peut créer un fichier XML pour définir la mise en page de notre boîte de dialogue.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <ImageView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/ic_launcher"/>
    <TextView
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
```

```
18          android:text="Je suis une jolie boîte de dialogue !"/>
19      </LinearLayout>
20  </LinearLayout>
```

On peut associer ce fichier XML à une boîte de dialogue comme on le fait pour une activité :

```
1  Dialog box = new Dialog(this);
2  box.setContentView(R.layout.dialog);
3  box.setTitle("Belle ! On dirait un mot inventé pour moiiii !");
```

Sur le résultat, visible à la figure 10.13, on voit bien à gauche l'icône de notre application et à droite le texte qu'on avait inséré. On voit aussi une des contraintes des boîtes de dialogue : le titre ne doit pas dépasser une certaine taille limite.

Figure 10.13 – Résultat en image

Cependant il est assez rare d'utiliser ce type de boîte de dialogue. Il y a des classes bien plus pratiques.

AlertDialog

On les utilise à partir du package `android.app.AlertDialog`. Il s'agit de la boîte de dialogue la plus polyvalente. Typiquement, elle peut afficher un titre, un texte et/ou une liste d'éléments.

La force d'une `AlertDialog` est qu'elle peut contenir jusqu'à trois boutons pour demander à l'utilisateur ce qu'il souhaite faire. Bien entendu, elle peut aussi n'en contenir aucun.

Pour construire une `AlertDialog`, on peut passer par le constructeur de la classe `AlertDialog` bien entendu, mais on préférera utiliser la classe `AlertDialog.Builder`, qui permet de simplifier énormément la construction. Ce constructeur prend en argument un `Context`.

Un objet de type `AlertDialog.Builder` connaît les méthodes suivantes :

- `AlertDialog.Builder setCancelable (boolean cancelable)` : si le paramètre `cancelable` vaut `true`, alors on pourra sortir de la boîte grâce au bouton retour de notre appareil.
- `AlertDialog.Builder setIcon (int ressource)` ou `AlertDialog.Builder setIcon (Drawable icon)` : le paramètre `icon` doit référencer une ressource de type `drawable` ou directement un objet de type `Drawable`. Permet d'ajouter une icône à la boîte de dialogue.

- `AlertDialog.Builder setMessage (int ressource)` ou `AlertDialog.Builder setMessage (String message)` : le paramètre `message` doit être une ressource de type `String` ou une `String`.
- `AlertDialog.Builder setTitle (int ressource)` ou `AlertDialog.Builder setTitle (String title)` : le paramètre `title` doit être une ressource de type `String` ou une `String`.
- `AlertDialog.Builder setView (View view)` ou `AlertDialog.Builder setView (int ressource)` : le paramètre `view` doit être une vue. Il s'agit de l'équivalent de `setContentView` pour un objet de type `Context`. Ne perdez pas de vue qu'il ne s'agit que d'une boîte de dialogue, elle est censée être de dimension réduite : il ne faut donc pas ajouter trop d'éléments à afficher.

On peut ensuite ajouter des boutons avec les méthodes suivantes :

- `AlertDialog.Builder setPositiveButton (text, DialogInterface.OnClick Listener listener)`, avec `text` qui doit être une ressource de type `String` ou une `String`, et `listener` qui définira que faire en cas de clic. Ce bouton se trouvera tout à gauche.
- `AlertDialog.Builder setNeutralButton (text, DialogInterface.OnClick Listener listener)`. Ce bouton se trouvera entre les deux autres boutons.
- `AlertDialog.Builder setNegativeButton (text, DialogInterface.OnClick Listener listener)`. Ce bouton se trouvera tout à droite.

Enfin, il est possible de mettre une liste d'éléments et de déterminer combien d'éléments on souhaite pouvoir choisir. Regardez le tableau suivant.

Les autres widgets

Date et heure

Il arrive assez fréquemment qu'on ait à demander à un utilisateur de préciser une date ou une heure, par exemple pour ajouter un rendez-vous dans un calendrier.

On va d'abord réviser comment on utilise les dates en Java. C'est simple, il suffit de récupérer un objet de type `Calendar` à l'aide de la méthode de classe `Calendar.getInstance()`. Cette méthode retourne un `Calendar` qui contiendra les informations sur la date et l'heure, au moment de la création de l'objet.

Si le `Calendar` a été créé le 23 janvier 2012 à 23h58, il vaudra toujours « 23 janvier 2012 à 23h58 », même dix jours plus tard. Il faut demander une nouvelle instance de `Calendar` à chaque fois que c'est nécessaire.

Il est ensuite possible de récupérer des informations à partir de la méthode `int get(int champ)` avec `champ` qui prend une valeur telle que :

- `Calendar.YEAR` pour l'année ;
- `Calendar.MONTH` pour le mois. Attention, le premier mois est de rang 0, alors que le premier jour du mois est bien de rang 1 !

Méthode	**Éléments sélectionnables**	**Usage**
`AlertDialog.Builder setItems (CharSequence[] items, DialogInterface.OnClickListener listener)`	Aucun	Le paramètre `items` correspond au tableau contenant les éléments à mettre dans la liste, alors que le paramètre `listener` décrit l'action à effectuer quand on clique sur un élément.
`AlertDialog.Builder setSingleChoiceItems (CharSequence[] items, int checkedItem, DialogInterface.OnClickListener listener)`	Un seul à la fois	Le paramètre `checkedItem` indique l'élément qui est sélectionné par défaut. Comme d'habitude, on commence par le rang 0 pour le premier élément. Pour ne sélectionner aucun élément, il suffit de mettre -1. Les éléments seront associés à un bouton radio afin que l'on ne puisse en sélectionner qu'un seul.
`AlertDialog.Builder setMultipleChoiceItems (CharSequence[] items, boolean[] checkedItems, DialogInterface.OnClickListener listener)`	Plusieurs	Le tableau `checkedItems` permet de déterminer les éléments qui sont sélectionnés par défaut. Les éléments seront associés à une case à cocher afin que l'on puisse en sélectionner plusieurs.

- `Calendar.DAY_OF_MONTH` pour le jour dans le mois ;
- `Calendar.HOUR_OF_DAY` pour l'heure ;
- `Calendar.MINUTE` pour les minutes ;
- `Calendar.SECOND` pour les secondes.

```
// Contient la date et l'heure au moment de sa création
Calendar calendrier = Calendar.getInstance();
// On peut ainsi lui récupérer des attributs
int mois = calendrier.get(Calendar.MONTH);
```

Insertion de dates

Pour insérer une date, on utilise le widget `DatePicker`. Ce widget possède en particulier deux attributs XML intéressants. Tout d'abord `android:minDate` pour indiquer quelle est la date la plus ancienne à laquelle peut remonter le calendrier, et son opposé `android:maxDate`.

En Java, on peut tout d'abord initialiser le widget à l'aide de la méthode `void init(int annee, int mois, int jour_dans_le_mois, DatePicker.OnDateChangedListener listener_en_cas_de_changement_de_date)`. Tous les attributs semblent assez évidents de prime abord à l'exception du dernier, peut-être. Il s'agit d'un `Listener` qui s'enclenche dès que la date du widget est modifiée, on l'utilise comme n'importe quel autre `Listener`. Remarquez cependant que ce paramètre peut très bien rester `null`.

Enfin vous pouvez à tout moment récupérer l'année avec `int getYear()`, le mois avec `int getMonth()` et le jour dans le mois avec `int getDayOfMonth()`.

Par exemple, j'ai créé un `DatePicker` en XML, qui commence en 2012 et se termine en 2032 :

```
<DatePicker
  android:id="@+id/datePicker"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_centerHorizontal="true"
  android:layout_centerVertical="true"
  android:startYear="2012"
  android:endYear="2032" />
```

Puis je l'ai récupéré en Java afin de changer la date de départ (par défaut, un `DatePicker` s'initialise à la date du jour) :

```
mDatePicker = (DatePicker) findViewById(R.id.datePicker);
mDatePicker.updateDate(mDatePicker.getYear(), 0, 1);
```

Ce qui donne le résultat visible à la figure 10.14.

Insertion d'horaires

Pour choisir un horaire, on utilise `TimePicker`, classe pas très contraignante puisqu'elle fonctionne comme `DatePicker` ! Alors qu'il n'est pas possible de définir un horaire

FIGURE 10.14 – Notre DatePicker

maximal et un horaire minimal cette fois, il est possible de définir l'heure avec `void setCurrentHour(Integer hour)`, de la récupérer avec `Integer getCurrentHour()`, et de définir les minutes avec `void setCurrentMinute(Integer minute)`, puis de les récupérer avec `Integer getCurrentMinute()`.

Comme nous utilisons en grande majorité le format 24 heures (rappelons que pour nos amis américains il n'existe pas de 13e heure, mais une deuxième 1re heure), notez qu'il est possible de l'activer à l'aide de la méthode `void setIs24HourView(Boolean mettre_en_format_24h)`.

Le `Listener` pour le changement d'horaire est cette fois géré par :

```
void setOnTimeChangedListener(TimePicker.OnTimeChangedListener
    onTimeChangedListener)
```

Cette fois encore, je définis le `TimePicker` en XML :

```
<TimePicker
  android:id="@+id/timePicker"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_centerHorizontal="true"
  android:layout_centerVertical="true" />
```

Puis je le récupère en Java pour rajouter un `Listener` qui se déclenche à chaque fois que l'utilisateur change l'heure :

```
mTimePicker = (TimePicker) findViewById(R.id.timePicker);
mTimePicker.setIs24HourView(true);
mTimePicker.setOnTimeChangedListener(new TimePicker.
    OnTimeChangedListener() {
  @Override
  public void onTimeChanged(TimePicker view, int hourOfDay, int
      minute) {
    Toast.makeText(MainActivity.this, "C'est vous qui voyez, il
        est donc " + String.valueOf(hourOfDay) + ":" + String.
        valueOf(minute), Toast.LENGTH_SHORT).show();
  }
});
```

Ce qui donne la figure 10.15.

Sachez enfin que vous pouvez utiliser de manière équivalente des boîtes de dialogue qui contiennent ces widgets. Ces boîtes s'appellent `DatePickerDialog` et `TimePickerDialog`.

FIGURE 10.15 – Changement de l'heure

Affichage de dates

Il n'existe malheureusement pas de widgets permettant d'afficher la date pour l'API 7, mais il existe deux façons d'écrire l'heure actuelle, soit avec une horloge analogique (comme sur une montre avec des aiguilles) qui s'appelle `AnalogClock`, soit avec une horloge numérique (comme sur une montre sans aiguilles) qui s'appelle `DigitalClock`, les deux visibles à la figure 10.16.

FIGURE 10.16 – À gauche une `AnalogClock` et à droite une `DigitalClock`

Afficher des images

Le widget de base pour afficher une image est `ImageView`. On peut lui fournir une image en XML à l'aide de l'attribut `android:src` dont la valeur est une ressource de type drawable. L'attribut `android:scaleType` permet de préciser comment vous souhaitez que l'image réagisse si elle doit être agrandie à un moment (si vous mettez `android:layout_width="fill_parent"` par exemple).

Le ratio d'une image est le rapport entre la hauteur et la largeur. Si le ratio d'une image est constant, alors en augmentant la hauteur, la largeur augmente dans une proportion identique et *vice versa*. Ainsi, avec un ratio constant, un carré reste toujours un carré, puisque quand on augmente la hauteur de x la longueur augmente aussi de x. Si le ratio n'est pas constant, en augmentant une des dimensions l'autre ne bouge pas. Ainsi, un carré devient un rectangle, car, si on étire la hauteur par exemple, la largeur n'augmente pas.

Les différentes valeurs qu'on peut attribuer sont visibles à la figure 10.17.

- `fitXY` : la première image est redimensionnée avec un ratio variable et elle prendra le plus de place possible. Cependant, elle restera dans le cadre de l'`ImageView`.

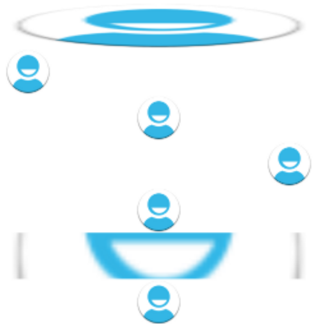


FIGURE 10.17 – L'image peut prendre différentes valeurs

- `fitStart` : la deuxième image est redimensionnée avec un ratio constant et elle prendra le plus de place possible. Cependant, elle restera dans le cadre de l'`ImageView`, puis ira se placer sur le côté en haut à gauche de l'`ImageView`.
- `fitCenter` : la troisième image est redimensionnée avec un ratio constant et elle prendra le plus de place possible. Cependant, elle restera dans le cadre de l'`ImageView`, puis ira se placer au centre de l'`ImageView`.
- `fitEnd` : la quatrième image est redimensionnée avec un ratio constant et elle prendra le plus de place possible. Cependant, elle restera dans le cadre de l'`ImageView`, puis ira se placer sur le côté bas à droite de l'`ImageView`.
- `center` : la cinquième image n'est pas redimensionnée. Elle ira se placer au centre de l'`ImageView`.
- `centerCrop` : la sixième image est redimensionnée avec un ratio constant et elle prendra le plus de place possible. Cependant, elle pourra dépasser le cadre de l'`ImageView`.
- `centerInside` : la dernière image est redimensionnée avec un ratio constant et elle prendra le plus de place possible. Cependant, elle restera dans le cadre de l'`ImageView`, puis ira se placer au centre de l'`ImageView`.

En Java, la méthode à employer dépend du typage de l'image. Par exemple, si l'image est décrite dans une ressource, on va passer par `void setImageResource(int id)`. On peut aussi insérer un objet `Drawable` avec la méthode `void setImageDrawable(Drawable image)` ou un fichier `Bitmap` avec `void setImageBitmap(Bitmap bm)`.

Enfin, il est possible de récupérer l'image avec la méthode `Drawable getDrawable()`.

C'est quoi la différence entre un `Drawable` et un `Bitmap` ?

Un `Bitmap` est une image de manière générale, pour être précis une image matricielle comme je les avais déjà décrites précédemment, c'est-à-dire une matrice (un tableau à deux dimensions) pour laquelle chaque case correspond à une couleur ; toutes les cases mises les unes à côté des autres forment une image. Un `Drawable` est un objet qui représente tout ce qui peut être dessiné. C'est-à-dire autant une image qu'un ensemble

d'images pour former une animation, qu'une forme (on peut définir un rectangle rouge dans un drawable), etc.

Notez enfin qu'il existe une classe appelée `ImageButton`, qui est un bouton normal, mais avec une image. `ImageButton` dérive de `ImageView`.

Pour des raisons d'accessibilité, il est conseillé de préciser le contenu d'un widget qui contient une image à l'aide de l'attribut XML `android:contentDescription`, afin que les malvoyants puissent avoir un aperçu sonore de ce que contient le widget. Cet attribut est disponible pour toutes les vues.

Autocomplétion

Quand on tape un mot, on risque toujours de faire une faute de frappe, ce qui est agaçant ! C'est pourquoi il existe une classe qui hérite de `EditText` et qui permet, en passant par un adaptateur, de suggérer à l'utilisateur le mot qu'il souhaite insérer.

Cette classe s'appelle `AutoCompleteTextView` et on va voir son utilisation dans un exemple dans lequel on va demander à l'utilisateur quelle est sa couleur préférée et l'aider à l'écrire plus facilement.

On peut modifier le nombre de lettres nécessaires avant de lancer l'autocomplétion à l'aide de l'attribut `android:completionThreshold` en XML et avec la méthode `void setThreshold(int threshold)` en Java.

Voici le fichier `main.xml` :

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res
       /android"
3    android:layout_width="fill_parent"
4    android:layout_height="fill_parent"
5    android:orientation="vertical" >
6
7    <AutoCompleteTextView
8      android:id="@+id/complete"
9      android:layout_width="fill_parent"
10     android:layout_height="wrap_content" />
11
12 </LinearLayout>
```

Ensuite, je déclare l'activité `AutoCompletionActivity` suivante :

```
1  import android.app.Activity;
2  import android.os.Bundle;
3  import android.widget.ArrayAdapter;
4  import android.widget.AutoCompleteTextView;
5
6  public class AutoCompletionActivity extends Activity {
```

```java
  private AutoCompleteTextView complete = null;

  // Notre liste de mots que connaîtra l'AutoCompleteTextView
  private static final String[] COULEUR = new String[] {
    "Bleu", "Vert", "Jaune", "Jaune canari", "Rouge", "Orange"
  };

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // On récupère l'AutoCompleteTextView déclaré dans notre layout
    complete = (AutoCompleteTextView) findViewById(R.id.complete);
    complete.setThreshold(2);

    // On associe un adaptateur à notre liste de couleurs...
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_dropdown_item_1line, COULEUR);
    // ... puis on indique que notre AutoCompleteTextView utilise cet adaptateur
    complete.setAdapter(adapter);
  }
}
```

Et voilà, dès que notre utilisateur a tapé deux lettres du nom d'une couleur, une liste défilante nous permet de sélectionner celle qui correspond à notre choix, comme le montre la figure 10.18.

Figure 10.18 – L'autocomplétion en marche

Vous remarquerez que cette autocomplétion se fait sur la ligne entière, c'est-à-dire que si vous tapez « Jaune rouge », l'application pensera que vous cherchez une couleur qui s'appelle « Jaune rouge », alors que bien entendu vous vouliez le mot « jaune » puis le mot « rouge ». Pour faire en sorte qu'une autocomplétion soit répartie entre plusieurs constituants d'une même chaîne de caractères, il faut utiliser la classe `MultiAutoCompleteTextView`. Toutefois, il faut préciser quel caractère sera utilisé pour séparer deux éléments avec la méthode `void setTokenizer(MultiAutoCompleteTextView.Tokenizer t)`. Par défaut, on peut par exemple utiliser un `MultiAutoCompleteTextView.CommaTo`

`kenizer`, qui différencie les éléments par des virgules (ce qui signifie qu'à chaque fois que vous écrirez une virgule, le `MultiAutoCompleteTextView` vous proposera une nouvelle suggestion).

En résumé

- L'affichage d'une liste s'organise de la manière suivante : on donne une liste de données à un adaptateur (`Adapter`) qui sera attaché à une liste (`AdapterView`). L'adaptateur se chargera alors de construire les différentes vues à afficher ainsi que de gérer leurs cycles de vie.
- Les adaptateurs peuvent être attachés à plusieurs types d'`AdapterView` :
  - `ListView` pour lister les vues les unes en dessous des autres.
  - `GridView` pour afficher les vues dans une grille.
  - `Spinner` pour afficher une liste de vues défilante.
- Lorsque vous désirez afficher des vues plus complexes, vous devez créer votre propre adaptateur qui étend la super classe `BaseAdapter` et redéfinir les méthodes en fonction de votre liste de données.
- Les boîtes de dialogue peuvent être confectionnées de 2 manières différentes : par la classe `Dialog` ou par un builder pour construire une `AlertDialog`, ce qui est plus puissant.
- Pour insérer un widget capable de gérer l'autocomplétion, on utilisera le widget `AutoCompleteTextView`.

Chapitre 11

Gestion des menus de l'application

Difficulté :

Il n'y a pas si longtemps, tous les terminaux sous Android possédaient un bouton physique pour afficher le menu. C'est devenu plus rare depuis que les constructeurs essaient au maximum de dématérialiser les boutons. Mais depuis Android 2.3, il existe un bouton directement dans l'interface de l'OS qui permet d'ouvrir un menu. Un menu est un endroit privilégié pour placer certaines fonctions tout en économisant notre précieux espace dans la fenêtre. Il existe deux sortes de menu dans Android : **le menu d'options**, qu'on peut ouvrir avec le bouton `Menu` sur le téléphone. Si le téléphone est dépourvu de cette touche, Android fournit un bouton dans son interface graphique pour y accéder ; et **les menus contextuels**, vous savez, ceux qui s'ouvrent à l'aide d'un clic droit sous Windows et Linux ? Eh bien, dans Android ils se déroulent dès lors qu'on effectue un long clic sur un élément de l'interface graphique. Ces menus peuvent bien sûr contenir des sous-menus, contenant des sous-menus, etc. On va encore devoir manipuler des fichiers XML mais vous êtes devenus des experts maintenant, non ?

Menu d'options

Créer un menu

Chaque activité est capable d'avoir son menu propre. On peut définir un menu de manière programmatique en Java, mais la meilleure façon de procéder est en XML. Tout d'abord, la racine de ce menu est de type `<menu>` (vous arriverez à retenir ?), et on ne peut pas vraiment le personnaliser avec des attributs, ce qui donne la majorité du temps :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <menu xmlns:android="http://schemas.android.com/apk/res/android
     ">
3   <!-- Code -->
4 </menu>
```

Ce menu doit être peuplé avec des éléments, et c'est sur ces éléments que cliquera l'utilisateur pour activer ou désactiver une fonctionnalité. Ces éléments s'appellent en XML des `<item>` et peuvent être personnalisés à l'aide de plusieurs attributs :

- `android:id`, que vous connaissez déjà. Il permet d'identifier de manière unique un `<item>`. Autant d'habitude cet attribut est facultatif, autant cette fois il est vraiment indispensable, vous verrez pourquoi dans cinq minutes.
- `android:icon`, pour agrémenter votre `<item>` d'une icône.
- `android:title`, qui sera son texte dans le menu.
- Enfin, on peut désactiver par défaut un `<item>` avec l'attribut `android:enabled="false"`.

Vous pouvez récupérer gratuitement et légalement les icônes fournies avec Android. Par rapport à l'endroit où se situe le SDK, vous les trouverez dans `.\platforms\android-x\data\res\` avec `x` le niveau de l'API que vous utilisez. Il est plutôt recommandé d'importer ces images en tant que drawables dans notre application, plutôt que de faire référence à l'icône utilisée actuellement par Android, car elle pourrait ne pas exister ou être différente en fonction de la version d'Android qu'exploite l'utilisateur. Si vous souhaitez faire vos propres icônes, sachez que la taille maximale recommandée est de 72 pixels pour les hautes résolutions, 48 pixels pour les moyennes résolutions et 38 pixels pour les basses résolutions.

Le problème est que l'espace consacré à un menu est assez réduit, comme toujours sur un périphérique portable, remarquez. Afin de gagner un peu de place, il est possible d'avoir un `<item>` qui ouvre un sous-menu, et ce sous-menu sera à traiter comme tout autre menu. On lui mettra donc des items aussi. En d'autres termes, la syntaxe est celle-ci :

```
1 <item>
2   <menu>
3     <item />
4     <!-- d'autres items-->
```

```
5       <item />
6     </menu>
7   </item>
```

Le sous-menu s'ouvrira dans une nouvelle fenêtre, et le titre de cette fenêtre se trouve dans l'attribut `android:title`. Si vous souhaitez mettre un titre plutôt long dans cette fenêtre et conserver un nom court dans le menu, utilisez l'attribut `android:titleCondensed`, qui permet d'indiquer un titre à utiliser si le titre dans `android:title` est trop long. Ces `<item>` qui se trouvent dans un sous-menu peuvent être modulés avec d'autres attributs, comme `android:checkable` auquel vous pouvez mettre `true` si vous souhaitez que l'élément puisse être coché, comme une `CheckBox`. De plus, si vous souhaitez qu'il soit coché par défaut, vous pouvez mettre `android:checked` à `true`. Je réalise que ce n'est pas très clair, aussi vous proposé-je de regarder les figures 11.1 et 11.2 : la première utilise `android:titleCondensed="Item 1"`, la deuxième `android:title="Item 1 mais avec un titre plus long quand même"`.

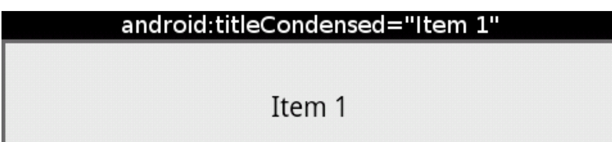


FIGURE 11.1 – Le titre est condensé

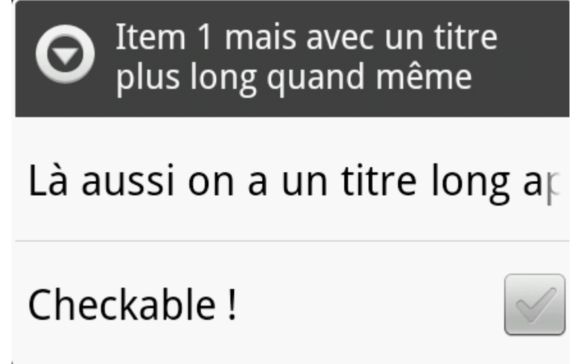


FIGURE 11.2 – Le titre est plus long

Enfin, il peut arriver que plusieurs éléments se ressemblent beaucoup ou fonctionnent ensemble, c'est pourquoi il est possible de les regrouper avec `<group>`. Si on veut que tous les éléments du groupe soient des `CheckBox`, on peut mettre au groupe l'attribut `android:checkableBehavior="all"`, ou, si on veut qu'ils soient tous des `RadioButton`, on peut mettre l'attribut `android:checkableBehavior="single"`.

Voici un exemple de menu qu'il vous est possible de créer avec cette méthode :

```
1 <?xml version="1.0" encoding="utf-8"?>
```

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
  <item android:id="@+id/item1" android:title="Item 1"></item>
  <item android:id="@+id/item2" android:titleCondensed="Item 2" android:title="Item 2 mais avec un nom assez long quand même">
    <menu>
      <item android:id="@+id/item3" android:title="Item 2.1" android:checkable="true"/>
      <item android:id="@+id/item4" android:title="Item 2.2"/>
    </menu>
  </item>
  <item android:id="@+id/item5" android:title="Item 3" android:checkable="true"/>
  <item android:id="@+id/item6" android:title="Item 4">
    <group android:id="@+id/group1" android:checkableBehavior="all">
      <item android:id="@+id/item7" android:title="Item 4.1"></item>
      <item android:id="@+id/item8" android:title="Item 4.2"></item>
    </group>
  </item>
  <group android:id="@+id/group2" android:enabled="false">
    <item android:id="@+id/item9" android:title="Item 5.1"></item>
    <item android:id="@+id/item10" android:title="Item 5.2"></item>
  </group>
</menu>
```

Comme pour un layout, il va falloir dire à Android qu'il doit parcourir le fichier XML pour construire le menu. Pour cela, c'est très simple, on va surcharger la méthode `boolean onCreateOptionsMenu (Menu menu)` d'une activité. Cette méthode est lancée au moment de la première pression du bouton qui fait émerger le menu. Cependant, comme avec les boîtes de dialogue, si vous souhaitez que le menu évolue à chaque pression du bouton, alors il vous faudra surcharger la méthode `boolean onPrepareOptionsMenu (Menu menu)`.

Pour parcourir le XML, on va l'inflater, eh oui ! encore une fois ! Encore un petit rappel de ce qu'est inflater ? *To inflate*, c'est désérialiser en français, et dans notre cas c'est transformer un objet qui n'est décrit qu'en XML en véritable objet qu'on peut manipuler. Voici le code type dès qu'on a constitué un menu en XML :

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  super.onCreateOptionsMenu(menu);
  MenuInflater inflater = getMenuInflater();
  //R.menu.menu est l'id de notre menu
  inflater.inflate(R.menu.menu, menu);
  return true;
```

```
8 }
```

Si vous testez ce code, vous remarquerez tout d'abord que, contrairement au premier exemple, il n'y a pas assez de place pour contenir tous les items, c'est pourquoi le 6e item se transforme en un bouton pour afficher les éléments cachés, comme à la figure 11.3.

FIGURE 11.3 – Un bouton permet d'accéder aux autres items

Ensuite vous remarquerez que les items 4.1 et 4.2 sont décrits comme `Checkable`, mais ne possèdent pas de case à cocher. C'est parce que les seuls `<item>` que l'on puisse cocher sont ceux qui se trouvent dans un sous-menu.

Les `<item>` 5.1 et 5.2 sont désactivés par défaut, mais vous pouvez les réactiver de manière programmatique à l'aide de la fonction `MenuItem setEnabled (boolean activer)` (le `MenuItem` retourné est celui sur lequel l'opération a été effectuée, de façon à pouvoir cumuler les *setters*).

Un *setter* est une méthode qui permet de modifier un des attributs d'un objet. Un *getter* est une méthode qui permet, elle, de récupérer un attribut d'un objet.

Vous pouvez aussi si vous le désirez construire un menu de manière programmatique avec la méthode suivante qui s'utilise sur un `Menu` :

```
1 MenuItem add (int groupId, int objectId, int ordre,
      CharSequence titre)
```

Où :

- `groupId` permet d'indiquer si l'objet appartient à un groupe. Si ce n'est pas le cas, vous pouvez mettre `Menu.NONE`.
- `objectId` est l'identifiant unique de l'objet, vous pouvez aussi mettre `Menu.NONE`, mais je ne le recommande pas.
- `ordre` permet de définir l'ordre dans lequel vous souhaitez le voir apparaître dans le menu. Par défaut, l'ordre respecté est celui du fichier XML ou de l'ajout avec la méthode `add`, mais avec cette méthode vous pouvez bousculer l'ordre établi pour indiquer celui que vous préférez. Encore une fois, vous pouvez mettre `Menu.NONE`.
- `titre` est le titre de l'item.

De manière identique et avec les mêmes paramètres, vous pouvez construire un sous-menu avec la méthode suivante :

```
MenuItem addSubMenu (int groupId, int objectId, int ordre,
    CharSequence titre)
```

Et c'est indispensable de passer le menu à la superclasse comme on le fait ?

La réponse courte est non, la réponse longue est non, mais faites-le quand même. En passant le menu à l'implémentation par défaut, Android va peupler le menu avec des items systèmes standards. Alors, en tant que débutants, vous ne verrez pas la différence, mais si vous devenez des utilisateurs avancés, un oubli pourrait bien vous encombrer.

Réagir aux clics

Vous vous rappelez quand je vous avais dit qu'il était inconcevable d'avoir un `<item>` sans identifiant ? C'était parce que l'identifiant d'un `<item>` permet de déterminer comment il réagit aux clics au sein de la méthode `boolean onOptionsItemSelected (MenuItem item)`.

Dans l'exemple précédent, si on veut que cliquer sur le premier item active les deux items inactifs, on pourrait utiliser le code suivant dans notre activité :

```
private Menu m = null;

@Override
public boolean onCreateOptionsMenu(Menu menu)
{
  MenuInflater inflater = getMenuInflater();
  inflater.inflate(R.menu.menu, menu);
  m = menu;
  return true;
}

@Override
public boolean onOptionsItemSelected (MenuItem item)
{
  switch(item.getItemId())
  {
    case R.id.item1:
      //Dans le Menu "m", on active tous les items dans le
          groupe d'identifiant "R.id.group2"
      m.setGroupEnabled(R.id.group2, true);
      return true;
  }
  return super.onOptionsItemSelected(item);
}
```

Et ils veulent dire quoi les `true` et `false` en retour ?

On retourne `true` si on a bien géré l'item, `false` si on a rien géré. D'ailleurs, si on passe l'item à `super.onOptionsItemSelected(item)`, alors la méthode retournera `false` puisqu'elle ne sait pas gérer l'item. En revanche, je vous conseille de toujours retourner `super.onOptionsItemSelected(item)` quand vous êtes dans une classe qui ne dérive pas directement de `Activity`, puisqu'il se peut que vous gériez l'item dans une superclasse de votre classe actuelle.

Dans `boolean onCreateOptionsMenu(menu)`, on retourne toujours `true` puisqu'on gère dans tous les cas la création du menu.

Google nous fournit une astuce de qualité sur son site : souvent, une application partage plus ou moins le(s) même(s) menu(s) entre tous ses écrans (et donc toutes ses activités), c'est pourquoi il est conseillé d'avoir une activité de base qui ne gère que les évènements liés au(x) menu(s) (création dans `onCreateOptionsMenu`, mise à jour dans `onPrepareOptionsMenu` et gestion des évènements dans `onOptionsItemSelected`), puis d'en faire dériver toutes les activités de notre application qui utilisent les mêmes menus.

Menu contextuel

Le menu contextuel est très différent du menu d'options, puisqu'il n'apparaît pas quand on appuie sur le bouton d'options, mais plutôt quand on clique sur n'importe quel élément ! Sur Windows, c'est le menu qui apparaît quand vous faites un clic droit.

Alors, on ne veut peut-être pas que tous les objets aient un menu contextuel, c'est pourquoi il faut déclarer quels widgets en possèdent, et cela se fait dans la méthode de la classe `Activity void registerForContextMenu (View vue)`. Désormais, dès que l'utilisateur fera un clic long sur cette vue, un menu contextuel s'ouvrira... enfin, si vous le définissez !

Ce menu se définit dans la méthode suivante :

```
1 void onCreateContextMenu (ContextMenu menu, View vue,
      ContextMenu.ContextMenuInfo menuInfo)
```

Où `menu` est le menu à construire, `vue` la vue sur laquelle le menu a été appelé et `menuInfo` indique sur quel élément d'un adaptateur a été appelé le menu, si on se trouve dans une liste par exemple. Cependant, il n'existe pas de méthode du type `OnPrepare` cette fois-ci, par conséquent le menu est détruit puis reconstruit à chaque appel. C'est pourquoi il n'est pas conseillé de conserver le menu dans un paramètre comme nous l'avions fait pour le menu d'options. Voici un exemple de construction de menu contextuel de manière programmatique :

```
//Notez qu'on utilise Menu.FIRST pour indiquer le premier élément d'un menu
private int final static MENU_DESACTIVER = Menu.FIRST;
private int final static MENU_RETOUR = Menu.FIRST + 1;

@Override
public void onCreateContextMenu(ContextMenu menu, View vue, ContextMenuInfo menuInfo) {
  super.onCreateContextMenu(menu, vue, menuInfo);
  menu.add(Menu.NONE, MENU_DESACTIVER, Menu.NONE, "Supprimer cet élément");
  menu.add(Menu.NONE, MENU_RETOUR, Menu.NONE, "Retour");
}
```

On remarque deux choses. Tout d'abord pour écrire des identifiants facilement, la classe `Menu` possède une constante `Menu.FIRST` qui permet de désigner le premier élément, puis le deuxième en incrémentant, etc. Ensuite, on passe les paramètres à la superclasse. En fait, cette manœuvre a pour but bien précis de permettre de récupérer le `ContextMenuInfo` dans la méthode qui gère l'évènementiel des menus contextuels, la méthode `boolean onContextItemSelected (MenuItem item)`. Ce faisant, vous pourrez récupérer des informations sur la vue qui a appelé le menu avec la méthode `ContextMenu.ContextMenuInfo getMenuInfo ()` de la classe `MenuItem`. Un exemple d'implémentation pour notre exemple pourrait être :

```
@Override
public boolean onContextItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case MENU_DESACTIVER:
      item.getMenuInfo().targetView.setEnabled(false);

    case MENU_RETOUR:
      return true;
  }
  return super.onContextItemSelected(item);
}
```

Voilà ! Le `ContextMenuInfo` a permis de récupérer la vue grâce à son attribut `targetView`. Il possède aussi un attribut `id` pour récupérer l'identifiant de l'item (dans l'adaptateur) concerné ainsi qu'un attribut `position` pour récupérer sa position au sein de la liste.

Maintenant que vous maîtrisez les menus, oubliez tout

Titre racoleur, j'en conviens, mais qui révèle une vérité qu'il vous faut considérer : le bouton `Menu` est amené à disparaître. De manière générale, les utilisateurs n'utilisent pas ce bouton, il n'est pas assez visuel pour eux, ce qui fait qu'ils n'y pensent pas ou ignorent son existence. C'est assez grave, oui. Je vous apprends à l'utiliser parce que c'est quand même sacrément pratique et puissant, mais c'est à vous de faire la

démarche d'apprendre à l'utilisateur comment utiliser correctement ce bouton, avec un `Toast` par exemple.

Il existe des solutions qui permettent de se passer de ce menu. Android a introduit dans son API 11 (Android 3.0) l'**ActionBar**, qui est une barre de titre étendue sur laquelle il est possible d'ajouter des widgets de façon à disposer d'options constamment visibles. Cette initiative a été efficace puisque le taux d'utilisation de l'ActionBar est bien supérieur à celui du bouton `Menu`.

Cependant, pour notre cours, cette ActionBar n'est pas disponible puisque nous utilisons l'API 7, et qu'il n'est pas question d'utiliser l'API 11 rien que pour ça — vous ne toucheriez plus que 5 % des utilisateurs de l'Android Market, au lieu des 98 % actuels... Il existe des solutions alternatives que je vous invite à découvrir par vous-mêmes.

▷ Solution officielle
Code web : 295997

▷ Solution puissante
Code web : 115602

Histoire de retourner le couteau dans la plaie, sachez que les menus contextuels sont rarement utilisés, puisqu'en général l'utilisateur ignore leur présence ou ne sait pas comment les utiliser (faire un appui long, c'est compliqué pour l'utilisateur, vraiment >_<). Encore une fois, vous pouvez enseigner à vos utilisateurs comment les utiliser, ou bien ajouter une alternative plus visuelle pour ouvrir un menu sur un objet. Ça tombe super bien, c'est le sujet du prochain chapitre.

En résumé

- La création d'un menu se fait en XML pour tout ce qui est statique et en Java pour tout ce qui est dynamique.
- La déclaration d'un menu se fait obligatoirement avec un élément `menu` à la racine du fichier et contiendra des éléments `item`.
- Un menu d'options s'affiche lorsque l'utilisateur clique sur le bouton de menu de son appareil. Il ne sera affiché que si le fichier XML représentant votre menu est désérialisé dans la méthode `boolean onCreateOptionsMenu(Menu menu)` et que vous avez donné des actions à chaque `item` dans la méthode `boolean onOptionsItemSelected(MenuItem item)`.
- Un menu contextuel s'affiche lorsque vous appuyez longtemps sur un élément de votre interface. Pour ce faire, vous devez construire votre menu à partir de la méthode `void onCreateContextMenu(ContextMenu menu, View vue, ContextMenu.ContextMenuInfo menuInfo)` et récupérer la vue qui a fait appel à votre menu contextuel à partir de la méthode `boolean onContextItemSelected(MenuItem item)`.
- Sachez tout de même que le bouton menu physique tend à disparaitre de plus en plus pour un menu tactile.

Chapitre 12

Création de vues personnalisées

Difficulté :

Vous savez désormais l'essentiel pour développer de belles interfaces graphiques fonctionnelles ! Mais il vous manque encore l'outil ultime pour donner vie à tous vos fantasmes les plus extravagants : être capables de produire vos propres vues et ainsi contrôler leur aspect, leur taille, leurs réactions et leur fonction.

On différencie typiquement trois types de vues personnalisées :

– Si vous souhaitez faire une vue qui ressemble à une vue standard que vous connaissez déjà, vous pourrez partir de cette vue et modifier son fonctionnement.
– Autrement, vous pourriez exploiter des vues qui existent déjà et les réunir de façon à produire une nouvelle vue qui exploite le potentiel de ses vues constitutives.
– Ou bien encore, si vous souhaitez forger une vue qui n'existe pas du tout, vous pouvez la fabriquer en partant de zéro. C'est la solution la plus radicale, la plus exigeante, mais aussi la plus puissante.

Règles avancées concernant les vues

Cette section est très théorique, je vous conseille de la lire une fois, de la comprendre, puis de continuer dans le cours et d'y revenir au besoin. Et vous en aurez sûrement besoin, d'ailleurs.

Si vous deviez instancier un objet de type `View` et l'afficher dans une interface graphique, vous vous retrouveriez devant un carré blanc qui mesure 100 pixels de côté. Pas très glamour, j'en conviens. C'est pourquoi, quand on crée une vue, on doit jouer sur au moins deux tableaux : les dimensions de la vue, et son dessin.

Dimensions et placement d'une vue

Les dimensions d'une vue sont deux entiers qui représentent la taille que prend la vue sur les deux axes de l'écran : la largeur et la hauteur. Toute vue ne possède pas qu'une paire de dimensions, mais bien deux : celles que vous connaissez et qui vous sembleront logiques sont les dimensions réelles occupées par la vue sur le terrain. Cependant, avant que les coordonnées réelles soient déterminées, une vue passe par une phase de calcul où elle s'efforce de déterminer les dimensions qu'elle souhaiterait occuper, sans garantie qu'il s'agira de ses dimensions finales.

Par exemple, si vous dites que vous disposez d'une vue qui occupe toute seule son layout parent et que vous lui donnez l'instruction `FILL_PARENT`, alors les dimensions réelles seront identiques aux dimensions demandées puisque la vue peut occuper tout le parent. En revanche, s'il y a plusieurs vues qui utilisent `FILL_PARENT` pour un même layout, alors les dimensions réelles seront différentes de celles demandées, puisque le layout fera en sorte de répartir les dimensions entre chacun de ses enfants.

Un véritable arbre généalogique

Vous le savez déjà, on peut construire une interface graphique dans le code ou en XML. Je vais vous demander de réfléchir en XML ici, pour simplifier le raisonnement. Un fichier XML contient toujours un premier élément unique qui n'a pas de **frère**, cet élément s'appelle la **racine**, et dans le contexte du développement d'interfaces graphiques pour Android cette racine sera très souvent un **layout**. Dans le code suivant, la racine est un `RelativeLayout`.

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/
       res/android"
2    xmlns:tools="http://schemas.android.com/tools"
3    android:layout_width="fill_parent"
4    android:layout_height="fill_parent" >
5
6    <Button
7      android:id="@+id/passerelle"
8      android:layout_width="wrap_content"
```

```
9        android:layout_height="wrap_content"
10       android:layout_centerHorizontal="true"
11       android:layout_centerVertical="true" />
12
13   </RelativeLayout>
```

Ce layout peut avoir des enfants, qui seront des widgets ou d'autres layouts. Dans l'éventualité où un enfant serait un layout, alors il peut aussi avoir des enfants à son tour. On peut donc affirmer que, comme pour une famille, il est possible de construire un véritable arbre généalogique qui commence par la racine et s'étend sur plusieurs générations, comme à la figure 12.1.

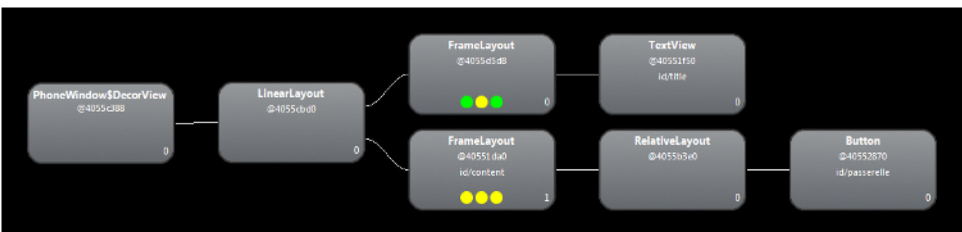


Figure 12.1 – Dans cet exemple, on peut voir que toutes les vues sont des enfants ou petits-enfants du « LinearLayout » et que les autres layouts peuvent aussi avoir des enfants, tandis que les widgets n'ont pas d'enfant

Ce que vous ne savez pas, c'est que la racine de notre application n'est pas la racine de la hiérarchie des vues et qu'elle sera forcément l'enfant d'une autre vue qu'a créée Android dans notre dos et à laquelle nous n'avons pas accès. *Ainsi, chaque vue que nous utiliserons sera directement l'enfant d'un layout.*

Le placement

Le placement — qui se dit aussi *layout* en anglais (à ne pas confondre avec les layouts qui sont des vues qui contiennent des vues, et le layout correspondant à la mise en page de l'interface graphique) — est l'opération qui consiste à placer les vues dans l'interface graphique. Ce processus s'effectue en deux étapes qui s'exécuteront dans l'ordre chronologique. Tout d'abord et en partant de la racine, chaque layout va donner à ses enfants des instructions quant à la taille qu'ils devraient prendre. Cette étape se fait dans la méthode `void measure(int widthMeasureSpec, int heightMeasureSpec)`, ne vous préoccupez pas trop de cette méthode, on ne l'implémentera pas. Puis vient la seconde étape, qui débute elle aussi par la racine et où chaque layout transmettra à ses enfants leurs dimensions finales en fonction des mesures déterminées dans l'étape précédente. Cette manœuvre se déroule durant l'exécution de `void layout(int bord_gauche, int plafond, int bord_droit, int plancher)`, mais on ne l'implémentera pas non plus.

Si à un quelconque moment vous rencontrez une vue dont les limites ne lui correspondent plus, vous pouvez essayer de la faire se redimensionner en lançant sa méthode `void requestLayout ()` — ainsi le calcul se fera sur la vue et sur toutes les autres vues qui pourraient être influencées par les nouvelles dimensions de la vue.

Récupérer les dimensions

De manière à récupérer les instructions de dimensions, vous pouvez utiliser `int getMeasuredWidth ()` pour la largeur et `int getMeasuredHeight ()` pour la hauteur, cependant *uniquement* après qu'un appel à `measure(int, int)` a été effectué, sinon ces valeurs n'ont pas encore été attribuées. Enfin, vous pouvez les attribuer vous-mêmes avec la méthode `void setMeasuredDimension (int measuredWidth, int measuredHeight)`.

Ces instructions doivent vous sembler encore mystérieuses puisque vous ne devez pas du tout savoir quoi insérer. En fait, ces entiers sont... un code. En effet, vous pouvez à partir de ce code déterminer un mode de façonnage et une taille.

- La taille se récupère avec la fonction statique `int MeasureSpec.getSize (int measureSpec)`.
- Le mode se récupère avec la fonction statique `int MeasureSpec.getMode (int measureSpec)`. S'il vaut `MeasureSpec.UNSPECIFIED`, alors le parent n'a pas donné d'instruction particulière sur la taille à prendre. S'il vaut `MeasureSpec.EXACTLY`, alors la taille donnée est la taille exacte à adopter. S'il vaut `MeasureSpec.AT_MOST`, alors la taille donnée est la taille maximale que peut avoir la vue.

Par exemple pour obtenir le code qui permet d'avoir un cube qui fait 10 pixels au plus, on peut faire :

```
1 int taille = MeasureSpec.makeMeasureSpec(10, MeasureSpec.
    AT_MOST);
2 setMeasuredDimension(taille, taille);
```

De plus, il est possible de connaître la largeur finale d'une vue avec `int getWidth ()` et sa hauteur finale avec `int getHeight ()`.

Enfin, on peut récupérer la position d'une vue par rapport à son parent à l'aide des méthodes `int getTop ()` (position du haut de cette vue par rapport à son parent), `int getBottom ()` (en bas), `int getLeft ()` (à gauche) et `int getRight ()` (à droite). C'est pourquoi vous pouvez demander très simplement à n'importe quelle vue ses dimensions en faisant :

```
1 vue.getWidth();
2 vue.getLeft();
```

Le dessin

C'est seulement une fois le placement effectué qu'on peut dessiner notre vue (vous imaginez bien qu'avant Android ne saura pas où dessiner). Le dessin s'effectue dans

la méthode `void draw (Canvas canvas)`, qui ne sera pas non plus à implémenter. Le `Canvas` passé en paramètre est la surface sur laquelle le dessin sera tracé.

Obsolescence régionale

Tout d'abord, une vue ne décide pas d'elle-même quand elle doit se dessiner, elle en reçoit l'ordre, soit par le `Context`, soit par le programmeur. Par exemple, le contexte indique à la racine qu'elle doit se dessiner au lancement de l'application. Dès qu'une vue reçoit cet ordre, sa première tâche sera de déterminer ce qui doit être dessiné parmi les éléments qui composent la vue.

Si la vue comporte un nouveau composant ou qu'un de ses composants vient d'être modifié, alors la vue déclare que ces éléments sont dans une zone qu'il faut redessiner, puisque leur état actuel ne correspond plus à l'ancien dessin de la vue. La surface à redessiner consiste en un rectangle, le plus petit possible, qui inclut tous les éléments à redessiner, mais pas plus. Cette surface s'appelle la **dirty region**. L'action de délimiter la dirty region s'appelle l'**invalidation** (c'est pourquoi on appelle aussi la dirty region la **région d'invalidation**) et on peut la provoquer avec les méthodes `void invalidate (Rect dirty)` (où `dirty` est le rectangle qui délimite la dirty region) ou `void invalidate (int gauche, int haut, int droite, int bas)` avec `gauche` la limite gauche du rectangle, `haut` le plafond du rectangle, etc., les coordonnées étant exprimées par rapport à la vue. Si vous souhaitez que toute la vue se redessine, utilisez la méthode `void invalidate ()`, qui est juste un alias utile de `void invalidate (0, 0, largeur_de_la_vue, hauteur_de_la_vue)`. Enfin, évitez de trop le faire puisque dessiner est un processus exigeant. :-°

Par exemple, quand on passe d'une `TextView` vide à une `TextView` avec du texte, la seule chose qui change est le caractère « i » qui apparaît, la région la plus petite est donc un rectangle qui entoure tout le « i », comme le montre la figure 12.2.

FIGURE 12.2 – La seule chose qui change est le caractère « i » qui apparaît

En revanche, quand on a un `Button` normal et qu'on appuie dessus, le texte ne change pas, mais toute la couleur du fond change, comme à la figure 12.3. Par conséquent la région la plus petite qui contient tous les éléments nouveaux ou qui auraient changé englobe tout l'arrière-plan et subséquemment englobe toute la vue.

FIGURE 12.3 – La couleur du fond change

Ainsi, en utilisant un rectangle, on peut très bien demander à une vue de se redessiner dans son intégralité de cette manière :

```
vue.invalidate(new Rect(vue.getLeft(), vue.getTop(), vue.
    getRight(), vue.getDown());
```

La propagation

Quand on demande à une vue de se dessiner, elle lance le processus puis transmet la requête à ses enfants si elle en a. Cependant, elle ne le transmet pas à tous ses enfants, seulement à ceux qui se trouvent dans sa région d'invalidation. Ainsi, le parent sera dessiné en premier, puis les enfants le seront dans l'ordre dans lequel ils sont placés dans l'arbre hiérarchique, mais uniquement s'ils doivent être redessinés.

Pour demander à une vue qu'elle se redessine, utilisez une des méthodes invalidate vues précédemment, pour qu'elle détermine sa région d'invalidité, se redessine puis propage l'instruction.

Méthode 1 : à partir d'une vue préexistante

Le principe ici sera de dériver d'un widget ou d'un layout qui est fourni par le SDK d'Android. Nous l'avons déjà fait par le passé, mais nous n'avions manipulé que le comportement *logique* de la vue, pas le comportement *visuel*.

De manière générale, quand on développe une vue, on fait en sorte d'implémenter les trois constructeurs standards. Petit rappel à ce sujet :

```
// Il y a un constructeur qui est utilisé pour instancier la
    vue depuis le code :
View(Context context);

// Un pour l'inflation depuis le XML :
View(Context context, AttributeSet attrs);
// Le paramètre attrs contenant les attributs définis en XML

// Et un dernier pour l'inflation en XML et dont un style est
    associé à la vue :
View(Context context, AttributeSet attrs, int defStyle);
// Le paramètre defStyle contenant une référence à une
    ressource, ou 0 si aucun style n'a été défini
```

De plus, on développe aussi les méthodes qui commencent par `on...`. Ces méthodes sont des fonctions de *callback* et elles sont appelées dès qu'une méthode au nom identique (mais sans `on...`) est utilisée. Je vous ai par exemple parlé de `void measure (int widthMeasureSpec, int heightMeasureSpec)`, à chacune de ses exécutions, la fonction de *callback* `void onMeasure (int widthMeasureSpec, int heightMeasureSpec)` est lancée. Vous voyez, c'est simple comme bonjour.

Vous trouverez sur la documentation une liste intégrale des méthodes que vous pouvez implémenter.

▷ Liste des méthodes
Code web : 493745

Par exemple, j'ai créé un bouton qui permet de visualiser plusieurs couleurs. Tout d'abord, j'ai déclaré une ressource qui contient une liste de couleurs :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <array name="colors">
    <item>#FF0000</item>
    <item>#0FF000</item>
    <item>#000FF0</item>
    <item>#FFFFFF</item>
  </array>
</resources>
```

Ce type de ressources s'appelle un `TypedArray`, c'est-à-dire un tableau qui peut contenir n'importe quelles autres ressources. Une fois ce tableau désérialisé, je peux récupérer les éléments qui le composent avec la méthode appropriée, dans notre cas, comme nous manipulons des couleurs, `int getColor (int position, int defaultValue)` (`position` étant la position de l'élément voulu et `defaultValue` la valeur renvoyée si l'élément n'est pas trouvé).

```
import android.content.Context;
import android.content.res.Resources;
import android.content.res.TypedArray;

import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Rect;

import android.util.AttributeSet;

import android.view.MotionEvent;

import android.widget.Button;

public class ColorButton extends Button {
  /** Liste des couleurs disponibles **/
  private TypedArray mCouleurs = null;
  /** Position dans la liste des couleurs **/
  private int position = 0;

  /**
   * Constructeur utilisé pour inflater avec un style
   */
  public ColorButton(Context context, AttributeSet attrs, int
```

```
        defStyle) {
25      super(context, attrs, defStyle);
26          init();
27    }
28
29    /**
30     * Constructeur utilisé pour inflater sans style
31     */
32    public ColorButton(Context context, AttributeSet attrs) {
33          super(context, attrs);
34          init();
35    }
36
37    /**
38     * Constructeur utilisé pour construire dans le code
39     */
40    public ColorButton(Context context) {
41          super(context);
42          init();
43    }
44
45    private void init() {
46          // Je récupère mes ressources
47          Resources res = getResources();
48          // Et dans ces ressources je récupère mon tableau de
                couleurs
49          mCouleurs = res.obtainTypedArray(R.array.colors);
50
51          setText("Changer de couleur");
52    }
53
54    /* ... */
55
56  }
```

Je redéfinis `void onLayout (boolean changed, int left, int top, int right, int bottom)` pour qu'à chaque fois que la vue est redimensionnée je puisse changer la taille du carré qui affiche les couleurs de manière à ce qu'il soit toujours conforme au reste du bouton.

```
1  /** Rectangle qui délimite le dessin */
2  private Rect mRect = null;
3
4  @Override
5  protected void onLayout (boolean changed, int left, int top,
       int right, int bottom)
6  {
7    //Si le layout a changé
8    if(changed)
9      //On redessine un nouveau carré en fonction des nouvelles
          dimensions
```

```
10        mRect = new Rect(Math.round(0.5f * getWidth() - 50),
11                         Math.round(0.75f * getHeight() - 50),
12                         Math.round(0.5f * getWidth() + 50),
13                         Math.round(0.75f * getHeight() + 50));
14    //Ne pas oublier
15    super.onLayout(changed, left, top, right, bottom);
16  }
```

J'implémente `boolean onTouchEvent (MotionEvent event)` pour qu'à chaque fois que l'utilisateur appuie sur le bouton la couleur qu'affiche le carré change. Le problème est que cet évènement se lance à chaque toucher, et qu'un toucher ne correspond pas forcément à un clic, mais aussi à n'importe quelle fois où je bouge mon doigt sur le bouton, ne serait-ce que d'un pixel. Ainsi, la couleur change constamment si vous avez le malheur de bouger le doigt quand vous restez appuyé sur le bouton. C'est pourquoi j'ai rajouté une condition pour que le dessin ne réagisse que quand on appuie sur le bouton, pas quand on bouge ou qu'on lève le doigt. Pour cela, j'ai utilisé la méthode `int getAction ()` de `MotionEvent`. Si la valeur retournée est `MotionEvent.ACTION_DOWN`, c'est que l'évènement qui a déclenché le lancement de la méthode est un clic.

```
1  /** Outil pour peindre */
2  private Paint mPainter = new Paint(Paint.ANTI_ALIAS_FLAG);
3
4  @Override
5  public boolean onTouchEvent(MotionEvent event) {
6    // Uniquement si on appuie sur le bouton
7    if(event.getAction() == MotionEvent.ACTION_DOWN) {
8      // On passe à la couleur suivante
9      position ++;
10     // Évite de dépasser la taille du tableau
11     // (dès qu'on arrive à la longueur du tableau, on repasse à
           0)
12     position %= mCouleurs.length();
13
14     // Change la couleur du pinceau
15     mPainter.setColor(mCouleurs.getColor(position, -1));
16
17     // Redessine la vue
18     invalidate();
19   }
20   // Ne pas oublier
21   return super.onTouchEvent(event);
22 }
```

Enfin, j'écris ma propre version de `void onDraw(Canvas canvas)` pour dessiner le carré dans sa couleur actuelle. L'objet `Canvas` correspond à la fois à la toile sur laquelle on peut dessiner et à l'outil qui permet de dessiner, alors qu'un objet `Paint` indique juste au `Canvas` *comment* il faut dessiner, mais pas *ce qu'il faut* dessiner.

```
1  @Override
2  protected void onDraw(Canvas canvas) {
```

```
3      // Dessine le rectangle à l'endroit voulu avec la couleur
           voulue
4      canvas.drawRect(mRect, mPainter);
5      // Ne pas oublier
6      super.onDraw(canvas);
7   }
```

Vous remarquerez qu'à la fin de chaque méthode de type `on...`, je fais appel à l'équivalent de la superclasse de cette méthode. C'est tout simplement parce que les superclasses effectuent des actions pour la classe `Button` qui doivent être faites sous peine d'un comportement incorrect du bouton.

Ce qui donne la figure 12.4.

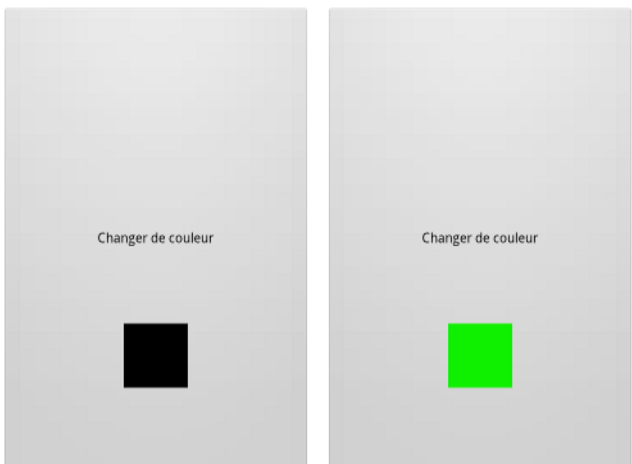


FIGURE 12.4 – On a un petit carré en bas de notre bouton (écran de gauche) et dès qu'on appuie sur le bouton le carré change de couleur (écran de droite)

Méthode 2 : une vue composite

On peut très bien se contenter d'avoir une vue qui consiste en un assemblage de vues qui existent déjà. D'ailleurs vous connaissez déjà au moins deux vues composites ! Pensez à `Spinner`, c'est un `Button` avec une `ListView`, non ? Et `AutoCompleteTextView`, c'est un `EditText` associé à une `ListView` aussi !

Logiquement, cette vue sera un assemblage d'autres vues et par conséquent ne sera pas un widget — qui ne peut pas contenir d'autres vues — mais bien un layout, elle devra donc dériver de `ViewGroup` ou d'une sous-classe de `ViewGroup`.

Je vais vous montrer une vue qui permet d'écrire du texte en HTML et d'avoir le résultat en temps réel. J'ai appelé ce widget `ToHtmlView`. Je n'explique pas le code

ligne par ligne puisque vous connaissez déjà tous ces concepts.

```
import android.content.Context;
import android.text.Editable;
import android.text.Html;
import android.text.InputType;
import android.text.TextWatcher;
import android.util.AttributeSet;

import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.TextView;

public class ToHtmlView extends LinearLayout {
  /** Pour insérer du texte */
  private EditText mEdit = null;
  /** Pour écrire le résultat */
  private TextView mText = null;

  /**
   * Constructeur utilisé quand on construit la vue dans le
       code
   * @param context
   */
  public ToHtmlView(Context context) {
    super(context);
    init();
  }

  /**
   * Constructeur utilisé quand on inflate la vue depuis le XML
   * @param context
   * @param attrs
   */
  public ToHtmlView(Context context, AttributeSet attrs) {
    super(context, attrs);
    init();
  }

  public void init() {
    // Paramètres utilisés pour indiquer la taille voulue pour
        la vue
    int wrap = LayoutParams.WRAP_CONTENT;
    int fill = LayoutParams.FILL_PARENT;

    // On veut que notre layout soit de haut en bas
    setOrientation(LinearLayout.VERTICAL);
    // Et qu'il remplisse tout le parent.
    setLayoutParams(new LayoutParams(fill, fill));

    // On construit les widgets qui sont dans notre vue
```

```
48        mEdit = new EditText(getContext());
49        // Le texte sera de type web et peut être long
50        mEdit.setInputType(InputType.
              TYPE_TEXT_VARIATION_WEB_EDIT_TEXT | InputType.
              TYPE_TEXT_FLAG_MULTI_LINE);
51        // Il fera au maximum dix lignes
52        mEdit.setMaxLines(10);
53        // On interdit le scrolling horizontal pour des questions
              de confort
54        mEdit.setHorizontallyScrolling(false);
55
56        // Listener qui se déclenche dès que le texte dans l'
              EditText change
57        mEdit.addTextChangedListener(new TextWatcher() {
58
59          // À chaque fois que le texte est édité
60          @Override
61          public void onTextChanged(CharSequence s, int start, int
                before, int count) {
62            // On change le texte en Spanned pour que les balises
                  soient interprétées
63            mText.setText(Html.fromHtml(s.toString()));
64          }
65
66          // Après que le texte a été édité
67          @Override
68          public void beforeTextChanged(CharSequence s, int start,
                int count, int after) {
69
70          }
71
72          // Après que le texte a été édité
73          @Override
74          public void afterTextChanged(Editable s) {
75
76          }
77        });
78
79        mText = new TextView(getContext());
80        mText.setText("");
81
82        // Puis on rajoute les deux widgets à notre vue
83        addView(mEdit, new LinearLayout.LayoutParams(fill, wrap));
84        addView(mText, new LinearLayout.LayoutParams(fill, wrap));
85      }
86    }
```

Ce qui donne, une fois intégré, la figure 12.5.

Mais j'aurais très bien pu passer par un fichier XML aussi ! Voici comment j'aurais pu faire :

Salut les <u>Zéros</u>, j'espère que tout va bien pour vous !

Salut les Zéros, j'espère que tout va bien pour **vous** !

FIGURE 12.5 – Le rendu du code

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:orientation="vertical" >

  <EditText
    android:id="@+id/edit"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:inputType="textWebEditText|textMultiLine"
    android:maxLines="10"
    android:scrollHorizontally="false">
    <requestFocus />
  </EditText>

  <TextView
    android:id="@+id/text"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="" />

</LinearLayout>
```

L'avantage par rapport aux deux autres méthodes, c'est que cette technique est très facile à mettre en place (pas de méthodes `onDraw` ou de `onMeasure` à redéfinir) et puissante. En revanche, on a beaucoup moins de contrôle.

Méthode 3 : créer une vue en partant de zéro

Il vous faut penser à tout ici, puisque votre vue dérivera directement de `View` et que cette classe ne gère pas grand-chose. Ainsi, vous savez que par défaut une vue est un carré blanc de 100 pixels de côté, il faudra donc au moins redéfinir `void onMeasure (int widthMeasureSpec, int heightMeasureSpec)` et `void onDraw (Canvas canvas)`. De plus, vous devez penser aux différents évènements (est-ce qu'il faut réagir au toucher, et si oui comment ? et à l'appui sur une touche ?), aux attributs de votre vue, aux constructeurs, etc.

Dans mon exemple, j'ai décidé de faire un échiquier.

La construction programmatique

Tout d'abord, j'implémente tous les constructeurs qui me permettront d'instancier des objets depuis le code. Pour cela, je redéfinis le constructeur standard et je développe un autre constructeur qui me permet de déterminer quelles sont les couleurs que je veux attribuer pour les deux types de case.

```
/** Pour la première couleur */
private Paint mPaintOne = null;
/** Pour la seconde couleur */
private Paint mPaintTwo = null;

public ChessBoardView(Context context) {
  super(context);
  init(-1, -1);
}

public ChessBoardView(Context context, int one, int two) {
  super(context);
  init(one, two);
}

private void init(int one, int two) {
  mPaintTwo = new Paint(Paint.ANTI_ALIAS_FLAG);
  if(one == -1)
    mPaintTwo.setColor(Color.LTGRAY);
  else
    mPaintTwo.setColor(one);

  mPaintOne = new Paint(Paint.ANTI_ALIAS_FLAG);
  if(two == -1)
    mPaintOne.setColor(Color.WHITE);
  else
    mPaintOne.setColor(two);
}
```

La construction par inflation

J'exploite les deux constructeurs destinés à l'inflation pour pouvoir récupérer les attributs que j'ai pu passer en attributs. En effet, il m'est possible de définir mes propres attributs pour ma vue. Pour cela, il me faut créer des ressources de type `attr` dans un tableau d'attributs. Ce tableau est un nœud de type `declare-styleable`. J'attribue un nom à chaque élément qui leur servira d'identifiant. Enfin, je peux dire pour chaque `attr` quel type d'informations il contiendra.

```
<resources>
```

```
2    <declare-styleable name="ChessBoardView">
3      <!-- L'attribut d'identifiant "colorOne" est de type "color
         " -->
4      <attr name="colorOne" format="color"/>
5      <attr name="colorTwo" format="color"/>
6    </declare-styleable>
7  </resources>
```

Pour utiliser ces attributs dans le layout, il faut tout d'abord déclarer utiliser un **namespace**, comme on le fait pour pouvoir utiliser les attributs qui appartiennent à Android : `xmlns:android="http://schemas.android.com/apk/res/android"`.

Cette déclaration nous permet d'utiliser les attributs qui commencent par `android:` dans notre layout, elle nous permettra donc d'utiliser nos propres attributs de la même manière.

Pour cela, on va se contenter d'agir d'une manière similaire en remplaçant `xmlns:android` par le nom voulu de notre namespace et `http://schemas.android.com/apk/res/android` par notre package actuel. Dans mon cas, j'obtiens :

```
xmlns:sdzName="http://schemas.android.com/apk/res/sdz.
   chapitreDeux.chessBoard"
```

Ce qui me donne ce XML :

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res
       /android"
3    xmlns:sdzName="http://schemas.android.com/apk/res/sdz.
       chapitreDeux.chessBoard"
4
5    android:layout_width="fill_parent"
6    android:layout_height="fill_parent"
7    android:orientation="vertical" >
8
9    <sdz.chapitreDeux.chessBoard.ChessBoardView
10     android:layout_width="fill_parent"
11     android:layout_height="fill_parent"
12     sdzName:colorOne="#FF0000"
13     sdzName:colorTwo="#00FF00" />
14 </LinearLayout>
```

Il me suffit maintenant de récupérer les attributs comme nous l'avions fait précédemment :

```
1  // attrs est le paramètre qui contient les attributs de notre
       objet en XML
2  public ChessBoardView(Context context, AttributeSet attrs, int
       defStyle) {
3    super(context, attrs, defStyle);
4    init(attrs);
5  }
```

```
6
7   // idem
8   public ChessBoardView(Context context, AttributeSet attrs) {
9     super(context, attrs);
10    init(attrs);
11  }
12
13  private void init(AttributeSet attrs) {
14    // Je récupère mon tableau d'attributs depuis le paramètre
          que m'a donné le constructeur
15    TypedArray attr = getContext().obtainStyledAttributes(attrs,
          R.styleable.ChessBoardView);
16    // Il s'agit d'un TypedArray qu'on sait déjà utiliser, je ré
          cupère la valeur de la couleur, 1 ou -1 si on ne la trouve
           pas
17    int tmpOne = attr.getColor(R.styleable.
          ChessBoardView_colorOne, -1);
18    // Je récupère la valeur de la couleur, 2 ou -1 si on ne la
          trouve pas
19    int tmpTwo = attr.getColor(R.styleable.
          ChessBoardView_colorTwo, -1);
20    init(tmpOne, tmpTwo);
21  }
```

onMeasure

La taille par défaut de 100 pixels est ridicule et ne conviendra jamais à un échiquier. Je vais faire en sorte que, si l'application me l'autorise, je puisse exploiter le carré le plus grand possible, et je vais faire en sorte qu'au pire notre vue prenne au moins la moitié de l'écran.

Pour cela, j'ai écrit une méthode qui calcule la dimension la plus grande entre la taille que me demande de prendre le layout et la taille qui correspond à la moitié de l'écran. Puis je compare en largeur et en hauteur quelle est la plus petite taille accordée, et mon échiquier s'accorde à cette taille.

Il existe plusieurs méthodes pour calculer la taille de l'écran. De mon côté, j'ai fait en sorte de l'intercepter depuis les ressources avec la méthode `DisplayMetrics getDisplayMetrics ()`. Je récupère ensuite l'attribut `heightPixels` pour avoir la hauteur de l'écran et `widthPixels` pour sa largeur.

```
1   /**
2    * Calcule la bonne mesure sur un axe uniquement
3    * @param spec - Mesure sur un axe
4    * @param screenDim - Dimension de l'écran sur cet axe
5    * @return la bonne taille sur cet axe
6    */
```

```
private int singleMeasure(int spec, int screenDim) {
  int mode = MeasureSpec.getMode(spec);
  int size = MeasureSpec.getSize(spec);

  // Si le layout n'a pas précisé de dimensions, la vue prendra
      la moitié de l'écran
  if(mode == MeasureSpec.UNSPECIFIED)
    return screenDim/2;
  else
    // Sinon, elle prendra la taille demandée par le layout
    return size;
}

@Override
protected void onMeasure (int widthMeasureSpec, int
    heightMeasureSpec) {
  // On récupère les dimensions de l'écran
  DisplayMetrics metrics = getContext().getResources().
      getDisplayMetrics();
  // Sa largeur...
  int screenWidth = metrics.widthPixels;
  // ... et sa hauteur
  int screenHeight = metrics.heightPixels;

  int retourWidth = singleMeasure(widthMeasureSpec, screenWidth
      );
  int retourHeight = singleMeasure(heightMeasureSpec,
      screenHeight);

  // Comme on veut un carré, on n'aura qu'une taille pour les
      deux axes, la plus petite possible
  int retour = Math.min(retourWidth, retourHeight);

  setMeasuredDimension(retour, retour);
}
```

Toujours avoir dans son implémentation de `onMeasure` un appel à la méthode `void setMeasuredDimension (int measuredWidth, int measuredHeight)`, sinon votre vue vous renverra une exception.

onDraw

Il ne reste plus qu'à dessiner notre échiquier ! Ce n'est pas grave si vous ne comprenez pas l'algorithme, du moment que vous avez compris toutes les étapes qui me permettent d'afficher cet échiquier tant voulu.

```
@Override
protected void onDraw(Canvas canvas) {
```

```
3     // Largeur de la vue
4     int width = getWidth();
5     // Hauteur de la vue
6     int height = getHeight();
7
8     int step = 0, min = 0;
9     // La taille minimale entre la largeur et la hauteur
10    min = Math.min(width, height);
11
12    // Comme on ne veut que 8 cases par ligne et 8 lignes, on
          divise la taille par 8
13    step = min / 8;
14
15    // Détermine quand on doit changer la couleur entre deux
          cases
16    boolean switchColor = true;
17    for(int i = 0 ; i < min ; i += step) {
18      for(int j = 0 ; j < min ; j += step) {
19        if(switchColor)
20          canvas.drawRect(i, j, i + step, j + step, mPaintTwo);
21        else
22          canvas.drawRect(i, j, i + step, j + step, mPaintOne);
23        // On change de couleur à chaque ligne...
24        switchColor = !switchColor;
25      }
26      // ... et à chaque case
27      switchColor = !switchColor;
28    }
29  }
```

Ce qui peut donner la figure 12.6.

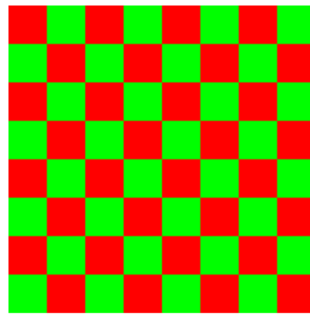


FIGURE 12.6 – Le choix des couleurs est discutable

En résumé

- Lors de la déclaration de nos interfaces graphiques, la hiérarchie des vues que nous déclarons aura toujours un layout parent qu'Android place sans nous le dire.
- Le placement est l'opération pendant laquelle Android placera les vues dans l'interface graphique. Elle se caractérise par l'appel de la méthode `void measure(int widthMeasureSpec, int heightMeasureSpec)` pour déterminer les dimensions réelles de votre vue et ensuite de la méthode `void layout(int bord_gauche, int plafond, int bord_droit, int plancher)` pour la placer à l'endroit demandé.
- Toutes les vues que vous avez déclarées dans vos interfaces offrent la possibilité de connaitre leurs dimensions une fois qu'elles ont été dessinées à l'écran.
- Une vue ne redessine que les zones qui ont été modifiées. Ces zones définissent ce qu'on appelle l'obsolescence régionale. Il est possible de demander à une vue de se forcer à se redessiner par le biais de la méthode `void invalidate ()` et toutes ses dérivées.
- Vous pouvez créer une nouvelle vue personnalisée à partir d'une vue préexistante que vous décidez d'étendre dans l'une de vos classes Java.
- Vous pouvez créer une nouvelle vue personnalisée à partir d'un assemblage d'autres vues préexistantes comme le ferait un `Spinner` qui est un assemblage entre un `Button` et une `ListView`.
- Vous pouvez créer une nouvelle vue personnalisée à partir d'une feuille blanche en dessinant directement sur le canvas de votre vue.

Troisième partie

Vers des applications plus complexes

Chapitre 13

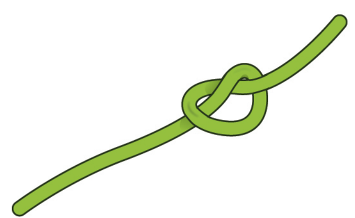
Préambule : quelques concepts avancés

Difficulté :

Le Manifest est un fichier que vous trouverez à la racine de votre projet sous le nom d'`AndroidManifest.xml` et qui vous permettra de spécifier différentes options pour vos projets, comme le matériel nécessaire pour les faire fonctionner, certains paramètres de sécurité ou encore des informations plus ou moins triviales telles que le nom de l'application ainsi que son icône.

Mais ce n'est pas tout, c'est aussi la première étape à maîtriser afin de pouvoir insérer plusieurs activités au sein d'une même application, ce qui sera la finalité des deux prochains chapitres.

Ce chapitre se chargera aussi de vous expliquer plus en détail comment manipuler le cycle d'une activité. En effet, pour l'instant nous avons toujours tout inséré dans `onCreate`, mais il existe des situations pour lesquelles cette attitude n'est pas du tout la meilleure à adopter.



Généralités sur le nœud <manifest>

Ce fichier est indispensable pour tous les projets Android, c'est pourquoi il est créé par défaut. Si je crée un nouveau projet, voici le Manifest qui est généré :

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sdz.chapitreTrois"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk android:minSdkVersion="7" />

  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name" >

    <activity
      android:name=".ManifestActivity"
      android:label="@string/app_name" >

      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>

    </activity>

  </application>

</manifest>
```

Voyons un petit peu de quoi il s'agit ici.

<manifest>

La racine du Manifest est un nœud de type `<manifest>`. Comme pour les vues et les autres ressources, on commence par montrer qu'on utilise l'espace de noms `android` :

```xml
xmlns:android="http://schemas.android.com/apk/res/android"
```

Puis, on déclare dans quel package se trouve notre application :

```xml
package="sdz.chapitreTrois"
```

... afin de pouvoir utiliser directement les classes qui se situent dans ce package sans avoir à préciser à chaque fois qu'elles s'y situent. Par exemple, dans notre Manifest

actuel, vous pouvez voir la ligne suivante : `android:name=".ManifestActivity"`. Elle fait référence à l'activité principale de mon projet : `ManifestActivity`. Cependant, si nous n'avions pas précisé `package="sdz.chapitreTrois"`, alors le nœud `android:name` aurait dû valoir `android:name="sdz.chapitreTrois.ManifestActivity"`. Imaginez seulement que nous ayons à le faire pour chaque activité de notre application… Cela risquerait d'être vite usant.

N'oubliez pas que le nom de package doit être unique si vous voulez être publié sur le Play Store. De plus, une fois votre application diffusée, ne changez pas ce nom puisqu'il agira comme un identifiant unique pour votre application.

Toujours dans le nœud `<manifest>`, il est ensuite possible d'indiquer quelle est la version actuelle du logiciel :

```
android:versionCode="1"
android:versionName="1.0"
```

L'attribut `android:versionCode` doit être un nombre entier (positif et sans virgule) qui indique quelle est la version actuelle de l'application. Mais attention, il ne s'agit pas du nombre qui sera montré à l'utilisateur, juste celui considéré par le Play Store. Si vous soumettez votre application avec un code de version supérieur à celui de votre ancienne soumission, alors le Play Store saura que l'application a été mise à jour. En revanche, le `android:versionName` peut être n'importe quelle chaîne de caractères et sera montré à l'utilisateur. Rien ne vous empêche donc de mettre `android:versionName="Première version alpha - 0.01a"` par exemple.

`<uses-sdk>`

On utilise ce nœud de manière à pouvoir filtrer les périphériques sur lesquels l'application est censée fonctionner en fonction de leur version d'Android. Ainsi, il vous est possible d'indiquer la version minimale de l'API que doit utiliser le périphérique :

```
<uses-sdk android:minSdkVersion="7" />
```

Ici, il faudra la version 2.1 d'Android (API 7) ou supérieure pour pouvoir utiliser cette application.

Votre application ne sera proposée sur le Google Play qu'à partir du moment où l'utilisateur utilise cette version d'Android ou une version supérieure.

Il existe aussi un attribut `android:targetSdkVersion` qui désigne non pas la version minimale d'Android demandée, mais plutôt la version à partir de laquelle on pourra exploiter à fond l'application. Ainsi, vous avez peut-être implémenté des fonctionnalités qui ne sont disponibles qu'à partir de versions d'Android plus récentes que la version minimale renseignée avec `android:minSdkVersion`, tout en faisant en sorte que l'application soit fonctionnelle en utilisant une version d'Android égale ou supérieure à

celle précisée dans `android:minSdkVersion`.

Vous pouvez aussi préciser une limite maximale à respecter avec `android:maxSdkVersion` si vous savez que votre application ne fonctionne pas sur les plateformes les plus récentes, mais je ne vous le conseille pas, essayez plutôt de rendre votre application compatible avec le plus grand nombre de terminaux !

<uses-feature>

Étant donné qu'Android est destiné à une très grande variété de terminaux différents, il fallait un moyen pour faire en sorte que les applications qui utilisent certains aspects hardware ne puissent être proposées que pour les téléphones qui possèdent ces capacités techniques. Par exemple, vous n'allez pas proposer un logiciel pour faire des photographies à un téléphone qui ne possède pas d'appareil photo (même si c'est rare) ou de capture sonore pour une tablette qui n'a pas de microphone (ce qui est déjà moins rare).

Ce nœud peut prendre trois attributs, mais je n'en présenterai que deux :

```
<uses-feature
  android:name="material"
  android:required="boolean" />
```

android:name

Vous pouvez préciser le nom du matériel avec l'attribut `android:name`. Par exemple, pour l'appareil photo (et donc la caméra), on mettra :

```
<uses-feature android:name="android.hardware.camera" />
```

Cependant, il arrive qu'on ne cherche pas uniquement un composant particulier mais une fonctionnalité de ce composant ; par exemple pour permettre à l'application de n'être utilisée que sur les périphériques qui ont un appareil photo avec autofocus, on utilisera :

```
<uses-feature android:name="android.hardware.camera.autofocus"
    />
```

Vous trouverez la liste exhaustive des valeurs que peut prendre `android:name` sur la documentation.

▷ Voir la liste
Code web : 879240

android:required

Comme cet attribut n'accepte qu'un booléen, il ne peut prendre que deux valeurs :

1. `true` : ce composant est indispensable pour l'utilisation de l'application.
2. `false` : ce composant n'est pas indispensable, mais sa présence est recommandée.

<supports-screens>

Celui-ci est très important, il vous permet de définir quels types d'écran supportent votre application. Cet attribut se présente ainsi :

```
<supports-screens android:smallScreens="boolean"
                  android:normalScreens="boolean"
                  android:largeScreens="boolean" />
```

Si un écran est considéré comme petit, alors il entrera dans la catégorie `smallScreen`, s'il est moyen, c'est un `normalScreen`, et les grands écrans sont des `largeScreen`.

L'API 9 a vu l'apparition de l'attribut `<supports-screens android:xlargeScreens="boolean" />`. Si cet attribut n'est pas défini, alors votre application sera lancée avec un mode de compatibilité qui agrandira tous les éléments graphiques de votre application, un peu comme si on faisait un zoom sur une image. Le résultat sera plus laid que si vous développiez une interface graphique dédiée, mais au moins votre application fonctionnera.

Le nœud <application>

Le nœud le plus important peut-être. Il décrit les attributs qui caractérisent votre application et en énumère les composants de votre application. Par défaut, votre application n'a qu'un composant, l'activité principale. Mais voyons d'abord les attributs de ce nœud.

Vous pouvez définir l'icône de votre application avec `android:icon`, pour cela vous devez faire référence à une ressource drawable : `android:icon="@drawable/ic_launcher"`.

Ne confondez pas avec `android:logo` qui, lui, permet de changer l'icône *uniquement* dans l'ActionBar.

Il existe aussi un attribut `android:label` qui permet de définir le nom de notre application.

Les thèmes

Vous savez déjà comment appliquer un style à plusieurs vues pour qu'elles respectent les mêmes attributs. Et si nous voulions que toutes nos vues respectent un même style au sein d'une application ? Que les textes de toutes ces vues restent noirs par exemple ! Ce serait contraignant d'appliquer le style à chaque vue. C'est pourquoi il est possible d'appliquer un style à une application, auquel cas on l'appelle un **thème**. Cette opération se déroule dans le Manifest, il vous suffit d'insérer l'attribut :

```
1 android:theme="@style/blackText"
```

Vous pouvez aussi exploiter les thèmes par défaut fournis par Android, par exemple pour que votre application ressemble à une boîte de dialogue :

```
1 <activity android:theme="@android:style/Theme.NoTitleBar">
```

Vous en retrouverez d'autres sur la documentation.

▷ Liste des thèmes
Code web : 352864

Laissez-moi maintenant vous parler de la notion de composants. Ce sont les éléments qui composeront vos projets. Il en existe cinq types, mais vous ne connaissez pour l'instant que les activités, alors je ne vais pas vous embrouiller plus avec ça. Sachez juste que votre application sera au final un ensemble de composants qui interagissent, entre eux et avec le reste du système.

`<activity>`

Ce nœud permet de décrire toutes les activités contenues dans notre application. Comme je vous l'ai déjà dit, une activité correspond à un écran de votre application, donc, si vous voulez avoir plusieurs écrans, il vous faudra plusieurs activités.

Le seul attribut vraiment indispensable ici est `android:name`, qui indique quelle est la classe qui implémente l'activité.

`android:name` définit aussi un identifiant pour Android qui permet de repérer ce composant parmi tous. Ainsi, ne changez pas l'attribut `android:name` d'un composant au cours d'une mise à jour, sinon vous risquez de rencontrer des effets de bord assez désastreux.

Vous pouvez aussi préciser un nom pour chaque activité avec `android:label`, c'est le mot qui s'affichera en haut de l'écran sur notre activité. Si vous ne le faites pas, c'est la `String` renseignée dans le `android:label` du nœud `<application>` qui sera utilisée.

Vous pouvez voir un autre nœud de type `<intent-filter>` qui indique comment se lancera cette activité. Pour l'instant, sachez juste que l'activité qui sera lancée depuis le menu principal d'Android contiendra toujours dans son Manifest ces lignes-ci :

```
1 <intent-filter>
2   <action android:name="android.intent.action.MAIN" />
3   <category android:name="android.intent.category.LAUNCHER" />
4 </intent-filter>
```

Je vous donnerai beaucoup plus de détails dans le prochain chapitre.

Vous pouvez aussi définir un thème pour une activité, comme nous l'avons fait pour une application.

Les permissions

Vous le savez sûrement, quand vous téléchargez une application sur le Play Store, on vous propose de regarder les autorisations que demande cette application avant de commencer le téléchargement (voir figure 13.1). Par exemple, pour une application qui vous permet de retenir votre numéro de carte bancaire, on peut légitimement se poser la question de savoir si dans ses autorisations se trouve « Accès à internet ».

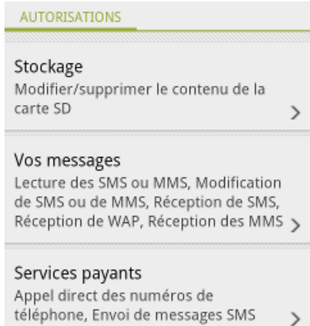


FIGURE 13.1 – On vous montre les autorisations que demande l'application avant de la télécharger

Par défaut, aucune application ne peut exécuter d'opération qui puisse nuire aux autres applications, au système d'exploitation ou à l'utilisateur. Cependant, Android est constitué de manière à ce que les applications puissent partager. C'est le rôle des permissions, elles permettent de limiter l'accès aux composants de vos applications.

Utiliser les permissions

Afin de pouvoir utiliser certaines API d'Android, comme l'accès à internet dans le cas précédent, vous devez préciser dans le Manifest que vous utilisez les permissions. Ainsi, l'utilisateur final est averti de ce que vous souhaitez faire, c'est une mesure de protection importante à laquelle vous devez vous soumettre.

Vous trouverez sur la documentation une liste des permissions qui existent déjà.

▷ Liste des permissions
Code web : 193139

Ainsi, pour demander un accès à internet, on indiquera la ligne :

```
1 <uses-permission android:name="android.permission.INTERNET" />
```

Créer ses permissions

Il est important que votre application puisse elle aussi partager ses composants puisque c'est comme ça qu'elle sait se rendre indispensable.

Une permission doit être de cette forme-ci :

```
<permission android:name="string"
            android:label="string resource"
            android:description="string resource"
            android:icon="drawable resource"
            android:protectionLevel=XXX />
```

Où :

- `android:name` est le nom qui sera utilisé dans un `uses-permission` pour faire référence à cette permission. Ce nom doit être unique.
- `android:label` est le nom qui sera indiqué à l'utilisateur, faites-en donc un assez explicite.
- `android:description` est une description plus complète de cette permission que le `label`.
- `android:icon` est assez explicite, il s'agit d'une icône qui est censée représenter la permission. Cet attribut est heureusement facultatif.
- `android:protectionLevel` indique le degré de risque du composant lié à cette permission. Il existe principalement trois valeurs :
  - `normal` si le composant est sûr et ne risque pas de dégrader le comportement du téléphone ;
  - `dangerous` si le composant a accès à des données sensibles ou peut dégrader le fonctionnement du périphérique ;
  - `signature` pour n'autoriser l'utilisation du composant que par les produits du même auteur (concrètement, pour qu'une application se lance sur votre terminal ou sur l'émulateur, il faut que votre application soit « approuvée » par Android. Pour ce faire, un certificat est généré — même si vous ne le demandez pas, l'ADT s'en occupe automatiquement — et il faut que les certificats des deux applications soient identiques pour que la permission soit acceptée).

Gérer correctement le cycle des activités

Comme nous l'avons vu, quand un utilisateur manipule votre application, la quitte pour une autre ou y revient, elle traverse plusieurs états symbolisés par le cycle de vie des activités, schématisé à la figure 13.2.

La transition entre chaque étape implique que votre application appelle une méthode. Cette méthode partage le même nom que l'état traversé, par exemple à la création est appelée la méthode `onCreate`.

Ces méthodes ne sont que des états de transition très éphémères entre les trois grands états dont nous avons déjà discuté : la période **active** peut être interrompue par la période **suspendue**, qui elle aussi peut être interrompue par la période **arrêtée**.

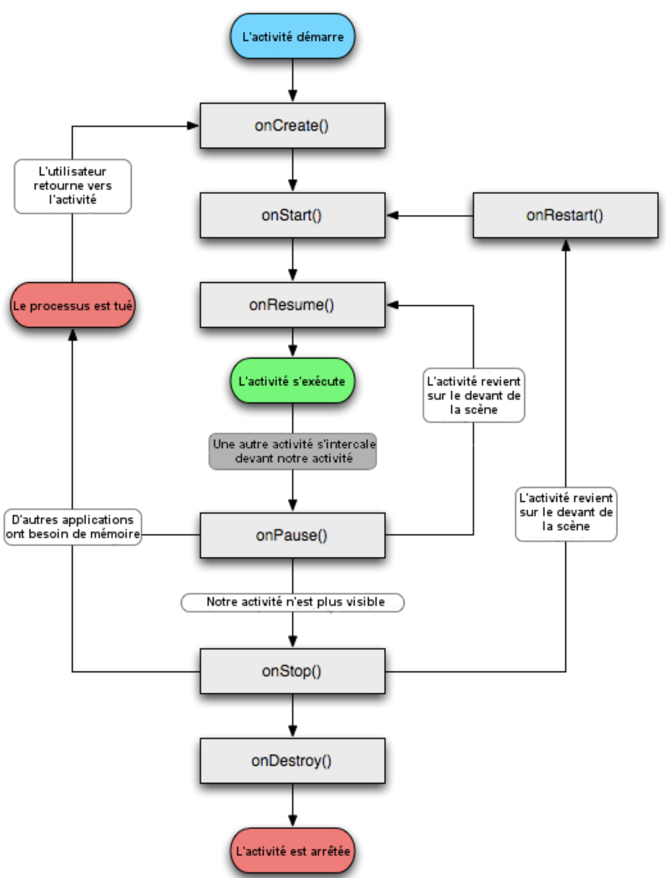


FIGURE 13.2 – Le cycle de vie d'une activité

Vous le comprendrez très vite, l'entrée dans chaque état est symbolisée par une méthode et la sortie de chaque état est symbolisée par une autre méthode. Ces deux méthodes sont complémentaires : ce qui est initialisé dans la première méthode doit être stoppé dans la deuxième, et ce presque toujours.

Sous les feux de la rampe : période suspendue

Cette période débute avec `onResume()` et se termine avec `onPause()`. On entre dans la période suspendue dès que votre activité n'est plus que partiellement visible, comme quand elle est voilée par une boîte de dialogue. Comme cette période arrive fréquemment, il faut que le contenu de ces deux méthodes s'exécute rapidement et nécessite peu de processeur.

`onPause()`

On utilise la méthode `onPause()` pour arrêter des animations, libérer des ressources telles que le GPS ou la caméra, arrêter des tâches en arrière-plan et de manière générale stopper toute activité qui pourrait solliciter le processeur. Attention, on évite de sauvegarder dans la base de données au sein de `onPause()`, car c'est une action qui prend beaucoup de temps à se faire, et il vaut mieux que `onPause()` s'exécute rapidement pour fluidifier la manipulation par l'utilisateur. De manière générale, il n'est pas nécessaire de sauvegarder des données dans cette méthode, puisque Android conserve une copie fonctionnelle de l'activité, et qu'au retour elle sera restaurée telle quelle.

`onResume()`

Cette méthode est exécutée à chaque fois que notre activité retourne au premier plan, mais aussi à chaque lancement, c'est pourquoi on l'utilise pour initialiser les ressources qui seront coupées dans le `onPause()`. Par exemple, dans votre application de localisation GPS, vous allez initialiser le GPS à la création de l'activité, mais pas dans le `onCreate(Bundle)`, plutôt dans le `onResume()` puisque vous allez le couper à chaque fois que vous passez dans le `onPause()`.

Convoquer le plan et l'arrière-plan : période arrêtée

Cette fois-ci, votre activité n'est plus visible du tout, mais elle n'est pas arrêtée non plus. C'est le cas si l'utilisateur passe de votre application à une autre (par exemple s'il retourne sur l'écran d'accueil), alors l'activité en cours se trouvera stoppée et on reprendra avec cette activité dès que l'utilisateur retournera dans l'application. Il est aussi probable que dans votre application vous ayez plus d'une activité, et passer d'une activité à l'autre implique que l'ancienne s'arrête.

Cet état est délimité par `onStop()` (toujours précédé de `onPause()`) et `onRestart()` (toujours suivi de `onResume()`, puis `onStart()`)). Cependant, il se peut que l'applica-

tion soit tuée par Android s'il a besoin de mémoire, auquel cas, après `onStop()`, l'application est arrêtée et, quand elle sera redémarrée, on reprendra à `onCreate(Bundle)`.

Là, en revanche, vous devriez sauvegarder les éléments dans votre base de données dans `onStop()` et les restaurer lorsque c'est nécessaire (dans `onStart()` si la restauration doit se faire au démarrage et après un `onStop()` ou dans `onResume()` si la restauration ne doit se faire qu'après un `onStop()`).

De la naissance à la mort

`onCreate(Bundle)`

Première méthode qui est lancée au démarrage de l'activité, c'est l'endroit privilégié pour initialiser l'interface graphique, pour démarrer les tâches d'arrière-plan qui s'exécuteront pendant toute la durée de vie de l'activité, pour récupérer des éléments de la base de données, etc.

Il se peut très bien que vous utilisiez une activité uniquement pour faire des calculs et prendre des décisions entre l'exécution de deux activités, auquel cas vous pouvez faire appel à la méthode `public void finish ()` pour passer directement à la méthode `onDestroy()`, qui symbolise la mort de l'activité. Notez bien qu'il s'agit du seul cas où il est recommandé d'utiliser la méthode `finish()` (c'est-à-dire qu'on évite d'ajouter un bouton pour arrêter son application par exemple).

`onDestroy()`

Il existe trois raisons pour lesquelles votre application peut atteindre la méthode `onDestroy`, c'est-à-dire pour lesquelles on va terminer notre application :

- Comme je l'ai déjà expliqué, il peut arriver que le téléphone manque de mémoire et qu'il ait besoin d'arrêter votre application pour en récupérer.
- Si l'utilisateur presse le bouton `Arrière` et que l'activité actuelle ne permet pas de retourner à l'activité précédente (ou s'il n'y en a pas !), alors on va quitter l'application.
- Enfin, si vous faites appel à la méthode `public void finish()` — mais on évite de l'utiliser en général —, il vaut mieux laisser l'utilisateur appuyer sur le bouton `Arrière`, parce qu'Android est conçu pour gérer le cycle des activités tout seul (c'est d'ailleurs la raison pour laquelle il faut éviter d'utiliser des *task killers*).

Comme vous le savez, c'est dans `onCreate(Bundle)` que doivent être effectuées les différentes initialisations ainsi que les tâches d'arrière-plan qu'on souhaite voir s'exécuter pendant toute l'application. Normalement, quand l'application arrive au `onDestroy()`, elle est déjà passée par `onPause()` et `onStop()`, donc la majorité des tâches de fond auront été arrêtées ; cependant, s'il s'agit d'une tâche qui devrait s'exécuter pendant toute la vie de l'activité — qui aura été démarrée dans `onCreate(Bundle)` —, alors c'est dans `onDestroy()` qu'il faudra l'arrêter.

Android passera d'abord par `onPause()` et `onStop()` dans tous les cas, à l'exception de l'éventualité où vous appeleriez la méthode `finish()` ! Si c'est le cas, on passe directement au `onDestroy()`. Heureusement, il vous est possible de savoir si le `onDestroy()` a été appelé suite à un `finish()` avec la méthode `public boolean isFinishing()`.

Si à un moment quelconque votre application lance une exception que vous ne catchez pas, alors l'application sera détruite sans passer par le `onDestroy()`, c'est pourquoi cette méthode n'est pas un endroit privilégié pour sauvegarder des données.

L'échange équivalent

Quand votre application est quittée de manière normale, par exemple si l'utilisateur presse le bouton `Arrière` ou qu'elle est encore ouverte et que l'utilisateur ne l'a plus consultée depuis longtemps, alors Android ne garde pas en mémoire de traces de vos activités, puisque l'application s'est arrêtée correctement. En revanche, si Android a dû tuer le processus, alors il va garder en mémoire une trace de vos activités afin de pouvoir les restaurer telles quelles. Ainsi, au prochain lancement de l'application, le paramètre de type `Bundle` de la méthode `onCreate(Bundle)` sera peuplé d'informations enregistrées sur l'état des vues de l'interface graphique.

Mais il peut arriver que vous ayez besoin de retenir d'autres informations qui, elles, ne sont pas sauvegardées par défaut. Heureusement, il existe une méthode qui est appelée à chaque fois qu'il y a des chances pour que l'activité soit tuée. Cette méthode s'appelle `protected void onSaveInstanceState(Bundle outState)`.

Un objet de type `Bundle` est l'équivalent d'une table de hachage qui à une chaîne de caractères associe un élément, mais seulement pour certains types précis. Vous pouvez voir dans la documentation tous les types qu'il est possible d'insérer. Par exemple, on peut insérer un entier et le récupérer à l'aide de :

```
1  private final static RESULTAT_DU_CALCUL = 0;
2
3  @Override
4  protected void onSaveInstanceState (Bundle bundle)
5  {
6    super.onSaveInstanceState(bundle);
7    bundle.putInt(RESULTAT_DU_CALCUL, 10);
8    /*
9     * On pourra le récupérer plus tard avec
10    * int resultat = bundle.getInt(RESULTAT_DU_CALCUL);
11   */
12 }
```

On ne peut pas mettre n'importe quel objet dans un `Bundle`, uniquement des objets sérialisables. La sérialisation est le procédé qui convertit un objet en un format qui peut être stocké (par exemple dans un fichier ou transmis sur un réseau) et ensuite reconstitué de manière parfaite. Vous faites le rapprochement avec la sérialisation d'une

vue en XML, n'est-ce pas ?

En ce qui concerne Android, on n'utilise pas la sérialisation standard de Java, avec l'interface `Java.io.Serializable`, parce que ce processus est trop lent. Or, quand nous essayons de faire communiquer des composants, il faut que l'opération se fasse de manière rapide. C'est pourquoi on utilise un système différent que nous aborderons en détail dans le prochain chapitre.

Cependant, l'implémentation par défaut de `onSaveInstanceState(Bundle)` ne sauvegarde pas toutes les vues, juste celles qui possèdent un identifiant ainsi que la vue qui a le focus, alors n'oubliez pas de faire appel à `super.onSaveInstanceState(Bundle)` pour vous simplifier la vie.

Par la suite, cet objet `Bundle` sera passé à `onCreate(Bundle)`, mais vous pouvez aussi choisir de redéfinir la méthode `onRestoreInstanceState(Bundle)`, qui est appelée après `onStart()` et qui recevra le même objet `Bundle`.

L'implémentation par défaut de `onRestoreInstanceState(Bundle)` restaure les vues sauvegardées par l'implémentation par défaut de `onSaveInstanceState ()`.

La figure suivante est un schéma qui vous permettra de mieux comprendre tous ces imbroglios.

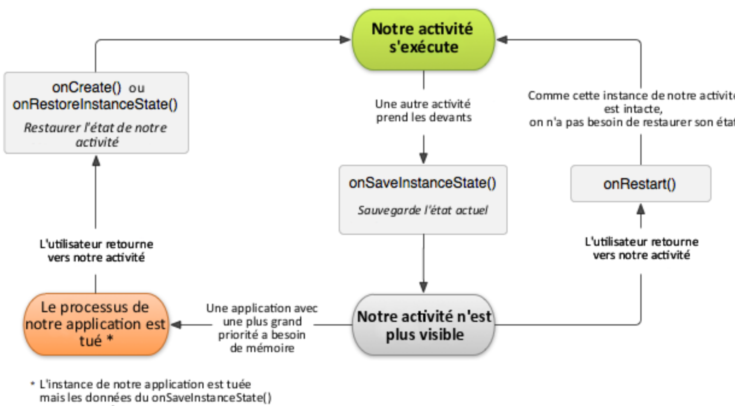


FIGURE 13.3 – Le cycle de sauvegarde de l'état d'une activité

Gérer le changement de configuration

Il se peut que la configuration de votre utilisateur change pendant qu'il utilise son terminal. Vous allez dire que je suis fou, mais un changement de configuration correspond simplement à ce qui pourrait contribuer à un changement d'interface graphique. Vous

vous rappelez les quantificateurs ? Eh bien, si l'un de ces quantificateurs change, alors on dit que la configuration change.

Et ça vous est déjà arrivé, j'en suis sûr. Réfléchissez ! Si l'utilisateur passe de paysage à portrait dans l'un de nos anciens projets, alors il change de configuration et par conséquent d'interface graphique.

Par défaut, dès qu'un changement qui pourrait changer les ressources utilisées par une application se produit, Android détruit tout simplement la ou les activités pour les recréer ensuite. Heureusement pour vous, Android va retenir les informations des widgets qui possèdent un identifiant. Dans une application très simple, on va créer un layout par défaut :

```
1  <LinearLayout xmlns:android="http://schemas.android.com/apk/res
       /android"
2    android:layout_width="fill_parent"
3    android:layout_height="fill_parent"
4    android:orientation="vertical" >
5
6    <EditText
7      android:id="@+id/editText"
8      android:layout_width="fill_parent"
9      android:layout_height="wrap_content" >
10
11   </EditText>
12
13   <EditText
14     android:layout_width="fill_parent"
15     android:layout_height="wrap_content" />
16
17 </LinearLayout>
```

Seul un de ces `EditText` possède un identifiant. Ensuite, on fait un layout presque similaire, mais avec un quantificateur pour qu'il ne s'affiche qu'en mode paysage :

```
1  <LinearLayout xmlns:android="http://schemas.android.com/apk/res
       /android"
2    android:layout_width="fill_parent"
3    android:layout_height="fill_parent"
4    android:orientation="vertical" >
5
6    <EditText
7      android:layout_width="fill_parent"
8      android:layout_height="wrap_content" >
9
10   </EditText>
11
12   <Button
13     android:layout_width="fill_parent"
14     android:layout_height="wrap_content"
15     />
16
```

```
17      <EditText
18        android:id="@+id/editText"
19        android:layout_width="fill_parent"
20        android:layout_height="wrap_content" />
21
22    </LinearLayout>
```

Remarquez bien que l'identifiant de l'`EditText` est passé à l'`EditText` du bas. Ainsi, quand vous lancez votre application, écrivez du texte dans les deux champs, comme à la figure 13.4.

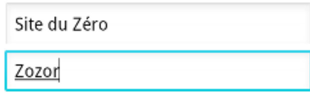


FIGURE 13.4 – Écrivez du texte dans les deux champs

Puis tournez votre terminal (avant qu'un petit malin ne casse son ordinateur en le mettant sur le côté : pour faire pivoter l'émulateur, c'est CTRL + F12 ou F11) et admirez le résultat, identique à la figure 13.5.

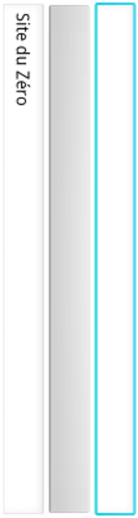


FIGURE 13.5 – Il y a perte d'information !

Vous voyez bien que le widget qui avait un identifiant a conservé son texte, mais pas l'autre ! Cela prouve bien qu'il peut y avoir perte d'information dès qu'un changement de configuration se produit.

Bien entendu, dans le cas des widgets, le problème est vite résolu puisqu'il suffit de leur ajouter un identifiant, mais il existe des informations à retenir en dehors des widgets. Alors, comment gérer ces problèmes-là ? Comme par défaut Android va détruire puis recréer les activités, vous pouvez très bien tout enregistrer dans la

méthode `onSaveInstanceState()`, puis tout restaurer dans `onCreate(Bundle)` ou `onRestoreInstanceState()`. Mais il existe un problème ! Vous ne pouvez passer que les objets sérialisables dans un `Bundle`. Alors comment faire ?

Il existe trois façons de faire :

- Utiliser une méthode alternative pour retenir vos objets, qui est spécifique aux changements de configuration.
- Gérer vous-mêmes les changements de configuration, auquel cas Android ne s'en chargera plus. Et comme cette technique est un peu risquée, je ne vais pas vous la présenter.
- Bloquer le changement de ressources.

Retenir l'état de l'activité

Donc, le problème avec `Bundle`, c'est qu'il ne peut pas contenir de gros objets et qu'en plus la sérialisation et la désérialisation sont des processus lents, alors que nous souhaiterions que la transition entre deux configurations soit fluide. C'est pourquoi nous allons faire appel à une autre méthode qui est appelée cette fois uniquement en cas de changement de configuration : `public Object onRetainNonConfigurationInstance()`. L'objet retourné peut être de n'importe quel ordre, vous pouvez même retourner directement une instance de l'activité si vous le souhaitez (mais bon, ne le faites pas).

Notez par ailleurs que la méthode `onRetainNonConfigurationInstance()` est appelée après `onStop()` mais avant `onDestroy()` et que vous feriez mieux de ne pas conserver des objets qui dépendent de la configuration (par exemple des chaînes de caractères qui changent en fonction de la langue) ou des objets qui sont liés à l'activité (un `Adapter` par exemple).

Ainsi, une des façons de procéder est de créer une classe spécialement dédiée à la détention de ces informations :

```
1  @Override
2  public Object onRetainNonConfigurationInstance() {
3    // La classe « DonneesConservees » permet de contenir tous
        les objets voulus
4    // Et la méthode "constituerDonnees" va construire un objet
5    // En fonction de ce que devra savoir la nouvelle instance de
        l'activité
6    DonneesConservees data = constituerDonnees();
7    return data;
8  }
```

Enfin, il est possible de récupérer cet objet dans le `onCreate(Bundle)` à l'aide de la méthode `public Object getLastNonConfigurationInstance()` :

```
1  @Override
2  public void onCreate(Bundle savedInstanceState) {
3    super.onCreate(savedInstanceState);
4    setContentView(R.layout.main);
5
```

```
6   DonneesConservees data = (DonneesConservees)
      getLastNonConfigurationInstance();
7   // S'il ne s'agit pas d'un retour depuis un changement de
      configuration, alors data est null
8   if(data == null)
9     ...
10 }
```

Empêcher le changement de ressources

De toute façon, il arrive parfois qu'une application n'ait de sens que dans une orientation. Pour lire un livre, il vaut mieux rester toujours en orientation portrait par exemple, de même il est plus agréable de regarder un film en mode paysage. L'idée ici est donc de conserver des fichiers de ressources spécifiques à une configuration, même si celle du terminal change en cours d'utilisation.

Pour ce faire, c'est très simple, il suffit de rajouter dans le nœud des composants concernés les lignes `android:screenOrientation = "portrait"` pour bloquer en mode portrait ou `android:screenOrientation = "landscape"` pour bloquer en mode paysage. Bon, le problème, c'est qu'Android va quand même détruire l'activité pour la recréer si on laisse ça comme ça, c'est pourquoi on va lui dire qu'on gère nous-mêmes les changements d'orientation en ajoutant la ligne `android:configChanges="orientation"` dans les nœuds concernés :

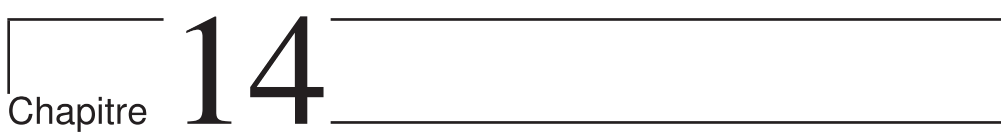
```
1  <manifest xmlns:android="http://schemas.android.com/apk/res/
     android"
2    package="fr.sdz.configuration.change"
3    android:versionCode="1"
4    android:versionName="1.0" >
5
6    <uses-sdk
7      android:minSdkVersion="7"
8      android:targetSdkVersion="15" />
9
10   <application
11     android:icon="@drawable/ic_launcher"
12     android:label="@string/app_name"
13     android:theme="@style/AppTheme" >
14     <activity
15       android:name=".MainActivity"
16       android:label="@string/title_activity_main"
17       android:configChanges="orientation"
18       android:screenOrientation="portrait" >
19       <intent-filter>
20         <action android:name="android.intent.action.MAIN" />
21
22         <category android:name="android.intent.category.
             LAUNCHER" />
23       </intent-filter>
```

```
24        </activity>
25    </application>
26
27 </manifest>
```

Voilà, maintenant vous aurez beau tourner le terminal dans tous les sens, l'application restera toujours orientée de la même manière.

En résumé

- Le fichier Manifest est indispensable à tous les projets Android. C'est lui qui déclarera toute une série d'informations sur votre application.
- Le nœud `<application>` listera les différents composants de votre application ainsi que les services qu'ils offrent.
- Vous pouvez signaler que vous utiliserez des permissions grâce à l'élément `uses-permission` ou que vous en créez grâce à l'élément `permission`.
- Comprendre le cycle de vie d'une activité est essentiel pour construire des activités robustes et ergonomiques.
- Les terminaux peuvent se tenir en mode paysage ou en mode portrait. Vous vous devez de gérer un minimum ce changement de configuration puisqu'au basculement de l'appareil, votre système reconstruira toute votre interface et perdra par conséquent les données que l'utilisateur aurait pu saisir.



Chapitre 14

La communication entre composants

Difficulté :

C'est très bien tout ça, mais on ne sait toujours pas comment lancer une activité depuis une autre activité. C'est ce que nous allons voir dans ce chapitre, et même un peu plus. On va apprendre à manipuler un mécanisme puissant qui permet de faire exécuter certaines actions et de faire circuler des messages entre applications ou à l'intérieur d'une même application. Ainsi, chaque application est censée vivre dans un compartiment cloisonné pour ne pas déranger le système quand elle s'exécute et surtout quand elle plante. À l'aide de ces liens qui lient les compartiments, Android devient un vrai puzzle dont chaque pièce apporte une fonctionnalité qui pourrait fournir son aide à une autre pièce, ou au contraire qui aurait besoin de l'aide d'une autre pièce.

Les agents qui sont chargés de ce mécanisme d'échange s'appellent les `intents`. Ce mécanisme est tellement important qu'Android lui-même l'utilise massivement en interne.

Aspect technique

Un intent est en fait un objet qui contient plusieurs champs, représentés à la figure 14.1.

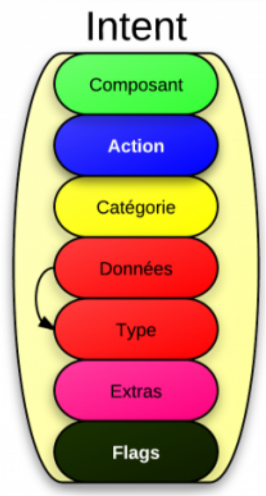


FIGURE 14.1 – Remarquez que le champ « Données » détermine le champ « Type » et que ce n'est pas réciproque

La façon dont sont renseignés ces champs détermine la nature ainsi que les objectifs de l'intent. Ainsi, pour qu'un intent soit dit « explicite », il suffit que son champ composant soit renseigné. Ce champ permet de définir le destinataire de l'intent, celui qui devra le gérer. Ce champ est constitué de deux informations : le *package* où se situe le composant, ainsi que le *nom* du composant. Ainsi, quand l'intent sera exécuté, Android pourra retrouver le composant de destination de manière précise.

À l'opposé des intents explicites se trouvent les intents « implicites ». Dans ce cas de figure, on ne connaît pas de manière précise le destinataire de l'intent, c'est pourquoi on va s'appliquer à renseigner d'autres champs pour laisser Android déterminer qui est capable de réceptionner cet intent. Il faut au moins fournir deux informations essentielles :

- Une action : ce qu'on désire que le destinataire fasse.
- Un ensemble de données : sur quelles données le destinataire doit effectuer son action.

Il existe aussi d'autres informations, pas forcément obligatoires, mais qui ont aussi leur utilité propre le moment venu :

- La catégorie : permet d'apporter des informations supplémentaires sur l'action à exécuter et le type de composant qui devra gérer l'intent.
- Le type : pour indiquer quel est le type des données incluses. Normalement ce type est contenu dans les données, mais en précisant cet attribut vous pouvez désactiver

cette vérification automatique et imposer un type particulier.
- Les extras : pour ajouter du contenu à vos intents afin de les faire circuler entre les composants.
- Les flags : permettent de modifier le comportement de l'intent.

Injecter des données dans un intent

Types standards

Nous avons vu à l'instant que les intents avaient un champ « extra » qui leur permet de contenir des données à véhiculer entre les applications. Un extra est en fait une clé à laquelle on associe une valeur. Pour insérer un extra, c'est facile, il suffit d'utiliser la méthode `Intent putExtra(String key, X value)` avec `key` la clé de l'extra et `value` la valeur associée. Vous voyez que j'ai mis un `X` pour indiquer le type de la valeur — ce n'est pas syntaxiquement exact, je le sais. Je l'utilise juste pour indiquer qu'on peut y mettre un peu n'importe quel type de base, par exemple `int`, `String` ou `double[]`.

Puis vous pouvez récuperer tous les extras d'un intent à l'aide de la méthode `Bundle getExtras()`, auquel cas vos couples clé-valeurs sont contenus dans le `Bundle`. Vous pouvez encore récupérer un extra précis à l'aide de sa clé et de son type en utilisant la méthode `X get{X}Extra(String key, X defaultValue)`, `X` étant le type de l'extra et `defaultValue` la valeur qui sera retournée si la clé passée ne correspond à aucun extra de l'intent. En revanche, pour les types un peu plus complexes tels que les tableaux, on ne peut préciser de valeur par défaut, par conséquent on devra par exemple utiliser la méthode `float[] getFloatArrayExtra(String key)` pour un tableau de `float`.

En règle générale, la clé de l'extra commence par le package duquel provient l'intent.

```
1  // On déclare une constante dans la classe FirstClass
2  public final static String NOMS = "sdz.chapitreTrois.intent.
       examples.NOMS";
3
4  ...
5
6  // Autre part dans le code
7  Intent i = new Intent();
8  String[] noms = new String[] {"Dupont", "Dupond"};
9  i.putExtra(FirstClass.NOMS, noms);
10
11 // Encore autre part
12 String[] noms = i.getStringArrayExtra(FirstClass.NOMS);
```

Il est possible de rajouter *un unique* `Bundle` en extra avec la méthode `Intent putExtras(Bundle extras)` et *un unique* `Intent` avec la méthode `Intent putExtras(Intent extras)`.

Les parcelables

Cependant, `Bundle` ne peut pas prendre tous les objets, comme je vous l'ai expliqué précédemment, il faut qu'ils soient sérialisables. Or, dans le cas d'Android, on considère qu'un objet est sérialisable à partir du moment où il implémente correctement l'interface `Parcelable`. Si on devait entrer dans les détails, sachez qu'un `Parcelable` est un objet qui sera transmis à un `Parcel`, et que l'objectif des `Parcel` est de transmettre des messages entre différents processus du système.

Pour implémenter l'interface `Parcelable`, il faut redéfinir deux méthodes :

- `int describeContents()`, qui permet de définir si vous avez des paramètres spéciaux dans votre `Parcelable`. En ce mois de juillet 2012 (à l'heure où j'écris ces lignes), les seuls objets spéciaux à considérer sont les `FileDescriptor`. Ainsi, si votre objet ne contient pas d'objet de type `FileDescriptor`, vous pouvez renvoyer 0, sinon renvoyez `Parcelable.CONTENT_FILE_DESCRIPTOR`.
- `void writeToParcel(Parcel dest, int flags)`, avec `dest` le `Parcel` dans lequel nous allons insérer les attributs de notre objet et `flags` un entier qui vaut la plupart du temps 0. C'est dans cette classe que nous allons écrire dans le `Parcel` qui transmettra le message.

Les attributs sont à insérer dans le `Parcel` dans l'ordre dans lequel ils sont déclarés dans la classe !

Si on prend l'exemple simple d'un contact dans un répertoire téléphonique :

```
1  import android.os.Parcel;
2  import android.os.Parcelable;
3
4  public class Contact implements Parcelable{
5    private String mNom;
6    private String mPrenom;
7    private int mNumero;
8
9    public Contact(String pNom, String pPrenom, int pNumero) {
10     mNom = pNom;
11     mPrenom = pPrenom;
12     mNumero = pNumero;
13   }
14
15   @Override
16   public int describeContents() {
17     //On renvoie 0, car notre classe ne contient pas de
           FileDescriptor
18     return 0;
19   }
20
21   @Override
22   public void writeToParcel(Parcel dest, int flags) {
```

```
23      // On ajoute les objets dans l'ordre dans lequel on les a d
            éclarés
24      dest.writeString(mNom);
25      dest.writeString(mPrenom);
26      dest.writeInt(mNumero);
27    }
28  }
```

Tous nos attributs sont désormais dans le `Parcel`, on peut transmettre notre objet.

C'est presque fini, cependant, il nous faut encore ajouter un champ statique de type `Parcelable.Creator` et qui s'appellera impérativement « CREATOR », sinon nous serions incapables de reconstruire un objet qui est passé par un `Parcel` :

```
1   public static final Parcelable.Creator<Contact> CREATOR = new
        Parcelable.Creator<Contact>() {
2     @Override
3     public Contact createFromParcel(Parcel source) {
4       return new Contact(source);
5     }
6
7     @Override
8     public Contact[] newArray(int size) {
9       return new Contact[size];
10    }
11  };
12
13  public Contact(Parcel in) {
14    mNom = in.readString();
15    mPrenom = in.readString();
16    mNumero = in.readInt();
17  }
```

Enfin, comme n'importe quel autre objet, on peut l'ajouter dans un intent avec `putExtra` et on peut le récupérer avec `getParcelableExtra`.

```
1   Intent i = new Intent();
2   Contact c = new Contact("Dupont", "Dupond", 06);
3   i.putExtra("sdz.chapitreTrois.intent.examples.CONTACT", c);
4
5   // Autre part dans le code
6
7   Contact c = i.getParcelableExtra("sdz.chapitreTrois.intent.
        examples.CONTACT");
```

Les intents explicites

Créer un intent explicite est très simple puisqu'il suffit de donner un `Context` qui appartienne au package où se trouve la classe de destination :

```
Intent intent = new Intent(Context context, Class<?> cls);
```

Par exemple, si la classe de destination appartient au package du `Context` actuel :

```
Intent intent = new Intent(Activite_de_depart.this,
    Activite_de_destination.class);
```

À noter qu'on aurait aussi pu utiliser la méthode `Intent setClass(Context packageContext, Class<?> cls)` avec `packageContext` un `Context` qui appartient au même package que le composant de destination et `cls` le nom de la classe qui héberge cette activité.

Il existe ensuite deux façons de lancer l'intent, selon qu'on veuille que le composant de destination nous renvoie une réponse ou pas.

Sans retour

Si vous ne vous attendez pas à ce que la nouvelle activité vous renvoie un résultat, alors vous pouvez l'appeler très naturellement avec `void startActivity (Intent intent)` dans votre activité. La nouvelle activité sera indépendante de l'actuelle. Elle entreprendra un cycle d'activité normal, c'est-à-dire en commençant par un `onCreate`.

Voici un exemple tout simple : dans une première activité, vous allez mettre un bouton et vous allez faire en sorte qu'appuyer sur ce bouton lance une seconde activité :

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity {
  public final static String AGE = "sdz.chapitreTrois.intent.
      example.AGE";

  private Button mPasserelle = null;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mPasserelle = (Button) findViewById(R.id.passerelle);

    mPasserelle.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        // Le premier paramètre est le nom de l'activité
            actuelle
        // Le second est le nom de l'activité de destination
```

```
24          Intent secondeActivite = new Intent(MainActivity.this,
                IntentExample.class);
25
26          // On rajoute un extra
27          secondeActivite.putExtra(AGE, 31);
28
29          // Puis on lance l'intent !
30          startActivity(secondeActivite);
31        }
32      });
33    }
34  }
```

La seconde activité ne fera rien de particulier, si ce n'est afficher un layout différent :

```
1  package sdz.chapitreTrois.intent.example;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5
6  public class IntentExample extends Activity {
7    @Override
8    public void onCreate(Bundle savedInstanceState) {
9      super.onCreate(savedInstanceState);
10     setContentView(R.layout.layout_example);
11
12     // On récupère l'intent qui a lancé cette activité
13     Intent i = getIntent();
14
15     // Puis on récupère l'âge donné dans l'autre activité, ou 0
           si cet extra n'est pas dans l'intent
16     int age = i.getIntExtra(MainActivity.AGE, 0);
17
18     // S'il ne s'agit pas de l'âge par défaut
19     if(age != 0)
20       // Traiter l'âge
21       age = 2;
22   }
23 }
```

Enfin, n'oubliez pas de préciser dans le Manifest que vous avez désormais deux activités au sein de votre application :

```
1  <manifest xmlns:android="http://schemas.android.com/apk/res/
       android"
2    package="sdz.chapitreTrois.intent.example"
3    android:versionCode="1"
4    android:versionName="1.0" >
5
6    <uses-sdk
7      android:minSdkVersion="7"
```

```
8       android:targetSdkVersion="7" />
9
10    <application
11      android:icon="@drawable/ic_launcher"
12      android:label="@string/app_name"
13      android:theme="@style/AppTheme" >
14      <activity
15        android:name=".MainActivity"
16        android:label="@string/title_activity_main" >
17        <intent-filter>
18          <action android:name="android.intent.action.MAIN" />
19          <category android:name="android.intent.category.
              LAUNCHER" />
20        </intent-filter>
21      </activity>
22
23      <activity
24        android:name=".IntentExample"
25        android:label="@string/title_example" >
26      </activity>
27    </application>
28
29  </manifest>
```

Ainsi, dès qu'on clique sur le bouton de la première activité, on passe directement à la seconde activité, comme le montre la figure 14.2.

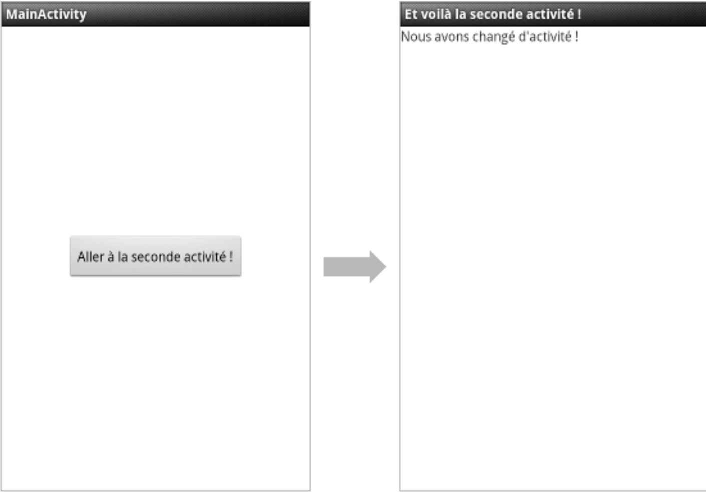


FIGURE 14.2 – En cliquant sur le bouton de la première activité, on passe à la seconde

Avec retour

Cette fois, on veut qu'au retour de l'activité qui vient d'être appelée cette dernière nous renvoie un petit *feedback*. Pour cela, on utilisera la méthode `void startActivityForResult(Intent intent, int requestCode)`, avec `requestCode` un code passé qui permet d'identifier de manière unique un intent.

Ce code doit être supérieur ou égal à 0, sinon Android considérera que vous n'avez pas demandé de résultat.

Quand l'activité appelée s'arrêtera, la première méthode de *callback* appelée dans l'activité précédente sera `void onActivityResult(int requestCode, int resultCode, Intent data)`. On retrouve `requestCode`, qui sera le même code que celui passé dans le `startActivityForResult` et qui permet de repérer quel intent a provoqué l'appel de l'activité dont le cycle vient de s'interrompre. `resultCode` est quant à lui un code renvoyé par l'activité qui indique comment elle s'est terminée (typiquement `Activity.RESULT_OK` si l'activité s'est terminée normalement, ou `Activity.RESULT_CANCELED` s'il y a eu un problème ou qu'aucun code de retour n'a été précisé). Enfin, `intent` est un intent qui contient éventuellement des données.

Par défaut, le code renvoyé par une activité est `Activity.RESULT_CANCELED` de façon que, si l'utilisateur utilise le bouton Retour avant que l'activité ait fini de s'exécuter, vous puissiez savoir que le résultat fourni ne sera pas adapté à vos besoins.

Dans la seconde activité, vous pouvez définir un résultat avec la méthode `void setResult(int resultCode, Intent data)`, ces paramètres étant identiques à ceux décrits ci-dessus.

Ainsi, l'attribut `requestCode` de `void startActivityForResult(Intent intent, int requestCode)` sera similaire au `requestCode` que nous fournira la méthode de *callback* `void onActivityResult(int requestCode, int resultCode, Intent data)`, de manière à pouvoir identifier quel intent est à l'origine de ce retour.

Le code de ce nouvel exemple sera presque similaire à celui de l'exemple précédent, sauf que cette fois la seconde activité proposera à l'utilisateur de cliquer sur deux boutons. Cliquer sur un de ces boutons retournera à l'activité précédente en lui indiquant lequel des deux boutons a été pressé. Ainsi, `MainActivity` ressemble désormais à :

```
1 package sdz.chapitreTrois.intent.example;
2 
3 import android.annotation.SuppressLint;
4 import android.app.Activity;
5 import android.content.Intent;
6 import android.os.Bundle;
7 import android.view.View;
8 import android.widget.Button;
9 import android.widget.Toast;
```

```
10
11 public class MainActivity extends Activity {
12   private Button mPasserelle = null;
13   // L'identifiant de notre requête
14   public final static int CHOOSE_BUTTON_REQUEST = 0;
15   // L'identifiant de la chaîne de caractères qui contient le r
       ésultat de l'intent
16   public final static String BUTTONS = "sdz.chapitreTrois.
       intent.example.Boutons";
17
18   @Override
19   public void onCreate(Bundle savedInstanceState) {
20     super.onCreate(savedInstanceState);
21     setContentView(R.layout.activity_main);
22
23     mPasserelle = (Button) findViewById(R.id.passerelle);
24
25     mPasserelle.setOnClickListener(new View.OnClickListener() {
26       @Override
27       public void onClick(View v) {
28         Intent secondeActivite = new Intent(MainActivity.this,
             IntentExample.class);
29         // On associe l'identifiant à notre intent
30         startActivityForResult(secondeActivite,
             CHOOSE_BUTTON_REQUEST);
31       }
32     });
33   }
34
35   @Override
36   protected void onActivityResult(int requestCode, int
       resultCode, Intent data) {
37     // On vérifie tout d'abord à quel intent on fait référence
         ici à l'aide de notre identifiant
38     if (requestCode == CHOOSE_BUTTON_REQUEST) {
39       // On vérifie aussi que l'opération s'est bien déroulée
40       if (resultCode == RESULT_OK) {
41         // On affiche le bouton qui a été choisi
42         Toast.makeText(this, "Vous avez choisi le bouton " +
             data.getStringExtra(BUTTONS), Toast.LENGTH_SHORT).
             show();
43       }
44     }
45   }
46 }
```

Alors que la seconde activité devient :

```
1 package sdz.chapitreTrois.intent.example;
2
3 import android.app.Activity;
```

```
4  import android.content.Intent;
5  import android.os.Bundle;
6  import android.view.View;
7  import android.widget.Button;
8
9  public class IntentExample extends Activity {
10   private Button mButton1 = null;
11   private Button mButton2 = null;
12
13   @Override
14   public void onCreate(Bundle savedInstanceState) {
15     super.onCreate(savedInstanceState);
16     setContentView(R.layout.layout_example);
17
18     mButton1 = (Button) findViewById(R.id.button1);
19     mButton1.setOnClickListener(new View.OnClickListener() {
20       @Override
21       public void onClick(View v) {
22         Intent result = new Intent();
23         result.putExtra(MainActivity.BUTTONS, "1");
24         setResult(RESULT_OK, result);
25         finish();
26       }
27     });
28
29     mButton2 = (Button) findViewById(R.id.button2);
30     mButton2.setOnClickListener(new View.OnClickListener() {
31       @Override
32       public void onClick(View v) {
33         Intent result = new Intent();
34         result.putExtra(MainActivity.BUTTONS, "2");
35         setResult(RESULT_OK, result);
36         finish();
37       }
38     });
39   }
40 }
```

Et voilà, dès que vous cliquez sur un des boutons, la première activité va lancer un `Toast` qui affichera quel bouton a été pressé, comme le montre la figure 14.3.

Les intents implicites

Ici, on fera en sorte d'envoyer une requête à un destinataire, sans savoir qui il est, et d'ailleurs on s'en fiche tant que le travail qu'on lui demande de faire est effectué. Ainsi, les applications destinataires sont soit fournies par Android, soit par d'autres applications téléchargées sur le Play Store par exemple.

FIGURE 14.3 – Un Toast affiche quel bouton a été pressé

Les données

L'URI

La première chose qu'on va étudier, c'est les données, parce qu'elles sont organisées selon une certaine syntaxe qu'il vous faut connaître. En fait, elles sont formatées à l'aide des URI (Uniform Resource Identifier). Un URI est une chaîne de caractères qui permet d'identifier un endroit. Par exemple sur internet, ou dans le cas d'Android sur le périphérique ou une ressource. Afin d'étudier les URI, on va faire l'analogie avec les adresses URL qui nous permettent d'accéder à des sites internet. En effet, un peu à la manière d'un serveur, nos fournisseurs de contenu vont répondre en fonction de l'URI fournie. De plus, la forme générale d'une URI rappelle fortement les URL. Prenons l'exemple du Site du Zéro avec une URL inventée : `http://www.siteduzero.com/forum/android/aide.html`. On identifie plusieurs parties :

- `http://`
- `www.siteduzero.com - www.siteduzero.com`
- `/forum/android/aide.html`

Les URI se comportent d'une manière un peu similaire. La syntaxe d'un URI peut être analysée de la manière suivante (les parties entre accolades {} sont optionnelles) :

```
1 <schéma> : <information> { ? <requête> } { # <fragment> }
```

- Le `schéma` décrit quelle est la nature de l'information. S'il s'agit d'un numéro de téléphone, alors le schéma sera `tel`, s'il s'agit d'un site internet, alors le schéma sera `http`, etc.
- L'`information` est la donnée en tant que telle. Cette information respecte elle aussi une syntaxe, mais qui dépend du schéma cette fois-ci. Ainsi, pour un numéro de

téléphone, vous pouvez vous contenter d'insérer le numéro `tel:0606060606`, mais pour des coordonnées GPS il faudra séparer la latitude de la longitude à l'aide d'une virgule `geo:123.456789,-12.345678`. Pour un site internet, il s'agit d'un chemin hiérarchique.
- La `requête` permet de fournir une précision par rapport à l'information.
- Le `fragment` permet enfin d'accéder à une sous-partie de l'information.

Pour créer un objet URI, c'est simple, il suffit d'utiliser la méthode statique `Uri Uri.parse(String uri)`. Par exemple, pour envoyer un SMS à une personne, j'utiliserai l'URI :

```
1 Uri sms = Uri.parse("sms:0606060606");
```

Mais je peux aussi indiquer plusieurs destinataires et un corps pour ce message :

```
1 Uri sms = Uri.parse("sms:0606060606,0606060607?body=Salut%20les
      %20potes");
```

Comme vous pouvez le voir, le contenu de la chaîne doit être encodé, sinon vous rencontrerez des problèmes.

Type MIME

Le MIME (Multipurpose Internet Mail Extensions) est un identifiant pour les formats de fichier. Par exemple, il existe un type MIME `text`. Si une donnée est accompagnée du type MIME `text`, alors les données sont du texte. On trouve aussi `audio` et `video` par exemple. Il est ensuite possible de préciser un sous-type afin d'affiner les informations sur les données, par exemple `audio/mp3` et `audio/wav` sont deux types MIME qui indiquent que les données sont sonores, mais aussi de quelle manière elles sont encodées.

Les types MIME que nous venons de voir son standards, c'est-à-dire qu'il y a une organisation qui les a reconnus comme étant légitimes. Mais si vous vouliez créer vos propres types MIME ? Vous n'allez pas demander à l'organisation de les valider, ils ne seront pas d'accord avec vous. C'est pourquoi il existe une petite syntaxe à respecter pour les types personnalisés : `vnd.votre_package.le_type`, ce qui peut donner par exemple `vnd.sdz.chapitreTrois.contact_telephonique`.

Pour être tout à fait exact, sous Android vous ne pourrez jamais que préciser des sous-types, jamais des types.

Pour les intents, ce type peut être décrit de manière implicite dans l'URI (on voit bien par exemple que `sms:0606060606` décrit un numéro de téléphone, il n'est pas nécessaire de le préciser), mais il faudra par moments le décrire de manière explicite. On peut le faire dans le champ `type` d'un intent.

Préciser un type est surtout indispensable quand on doit manipuler des ensembles de données, comme par exemple quand on veut supprimer une ou plusieurs entrées dans le répertoire, car dans ce cas précis il s'agira d'un pointeur vers ces données. Avec

Android, il existe deux manières de manipuler ces ensembles de données, les curseurs (*cursor*) et les fournisseurs de contenus (*content provider*). Ces deux techniques seront étudiées plus tard, par conséquent nous allons nous cantonner aux données simples pour l'instant.

L'action

Une action est une constante qui se trouve dans la classe `Intent` et qui commence toujours par « ACTION_ » suivi d'un verbe (en anglais, bien sûr) de façon à bien faire comprendre qu'il s'agit d'une action. Si vous voulez *voir* quelque chose, on va utiliser l'action `ACTION_VIEW`. Par exemple, si vous utilisez `ACTION_VIEW` sur un numéro de téléphone, alors le numéro de téléphone s'affichera dans le composeur de numéros de téléphone.

Vous pouvez aussi créer vos propres actions. Pour cela, il vaux mieux respecter une syntaxe, qui est de commencer par votre package suivi de `.intent.action.NOM_DE_L_ACTION` :

```
1 public final static String ACTION_PERSO = "sdz.chapitreTrois.
      intent.action.PERSO";
```

Le tableau suivant indique quelques actions natives parmi les plus usitées.

Les actions suivies d'une astérisque sont celles que vous ne pourrez pas utiliser tant que nous n'aurons pas vu les `Content Provider`.

Pour créer un intent qui va ouvrir le composeur téléphonique avec le numéro de téléphone 0606060606, j'adapte mon code précédent en remplaçant le code du bouton par :

```
1 mPasserelle.setOnClickListener(new View.OnClickListener() {
2   @Override
3   public void onClick(View v) {
4     Uri telephone = Uri.parse("tel:0606060606");
5     Intent secondeActivite = new Intent(Intent.ACTION_DIAL,
          telephone);
6     startActivity(secondeActivite);
7   }
8 });
```

Ce qui donne, une fois que j'appuie dessus, la figure 14.4.

La résolution des intents

Quand on lance un `ACTION_VIEW` avec une adresse internet, c'est le navigateur qui se lance, et quand on lance un `ACTION_VIEW` avec un numéro de téléphone, c'est le composeur de numéros qui se lance. Alors, comment Android détermine qui doit répondre à un intent donné ?

Intitulé	**Action**	**Entrée attendue**	**Sortie attendue**
`ACTION_MAIN`	Pour indiquer qu'il s'agit du point d'entrée dans l'application	/	/
`ACTION_DIAL`	Pour ouvrir le composeur de numéros téléphoniques	Un numéro de téléphone semble une bonne idée :-p	/
`ACTION_DELETE*`	Supprimer des données	Un URI vers les données à supprimer	/
`ACTION_EDIT*`	Ouvrir un éditeur adapté pour modifier les données fournies	Un URI vers les données à éditer	/
`ACTION_INSERT*`	Insérer des données	L'URI du répertoire où insérer les données	L'URI des nouvelles données créées
`ACTION_PICK*`	Sélectionner un élément dans un ensemble de données	L'URI qui contient un répertoire de données à partir duquel l'élément sera sélectionné	L'URI de l'élément qui a été sélectionné
`ACTION_SEARCH`	Effectuer une recherche	Le texte à rechercher	/
`ACTION_SENDTO`	Envoyer un message à quelqu'un	La personne à qui envoyer le message	/
`ACTION_VIEW`	Permet de visionner une donnée	Un peu tout. Une adresse e-mail sera visionnée dans l'application pour les e-mails, un numéro de téléphone dans le composeur, etc.	/
`ACTION_WEB_SEARCH`	Effectuer une recherche sur internet	S'il s'agit d'un texte qui commence par « http », le site s'affichera directement, sinon c'est une recherche dans Google qui se fera	/

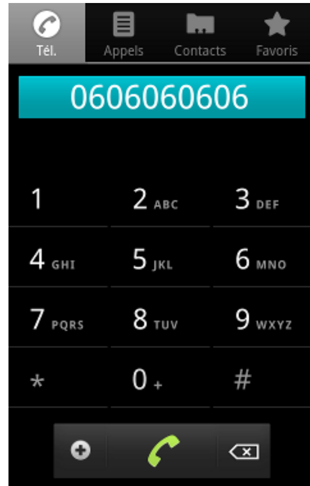


Figure 14.4 – Le composeur téléphonique est lancé avec le numéro souhaité

Ce que va faire Android, c'est qu'il va comparer l'intent à des filtres que nous allons déclarer dans le Manifest et qui signalent que les composants de nos applications peuvent gérer certains intents. Ces filtres sont les nœuds `<intent-filter>`, nous les avons déjà rencontrés et ignorés par le passé. Un composant d'une application doit avoir autant de filtres que de capacités de traitement. S'il peut gérer deux types d'intent, il doit avoir deux filtres.

Le test de conformité entre un intent et un filtre se fait sur trois critères.

L'action

Permet de filtrer en fonction du champ `Action` d'un intent. Il peut y en avoir un ou plus par filtre. Si vous n'en mettez pas, tous vos intents seront recalés. Un intent sera accepté si ce qui se trouve dans son champ `Action` est identique à au moins une des actions du filtre. Et si un intent ne précise pas d'action, alors il sera automatiquement accepté pour ce test.

C'est pour cette raison que les intents explicites sont toujours acceptés, ils n'ont pas de champ `Action`, par conséquent ils passent le test, même si le filtre ne précise aucune action.

```
1 <activity>
2   <intent-filter>
3     <action android:name="android.intent.action.VIEW" />
4     <action android:name="android.intent.action.SENDTO" />
5   </intent-filter>
6 </activity>
```

Cette activité ne pourra intercepter que les intents qui ont dans leur champ action `ACTION_VIEW` et/ou `ACTION_SENDTO`, car *toutes* ses actions sont acceptées par le filtre. Si un intent a pour action `ACTION_VIEW` et `ACTION_SEARCH`, alors il sera recalé, car une de ses actions n'est pas acceptée par le filtre.

La catégorie

Cette fois, il n'est pas indispensable d'avoir une indication de catégorie pour un intent, mais, s'il y en a une ou plusieurs, alors pour passer ce test il faut que *toutes* les catégories de l'intent correspondent à des catégories du filtre. Pour les matheux, on dit qu'il s'agit d'une application « injective » mais pas « surjective ».

On pourrait se dire que par conséquent, si un intent n'a pas de catégorie, alors il passe automatiquement ce test, mais dès qu'un intent est utilisé avec la méthode `startActivity()`, alors on lui ajoute la catégorie `CATEGORY_DEFAULT`. Donc, si vous voulez que votre composant accepte les intents implicites, vous devez rajouter cette catégorie à votre filtre.

Pour les actions et les catégories, la syntaxe est différente entre le Java et le XML. Par exemple, pour l'action `ACTION_VIEW` en Java, on utilisera `android.intent.action.VIEW` et pour la categorie `CATEGORY_DEFAULT` on utilisera `android.intent.category.DEFAULT`. De plus, quand vous créez vos propres actions ou catégories, le mieux est de les préfixer avec le nom de votre package afin de vous assurer qu'elles restent uniques.

```
1 <activity>
2   <intent-filter>
3     <action android:name="android.intent.action.VIEW" />
4     <action android:name="android.intent.action.SEARCH" />
5     <category android:name="android.intent.category.DEFAULT" />
6     <category android:name="com.sdz.intent.category.
          DESEMBROUILLEUR" />
7   </intent-filter>
8 </activity>
```

Il faut ici que l'intent ait pour action `ACTION_VIEW` et/ou `ACTION_SEARCH`. En ce qui concerne les catégories, il doit avoir `CATEGORY_DEFAULT` *et* `CATEGORY_DESEMBROUILLEUR`.

Le tableau suivant indique les principales catégories par défaut fournies par Android.

Les données

Il est possible de préciser plusieurs informations sur les données que cette activité peut traiter. Principalement, on peut préciser le schéma qu'on veut avec `android:scheme`, on peut aussi préciser le type MIME avec `android:mimeType`. Par exemple, si notre application traite des fichiers textes qui proviennent d'internet, on aura besoin du type « texte » et du schéma « internet », ce qui donne :

```
1 <data android:mimeType="text/plain" android:scheme="http" />
2 <data android:mimeType="text/plain" android:scheme="https" />
```

Catégorie	**Description**
`CATEGORY_DEFAULT`	Indique qu'il faut effectuer le traitement par défaut sur les données correspondantes. Concrètement, on l'utilise pour déclarer qu'on accepte que ce composant soit utilisé par des intents implicites.
`CATEGORY_BROWSABLE`	Utilisé pour indiquer qu'une activité peut être appelée sans risque depuis un navigateur web. Ainsi, si un utilisateur clique sur un lien dans votre application, vous promettez que rien de dangereux ne se passera à la suite de l'activation de cet intent.
`CATEGORY_TAB`	Utilisé pour les activités qu'on retrouve dans des onglets.
`CATEGORY_ALTERNATIVE`	Permet de définir une activité comme un traitement alternatif dans le visionnage d'éléments. C'est par exemple intéressant dans les menus, si vous souhaitez proposer à votre utilisateur de regarder telles données de la manière proposée par votre application ou d'une manière que propose une autre application.
`CATEGORY_SELECTED_ALTERNATIVE`	Comme ci-dessus, mais pour des éléments qui ont été sélectionnés, pas seulement pour les voir.
`CATEGORY_LAUNCHER`	Indique que c'est ce composant qui doit s'afficher dans le lanceur d'applications.
`CATEGORY_HOME`	Permet d'indiquer que c'est cette activité qui doit se trouver sur l'écran d'accueil d'Android.
`CATEGORY_PREFERENCE`	Utilisé pour identifier les `PreferenceActivity` (dont nous parlerons au chapitre suivant).

Et il se passe quoi en interne une fois qu'on a lancé un intent ?

Eh bien, il existe plusieurs cas de figure :

– Soit votre recherche est infructueuse et vous avez 0 résultat, auquel cas c'est grave et une `ActivityNotFoundException` sera lancée. Il vous faut donc penser à gérer ce type d'erreurs.
– Si on n'a qu'un résultat, comme dans le cas des intents explicites, alors ce résultat va directement se lancer.
– En revanche, si on a plusieurs réponses possibles, alors le système va demander à l'utilisateur de choisir à l'aide d'une boîte de dialogue. Si l'utilisateur choisit une action par défaut pour un intent, alors à chaque fois que le même intent sera émis ce sera toujours le même composant qui sera sélectionné. D'ailleurs, il peut arriver que ce soit une mauvaise chose parce qu'un même intent ne signifie pas toujours une même intention (ironiquement). Il se peut qu'avec ACTION_SEND, on cherche un jour à envoyer un SMS et un autre jour à envoyer un e-mail, c'est pourquoi il est possible de forcer la main à Android et à obliger l'utilisateur à choisir parmi plusieurs éventualités à l'aide de `Intent createChooser(Intent target, CharSequence titre)`. On peut ainsi insérer l'*intent* à traiter et le *titre* de la boîte de dialogue qui permettra à l'utilisateur de choisir une application.

Dans tous les cas, vous pouvez vérifier si un composant va réagir à un intent de manière programmatique à l'aide du `Package Manager`. Le `Package Manager` est un objet qui vous permet d'obtenir des informations sur les packages qui sont installés sur l'appareil. On y fait appel avec la méthode `PackageManager getPackageManager()` dans n'importe quel composant. Puis on demande à l'intent le nom de l'activité qui va pouvoir le gérer à l'aide de la méthode `ComponentName resolveActivity (PackageManager pm)` :

```
1 Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("tel:
      0606060606"));
2
3 PackageManager manager = getPackageManager();
4
5 ComponentName component = intent.resolveActivity(manager);
6 // On vérifie que component n'est pas null
7 if(component != null)
8   //Alors c'est qu'il y a une activité qui va gérer l'intent
```

Pour aller plus loin : navigation entre des activités

Une application possède en général plusieurs activités. Chaque activité est dédiée à un ensemble cohérent d'actions, mais toujours centrées vers un même objectif. Pour une application qui lit des musiques, il y a une activité pour choisir la musique à écouter, une autre qui présente les contrôles sur les musiques, encore une autre pour paramétrer

l'application, etc.

Je vous avais présenté au tout début du cours la pile des activités. En effet, comme on ne peut avoir qu'une activité visible à la fois, les activités étaient présentées dans une pile où il était possible d'ajouter ou d'enlever un élément au sommet afin de changer l'activité consultée actuellement. C'est bien sûr toujours vrai, à un détail près. Il existe en fait une pile par tâche. On pourrait dire qu'une tâche est une application, mais aussi les activités qui seront lancées par cette application et qui sont extérieures à l'application. Ainsi que les activités qui seront lancées par ces activités extérieures, etc.

Au démarrage de l'application, une nouvelle tâche est créée et l'activité principale occupe la racine de la pile.

Au lancement d'une nouvelle activité, cette dernière est ajoutée au sommet de la pile et acquiert ainsi le focus. L'activité précédente est arrêtée, mais l'état de son interface graphique est conservé. Quand l'utilisateur appuie sur le bouton `Retour`, l'activité actuelle est éjectée de la pile (elle est donc détruite) et l'activité précédente reprend son déroulement normal (avec restauration des éléments de l'interface graphique). S'il n'y a pas d'activité précédente, alors la tâche est tout simplement détruite.

Dans une pile, on ne manipule jamais que le sommet. Ainsi, si l'utilisateur appuie sur un bouton de l'activité 1 pour aller à l'activité 2, puis appuie sur un bouton de l'activité 2 pour aller dans l'activité 1, alors une nouvelle instance de l'activité 1 sera créée, comme le montre la figure 14.5.

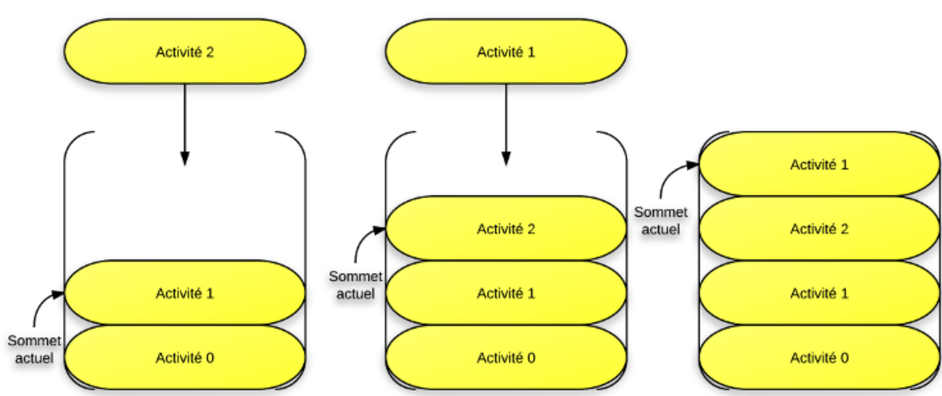


FIGURE 14.5 – On passe de l'activité 1 à l'activité 2, puis de l'activité 2 à l'activité 1, ce qui fait qu'on a deux différentes instances de l'activité 1 !

Pour changer ce comportement, il est possible de manipuler l'**affinité** d'une activité. Cette affinité est un attribut qui indique avec quelle tâche elle préfère travailler. Toutes les activités qui ont une affinité avec une même tâche se lanceront dans cette tâche-là.

Elle permet surtout de déterminer à quelle tâche une activité sera apparentée ainsi que la tâche qui va accueillir l'activité quand elle est lancée avec le flag FLAG_ACTIVITY_NEW_TASK. Par défaut, toutes les activités d'une même application ont la même affinité. En effet, si vous ne précisez pas d'affinité, alors cet attribut prendra la valeur du package de l'application.

Ce comportement est celui qui est préférable la plupart du temps. Cependant, il peut arriver que vous ayez besoin d'agir autrement, auquel cas il y a deux façons de faire.

Modifier l'activité dans le Manifest

Il existe six attributs que nous n'avons pas encore vus et qui permettent de changer la façon dont Android réagit à la navigation.

`android:taskAffinity`

Cet attribut permet de préciser avec quelle tâche cette activité possède une affinité. Exemple :

```
1 <activity android:taskAffinity="sdz.chapitreTrois.intent.
    exemple.tacheUn" />
2 <activity android:taskAffinity="sdz.chapitreTrois.intent.
    exemple.tacheDeux" />
```

`android:allowTaskReparenting`

Est-ce que l'activité peut se déconnecter d'une tâche dans laquelle elle a commencé à travailler pour aller vers une autre tâche avec laquelle elle a une affinité ?

Par exemple, dans le cas d'une application pour lire les SMS, si le SMS contient un lien, alors cliquer dessus lancera une activité qui permettra d'afficher la page web désignée par le lien. Si on appuie sur le bouton `Retour`, on revient à la lecture du SMS. En revanche, avec cet attribut, l'activité lancée sera liée à la tâche du navigateur et non plus du client SMS.

La valeur par défaut est `false`.

`android:launchMode`

Définit comment l'application devra être lancée dans une tâche. Il existe deux modes : soit l'activité peut être instanciée plusieurs fois dans la même tâche, soit elle est toujours présente de manière unique.

Dans le premier mode, il existe deux valeurs possibles :

- `standard` est le mode par défaut, dès qu'on lance une activité une nouvelle instance est créée dans la tâche. Les différentes instances peuvent aussi appartenir à plusieurs

tâches.
- Avec `singleTop`, si une instance de l'activité existe déjà au sommet de la tâche actuelle, alors le système redirigera l'intent vers cette instance au lieu de créer une nouvelle instance. Le retour dans l'activité se fera à travers la méthode de *callback* `void onNewIntent(Intent intent)`.

Le second mode n'est pas recommandé et doit être utilisé uniquement dans des cas précis. Surtout, on ne l'utilise que si l'activité est celle de lancement de l'application. Il peut prendre deux valeurs :

- Avec `singleTask`, le système crée l'activité à la racine d'une nouvelle tâche. Cependant, si une instance de l'activité existe déjà, alors on ouvrira plutôt cette instance-là.
- Enfin avec `singleInstance`, à chaque fois on crée une nouvelle tâche dont l'activité sera la racine.

`android:clearTaskOnLaunch`

Est-ce que toutes les activités doivent être enlevées de la tâche — à l'exception de la racine — quand on relance la tâche depuis l'écran de démarrage ? Ainsi, dès que l'utilisateur relance l'application, il retournera à l'activité d'accueil, sinon il retournera dans la dernière activité qu'il consultait.

La valeur par défaut est `false`.

`android:alwaysRetainTaskState`

Est-ce que l'état de la tâche dans laquelle se trouve l'activité — et dont elle est la racine — doit être maintenu par le système ?

Typiquement, une tâche est détruite si elle n'est pas active et que l'utilisateur ne la consulte pas pendant un certain temps. Cependant, dans certains cas, comme dans le cas d'un navigateur web avec des onglets, l'utilisateur sera bien content de récupérer les onglets qui étaient ouverts.

La valeur par défaut est `false`.

`android:finishOnTaskLaunch`

Est-ce que, s'il existe déjà une instance de cette activité, il faut la fermer dès qu'une nouvelle instance est demandée ?

La valeur par défaut est `false`.

Avec les intents

Il est aussi possible de modifier l'association par défaut d'une activité à une tâche à l'aide des flags contenus dans les intents. On peut rajouter un flag à un intent avec la méthode `Intent addFlags(int flags)`.

Il existe trois flags principaux :

- `FLAG_ACTIVITY_NEW_TASK` permet de lancer l'activité dans une nouvelle tâche, sauf si l'activité existe déjà dans une tâche. C'est l'équivalent du mode `singleTask`.
- `FLAG_ACTIVITY_SINGLE_TOP` est un équivalent de `singleTop`. On lancera l'activité dans une nouvelle tâche, quelques soient les circonstances.
- `FLAG_ACTIVITY_CLEAR_TOP` permet de faire en sorte que, si l'activité est déjà lancée dans la tâche actuelle, alors au lieu de lancer une nouvelle instance de cette activité toutes les autres activités qui se trouvent au-dessus d'elle seront fermées et l'intent sera délivré à l'activité (souvent utilisé avec `FLAG_ACTIVITY_NEW_TASK`). Quand on utilise ces deux flags ensemble, ils permettent de localiser une activité qui existait déjà dans une autre tâche et de la mettre dans une position où elle pourra répondre à l'intent.

Pour aller plus loin : diffuser des intents

On a vu avec les intents comment dire « Je veux que vous traitiez cela, alors que quelqu'un le fasse pour moi s'il vous plaît ». Ici on va voir comment dire « Cet évènement vient de se dérouler, je préviens juste, si cela intéresse quelqu'un ». C'est donc la différence entre « Je viens de recevoir un SMS, je cherche un composant qui pourra permettre à l'utilisateur de lui répondre » et « Je viens de recevoir un SMS, ça intéresse une application de le gérer ? ». Il s'agit ici uniquement de notifications, pas de demandes. Concrètement, le mécanisme normal des intents est visible pour l'utilisateur, alors que celui que nous allons étudier est totalement transparent pour lui.

Nous utiliserons toujours des intents, sauf qu'ils seront anonymes et diffusés à tout le système. Ce type d'intent est très utilisé au niveau du système pour transmettre des informations, comme par exemple l'état de la batterie ou du réseau. Ces intents particuliers s'appellent des *broadcast intents*. On utilise encore une fois un système de filtrage pour déterminer qui peut recevoir l'intent, mais c'est la façon dont nous allons recevoir les messages qui est un peu spéciale.

La création des broadcast intents est similaire à celle des intents classiques, sauf que vous allez les envoyer avec la méthode `void sendBroadcast(Intent intent)`. De cette manière, l'intent ne sera reçu que par les *broadcast receivers*, qui sont des classes qui dérivent de la classe `BroadcastReceiver`. De plus, quand vous allez déclarer ce composant dans votre Manifest, il faudra que vous annonciez qu'il s'agit d'un broadcast receiver :

```
1 <receiver android:name="CoucouReceiver">
2   <intent-filter>
3     <action android:name="sdz.chapitreTrois.intent.action.
        coucou" />
4   </intent-filter>
5 </receiver>
```

Il vous faudra alors redéfinir la méthode de *callback* `void onReceive (Context context, Intent intent)` qui est lancée dès qu'on reçoit un broadcast intent. C'est dans cette

classe qu'on gérera le message reçu.

Par exemple, si j'ai un intent qui transmet à tout le système le nom de l'utilisateur :

```
1  public class CoucouReceiver extends BroadcastReceiver {
2    private static final String NOM_USER = "sdz.chapitreTrois.
         intent.extra.NOM";
3
4    // Déclenché dès qu'on reçoit un broadcast intent qui réponde
         aux filtres déclarés dans le Manifest
5    @Override
6    public void onReceive(Context context, Intent intent) {
7      // On vérifie qu'il s'agit du bon intent
8      if(intent.getAction().equals("ACTION_COUCOU")) {
9        // On récupère le nom de l'utilisateur
10       String nom = intent.getExtra(NOM_USER);
11       Toast.makeText(context, "Coucou " + nom + " !", Toast.
             LENGTH_LONG).show();
12     }
13   }
14 }
```

Un broascast receiver déclaré de cette manière sera disponible tout le temps, même quand l'application n'est pas lancée, mais ne sera viable que pendant la durée d'exécution de sa méthode `onReceive`. Ainsi, ne vous attendez pas à retrouver votre receiver si vous lancez un thread, une boîte de dialogue ou un autre composant d'une application à partir de lui.

De plus, il ne s'exécutera pas en parallèle de votre application, mais bien de manière séquentielle (dans le même thread, donc), ce qui signifie que, si vous effectuez de gros calculs qui prennent du temps, les performances de votre application pourraient s'en trouver affectées.

Mais il est aussi possible de déclarer un broadcast receiver de manière dynamique, directement dans le code. Cette technique est surtout utilisée pour gérer les évènements de l'interface graphique.

Pour procéder, vous devrez créer une classe qui dérive de `BroadcastReceiver`, mais sans l'enregistrer dans le Manifest. Ensuite, vous pouvez lui rajouter des lois de filtrage avec la classe `IntentFilter`, puis vous pouvez l'enregistrer dans l'activité voulue avec la méthode `Intent registerReceiver(BroadcastReceiver receiver, IntentFilter filter)` et surtout, quand vous n'en n'aurez plus besoin, il faudra la désactiver avec `void unregisterReceiver(BroadcastReceiver receiver)`.

Ainsi, si on veut recevoir nos broadcast intents pour dire coucou à l'utilisateur, mais uniquement quand l'application se lance et qu'elle n'est pas en pause, on fait :

```
1  import android.app.Activity;
2  import android.content.IntentFilter;
3  import android.os.Bundle;
4
5  public class CoucouActivity extends Activity {
```

```
  private static final String COUCOU = "sdz.chapitreTrois.
      intent.action.coucou";
  private IntentFilter filtre = null;
  private CoucouReceiver receiver = null;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    filtre = new IntentFilter(COUCOU);
    receiver = new CoucouReceiver();
  }

  @Override
  public void onResume() {
    super.onResume();
    registerReceiver(receiver, filtre);
  }

  /** Si vous déclarez votre receiver dans le onResume, n'
      oubliez pas qu'il faut l'arrêter dans le onPause **/
  @Override
  public void onPause() {
    super.onPause();
    unregisterReceiver(receiver);
  }
}
```

De plus, il existe quelques messages diffusés par le système de manière native et que vous pouvez écouter, comme par exemple `ACTION_CAMERA_BUTTON` qui est lancé dès que l'utilisateur appuie sur le bouton de l'appareil photo.

Sécurité

N'importe quelle application peut envoyer des broadcast intents à votre receiver, ce qui est une faiblesse au niveau sécurité. Vous pouvez aussi faire en sorte que votre receiver déclaré dans le Manifest ne soit accessible qu'à l'intérieur de votre application en lui ajoutant l'attribut `android:exported="false"` :

```
<receiver android:name="CoucouReceiver"
  android:exported="false">
  <intent-filter>
    <action android:name="sdz.chapitreTrois.intent.action.
        coucou" />
  </intent-filter>
</receiver>
```

Notez que cet attribut est disponible pour tous les composants.

De plus, quand vous envoyez un broadcast intent, toutes les applications peuvent le recevoir. Afin de déterminer qui peut recevoir un broadcast intent, il suffit de lui ajouter une permission à l'aide de la méthode `void sendBroadcast (Intent intent, String receiverPermission)`, avec `receiverPermission` une permission que vous aurez déterminée. Ainsi, seuls les receivers qui déclarent cette permission pourront recevoir ces broadcast intents :

```
private String COUCOU_BROADCAST = "sdz.chapitreTrois.permission
    .COUCOU_BROADCAST";

...

sendBroadcast(i, COUCOU_BROADCAST);
```

Puis dans le Manifest, il suffit de rajouter :

```
<uses-permission android:name="sdz.chapitreTrois.permission.
    COUCOU_BROADCAST"/>
```

En résumé

- Les intents sont des objets permettant d'envoyer des messages entre vos activités, voire entre vos applications. Ils peuvent, selon vos préférences, contenir un certain nombre d'informations qu'il sera possible d'exploiter dans une autre activité.
- En général, les données contenues dans un intent sont assez limitées mais il est possible de partager une classe entière si vous étendez la classe `Parcelable` et que vous implémentez toutes les méthodes nécessaires à son fonctionnement.
- Les intents explicites sont destinés à se rendre à une activité très précise. Vous pourriez également définir que vous attendez un retour suite à cet appel via la méthode `void startActivityForResult(Intent intent, int requestCode)`.
- Les intents implicites sont destinés à demander à une activité, sans que l'on sache laquelle, de traiter votre message en désignant le type d'action souhaité et les données à traiter.
- Définir un nœud `<intent-filter>` dans le nœud d'une `<activity>` de votre fichier Manifest vous permettra de filtrer vos activités en fonction du champ d'action de vos intents.
- Nous avons vu qu'Android gère nos activités *via* une pile LIFO. Pour changer ce comportement, il est possible de manipuler l'affinité d'une activité. Cette affinité est un attribut qui indique avec quelle tâche elle préfère travailler.
- Les broadcast intents diffusent des intents à travers tout le système pour transmettre des informations de manière publique à qui veut. Cela se met en place grâce à un nœud `<receiver>` filtré par un nœud `<intent-filter>`.

Chapitre 15

Le stockage de données

Difficulté :

La plupart de nos applications auront besoin de stocker des données à un moment ou à un autre. La couleur préférée de l'utilisateur, sa configuration réseau ou encore des fichiers téléchargés sur internet. En fonction de ce que vous souhaitez faire et de ce que vous souhaitez enregistrer, Android vous fournit plusieurs méthodes pour sauvegarder des informations. Il existe deux solutions qui permettent d'enregistrer des données de manière rapide et flexible, si on exclut les bases de données. Nous aborderons tout d'abord les **préférences partagées**, qui permettent d'associer à un identifiant une valeur. Le couple ainsi créé permet de retenir les différentes options que l'utilisateur souhaiterait conserver ou l'état de l'interface graphique. Ces valeurs pourront être partagées entre plusieurs composants. Encore mieux, Android propose un ensemble d'outils qui permettront de faciliter grandement le travail et d'unifier les interfaces graphiques des activités dédiées à la sauvegarde des préférences des utilisateurs. Il peut aussi arriver qu'on ait besoin d'écrire ou de lire des fichiers qui sont stockés sur le terminal ou sur un périphérique externe.

Préférences partagées

Utile voire indispensable pour un grand nombre d'applications, pouvoir enregistrer les paramètres des utilisateurs leur permettra de paramétrer de manière minutieuse vos applications de manière à ce qu'ils obtiennent le rendu qui convienne le mieux à leurs exigences.

Les données partagées

Le point de départ de la manipulation des préférences partagées est la classe `SharedPreferences`. Elle possède des méthodes permettant d'enregistrer et récupérer des paires de type identifiant-valeur pour les types de données primitifs, comme les entiers ou les chaînes de caractères. L'avantage réel étant bien sûr que ces données sont conservées même si l'application est arrêtée ou tuée. Ces préférences sont de plus accessibles depuis plusieurs composants au sein d'une même application.

Il existe trois façons d'avoir accès aux `SharedPreferences` :

- La plus simple est d'utiliser la méthode statique `SharedPreferences PreferenceManager.getDefaultSharedPreferences(Context context)`.
- Si vous désirez utiliser un fichier standard par activité, alors vous pourrez utiliser la méthode `SharedPreferences getPreferences(int mode)`.
- En revanche, si vous avez besoin de plusieurs fichiers que vous identifierez par leur nom, alors utilisez `SharedPreferences getSharedPreferences (String name, int mode)` où `name` sera le nom du fichier.

En ce qui concerne le second paramètre, `mode`, il peut prendre trois valeurs :

- `Context.MODE_PRIVATE`, pour que le fichier créé ne soit accessible que par l'application qui l'a créé.
- `Context.MODE_WORLD_READABLE`, pour que le fichier créé puisse être lu par n'importe quelle application.
- `Context.MODE_WORLD_WRITEABLE`, pour que le fichier créé puisse être lu *et* modifié par n'importe quelle application.

Petit détail, appeler `SharedPreferences PreferenceManager.getDefaultSharedPreferences(Context context)` revient à appeler `SharedPreferences getPreferences(MODE_PRIVATE)` et utiliser `SharedPreferences getPreferences(int mode)` revient à utiliser `SharedPreferences getSharedPreferences (NOM_PAR_DEFAUT, mode)` avec `NOM_PAR_DEFAUT` un nom généré en fonction du package de l'application.

Afin d'ajouter ou de modifier des couples dans un `SharedPreferences`, il faut utiliser un objet de type `SharedPreference.Editor`. Il est possible de récupérer cet objet en utilisant la méthode `SharedPreferences.Editor edit()` sur un `SharedPreferences`.

Si vous souhaitez ajouter des informations, utilisez une méthode du genre `SharedPreferences.Editor putX(String key, X value)` avec `X` le type de l'objet, `key` l'identifiant et `value` la valeur associée. Il vous faut ensuite impérativement valider vos changements avec la méthode `boolean commit()`.

Les préférences partagées ne fonctionnent qu'avec les objets de type `boolean`, `float`, `int`, `long` et `String`.

Par exemple, pour conserver la couleur préférée de l'utilisateur, il n'est pas possible d'utiliser la classe `Color` puisque seuls les types de base sont acceptés, alors on pourrait conserver la valeur de la couleur sous la forme d'une chaîne de caractères :

```
1 public final static String FAVORITE_COLOR = "fav color";
2
3 ...
4
5 SharedPreferences preferences = PreferenceManager.
    getDefaultSharedPreferences(this);
6 SharedPreferences.Editor editor = preferences.edit();
7 editor.putString(FAVORITE_COLOR, "FFABB4");
8 editor.commit();
```

Pour récupérer une valeur, on peut utiliser la méthode `X getX(String key, X defValue)` avec `X` le type de l'objet désiré, `key` l'identifiant de votre valeur et `defValue` une valeur que vous souhaitez voir retournée au cas où il n'y ait pas de valeur associée à `key` :

```
1 // On veut la chaîne de caractères d'identifiant FAVORITE_COLOR
2 // Si on ne trouve pas cette valeur, on veut rendre "FFFFFF"
3 String couleur = preferences.getString(FAVORITE_COLOR, "FFFFFF"
    );
```

Si vous souhaitez supprimer une préférence, vous pouvez le faire avec `SharedPreferences.Editor removeString(String key)`, ou, pour radicalement supprimer toutes les préférences, il existe aussi `SharedPreferences.Editor clear()`.

Enfin, si vous voulez récupérer toutes les données contenues dans les préférences, vous pouvez utiliser la méthode `Map<String,?> getAll()`.

Des préférences prêtes à l'emploi

Pour enregistrer vos préférences, vous pouvez très bien proposer une activité qui permet d'insérer différents paramètres (voir figure 15.1). Si vous voulez développer vous-mêmes l'activité, grand bien vous fasse, ça fera des révisions, mais sachez qu'il existe un framework pour vous aider. Vous en avez sûrement déjà vus dans d'autres applications. C'est d'ailleurs un énorme avantage d'avoir toujours un écran similaire entre les applications pour la sélection des préférences.

Ce type d'activités s'appelle les « `PreferenceActivity` ». Un plus indéniable ici est que chaque couple identifiant/valeur est créé automatiquement et sera récupéré automatiquement, d'où un gain de temps énorme dans la programmation. La création se fait en plusieurs étapes, nous allons voir la première, qui consiste à établir une interface graphique en XML.

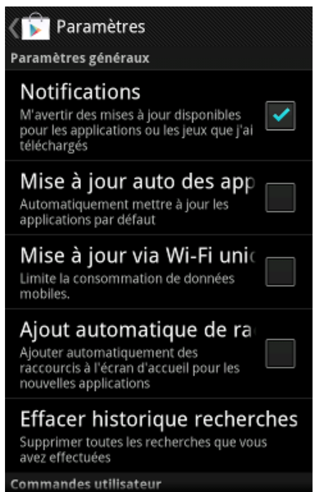


FIGURE 15.1 – L'activité permettant de choisir des paramètres pour le Play Store

Étape 1 : en XML

La racine de ce fichier doit être un `PreferenceScreen`.

Comme ce n'est pas vraiment un layout, on le définit souvent dans `/xml/preference.xml`.

Tout d'abord, il est possible de désigner des catégories de préférences. Une pour les préférences destinées à internet par exemple, une autre pour les préférences esthétiques, etc. On peut ajouter des préférences avec le nœud `PreferenceCategory`. Ce nœud est un layout, il peut donc contenir d'autre vues. Il ne peut prendre qu'un seul attribut, `android:title`, pour préciser le texte qu'il affichera.

Ainsi le code suivant :

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <PreferenceScreen xmlns:android="http://schemas.android.com/apk
       /res/android" >
3    <PreferenceCategory android:title="Réseau">
4
5    </PreferenceCategory>
6    <PreferenceCategory android:title="Luminosité">
7
8    </PreferenceCategory>
9    <PreferenceCategory android:title="Couleurs">
10
11   </PreferenceCategory>
12 </PreferenceScreen>
```

. . . donne le résultat visible à la figure 15.2.

Figure 15.2 – Le code en image

Nous avons nos catégories, il nous faut maintenant insérer des préférences ! Ces trois vues ont cinq attributs en commun :

- `android:key` est l'identifiant de la préférence partagée. C'est un attribut *indispensable*, ne l'oubliez *jamais*.
- `android:title` est le titre principal de la préférence.
- `android:summary` est un texte secondaire qui peut être plus long et qui explique mieux ce que veut dire cette préférence.
- Utilisez `android:dependency`, si vous voulez lier votre préférence à une autre activité. Il faut y insérer l'identifiant `android:key` de la préférence dont on dépend.
- `android:defaultValue` est la valeur par défaut de cette préférence.

Il existe au moins trois types de préférences, la première étant une case à cocher avec `CheckBoxPreference`, avec `true` ou `false` comme valeur (soit la case est cochée, soit elle ne l'est pas).

À la place du résumé standard, vous pouvez déclarer un résumé qui ne s'affiche que quand la case est cochée, `android:summaryOn`, ou uniquement quand la case est décochée, `android:summaryOff`.

```
1 <CheckBoxPreference
2   android:key="checkBoxPref"
3   android:title="Titre"
4   android:summaryOn="Résumé quand sélectionné"
5   android:summaryOff="Résumé quand pas sélectionné"
6   android:defaultValue="true"/>
```

Ce qui donne la figure 15.3.

Figure 15.3 – Regardez la première préférence, la case est cochée par défaut et c'est le résumé associé qui est affiché

Le deuxième type de préférences consiste à permettre à l'utilisateur d'insérer du texte

avec EditTextPreference, qui ouvre une boîte de dialogue contenant un EditText permettant à l'utilisateur d'insérer du texte. On retrouve des attributs qui vous rappelleront fortement le chapitre sur les boîtes de dialogue. Par exemple, android:dialogTitle permet de définir le texte de la boîte de dialogue, alors que android:negativeButtonText et android:positiveButtonText permettent respectivement de définir le texte du bouton à droite et celui du bouton à gauche dans la boîte de dialogue.

```
1 <EditTextPreference
2   android:key="editTextPref"
3   android:dialogTitle="Titre de la boîte"
4   android:positiveButtonText="Je valide !"
5   android:negativeButtonText="Je valide pas !"
6   android:title="Titre"
7   android:summary="Résumé"
8   android:dependency="checkBoxPref" />
```

Ce qui donne la figure 15.4.

FIGURE 15.4 – Le code en image

De plus, comme vous avez pu le voir, ce paramètre est lié à la CheckBoxPreference précédente par l'attribut android:dependency="checkBoxPref", ce qui fait qu'il ne sera accessible que si la case à cocher de checkBoxPref est activée, comme à la figure 15.5.

FIGURE 15.5 – Le paramètre n'est accessible que si la case est cochée

De plus, comme nous l'avons fait avec les autres boîtes de dialogue, il est possible d'imposer un layout à l'aide de l'attribut android:dialogLayout.

Le troisième type de préférences est un choix dans une liste d'options avec ListPreference. Dans cette préférence, on différencie ce qui est affiché de ce qui est réel. Pratique pour

traduire son application en plusieurs langues ! Il est possible d'utiliser les attributs `android:dialogTitle`, `android:negativeButtonText` et `android:positiveButtonText`. Les données de la liste que lira l'utilisateur sont à présenter dans l'attribut `android:entries`, alors que les données qui seront enregistrées sont à indiquer dans l'attribut `android:entryValues`. La manière la plus facile de remplir ces attributs se fait à l'aide d'une ressource de type `array`, par exemple pour la liste des couleurs :

```
1  <resources>
2    <array name="liste_couleurs_fr">
3      <item>Bleu</item>
4      <item>Rouge</item>
5      <item>Vert</item>
6    </array>
7    <array name="liste_couleurs">
8      <item>blue</item>
9      <item>red</item>
10     <item>green</item>
11   </array>
12 </resources>
```

Qu'on peut ensuite fournir aux attributs susnommés :

```
1  <ListPreference
2    android:key="listPref"
3    android:dialogTitle="Choisissez une couleur"
4    android:entries="@array/liste_couleurs_fr"
5    android:entryValues="@array/liste_couleurs"
6    android:title="Choisir couleur" />
```

Ce qui donne la figure 15.6.

FIGURE 15.6 – Le code en image

On a sélectionné « Vert », ce qui signifie que la valeur enregistrée sera `green`.

Étape 2 : dans le Manifest

Pour recevoir cette nouvelle interface graphique, nous avons besoin d'une activité. Il nous faut donc la déclarer dans le Manifest si on veut pouvoir y accéder avec les intents. Cette activité sera déclarée comme n'importe quelle activité :

```
1 <activity
2   android:name=".PreferenceActivityExample"
3   android:label="@string/
        title_activity_preference_activity_example" >
4 </activity>
```

Étape 3 : en Java

Notre activité sera en fait de type `PreferenceActivity`. On peut la traiter comme une activité classique, sauf qu'au lieu de lui assigner une interface graphique avec `setContentView`, on utilise `void addPreferencesFromResource(int preferencesResId)` en lui assignant notre layout :

```
1 public void onCreate(Bundle savedInstanceState) {
2   super.onCreate(savedInstanceState);
3   addPreferencesFromResource(R.xml.preference);
4 }
```

Manipulation des fichiers

On a déjà vu comment manipuler certains fichiers précis à l'aide des ressources, mais il existe aussi des cas de figure où il faudra prendre en compte d'autres fichiers, par exemple dans le cas d'un téléchargement ou de l'exploration de fichiers sur la carte SD d'un téléphone. En théorie, vous ne serez pas très dépaysés ici puisqu'on manipule en majorité les mêmes méthodes qu'en Java. Il existe bien entendu quand même des différences.

Il y a deux manières d'utiliser les fichiers : soit sur la mémoire interne du périphérique à un endroit bien spécifique, soit sur une mémoire externe (par exemple une carte SD). Dans tous les cas, on part toujours du `Context` pour manipuler des fichiers.

Rappels sur l'écriture et la lecture de fichiers

Ce n'est pas un sujet forcément évident en Java puisqu'il existe beaucoup de méthodes qui permettent d'écrire et de lire des fichiers en fonction de la situation.

Le cas le plus simple est de manipuler des flux d'octets, ce qui nécessite des objets de type `FileInputStream` pour lire un fichier et `FileOutputStream` pour écrire dans un fichier. La lecture s'effectue avec la méthode `int read()` et on écrit dans un fichier

avec `void write(byte[] b)`. Voici un programme très simple qui lit dans un fichier puis écrit dans un autre fichier :

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyBytes {
  public static void main(String[] args) throws IOException {

    FileInputStream in = null;
    FileOutputStream out = null;

    try {
      in = new FileInputStream("entree.txt");
      out = new FileOutputStream("sortie.txt");
      int octet;

      // La méthode read renvoie -1 dès qu'il n'y a plus rien à
          lire
      while ((octet = in.read()) != -1) {
        out.write(octet);
      }
      if (in != null)
        in.close();

      if (out != null)
        out.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

En interne

L'avantage ici est que la présence des fichiers dépend de la présence de l'application. Par conséquent, les fichiers seront supprimés si l'utilisateur désinstalle l'activité. En revanche, comme la mémoire interne du téléphone risque d'être limitée, on évite en général de placer les plus gros fichiers de cette manière.

Afin de récupérer un `FileOutputStream` qui pointera vers le bon répertoire, il suffit d'utiliser la méthode `FileOutputStream openFileOutput (String name, int mode)` avec `name` le nom du fichier et `mode` le mode dans lequel ouvrir le fichier. Eh oui, encore une fois, il existe plusieurs méthodes pour ouvrir un fichier :

– `MODE_PRIVATE` permet de créer (ou de remplacer, d'ailleurs) un fichier qui sera utilisé uniquement par l'application.

- MODE_WORLD_READABLE pour créer un fichier que même d'autres applications pourront lire.
- MODE_WORLD_WRITABLE pour créer un fichier où même d'autres applications pourront lire et écrire.
- MODE_APPEND pour écrire à la fin d'un fichier préexistant, au lieu de créer un fichier.

Par exemple, pour écrire mon pseudo dans un fichier, je ferai :

```
FileOutputStream output = null;
String userName = "Apollidore";

try {
  output = openFileOutput(PRENOM, MODE_PRIVATE);
  output.write(userName.getBytes());
  if(output != null)
    output.close();
} catch (FileNotFoundException e) {
  e.printStackTrace();
} catch (IOException e) {
  e.printStackTrace();
}
```

De manière analogue, on peut retrouver un fichier dans lequel lire à l'aide de la méthode `FileInputStream openFileInput (String name)`.

N'essayez pas d'insérer des « / » ou des « \ » pour mettre vos fichiers dans un autre répertoire, sinon les méthodes renverront une exception.

Ensuite, il existe quelques méthodes qui permettent de manipuler les fichiers au sein de cet emplacement interne, afin d'avoir un peu plus de contrôle. Déjà, pour retrouver cet emplacement, il suffit d'utiliser la méthode `File getFilesDir()`. Pour supprimer un fichier, on peut faire appel à `boolean deleteFile(String name)` et pour récupérer une liste des fichiers créés par l'application, on emploie `String[] fileList()`.

Travailler avec le cache

Les fichiers normaux ne sont supprimés que si quelqu'un le fait, que ce soit vous ou l'utilisateur. *A contrario*, les fichiers sauvegardés avec le cache peuvent aussi être supprimés par le système d'exploitation afin de libérer de l'espace. C'est un avantage, pour les fichiers qu'on ne veut garder que temporairement.

Pour écrire dans le cache, il suffit d'utiliser la méthode `File getCacheDir()` pour récupérer le répertoire à manipuler. De manière générale, on évite d'utiliser trop d'espace dans le cache, il s'agit vraiment d'un espace temporaire de stockage pour petits fichiers.

Ne vous attendez pas forcément à ce qu'Android supprime les fichiers, il ne le fera que quand il en aura besoin, il n'y a pas de manière de prédire ce comportement.

En externe

Le problème avec le stockage externe, c'est qu'il n'existe aucune garantie que vos fichiers soient présents. L'utilisateur pourra les avoir supprimés ou avoir enlevé le périphérique de son emplacement. Cependant, cette fois la taille disponible de stockage est au rendez-vous ! Enfin, quand nous parlons de périphérique externe, nous parlons principalement d'une carte SD, d'une clé USB... ou encore d'un ordinateur !

Pour écrire sur un périphérique externe, il vous faudra ajouter la permission `WRITE_EXTERNAL_STORAGE`. Pour ce faire, il faut rajouter la ligne suivante à votre Manifest : `<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />`.

Tout d'abord, pour vérifier que vous avez bien accès à la mémoire externe, vous pouvez utiliser la méthode statique `String Environment.getExternalStorageState()`. La chaîne de caractères retournée peut correspondre à plusieurs constantes, dont la plus importante reste `Environment.MEDIA_MOUNTED` pour savoir si le périphérique est bien monté et peut être lu (pour un périphérique bien monté mais qui ne peut pas être lu, on utilisera `Environment.MEDIA_MOUNTED_READ_ONLY`) :

```
1 if(Environment.MEDIA_MOUNTED.equals(Environment.
      getExternalStorageState()))
2   // Le périphérique est bien monté
3 else
4   // Le périphérique n'est pas bien monté ou on ne peut écrire
        dessus
```

Vous trouverez d'autres statuts à utiliser sur la documentation.

▷ Autres statuts
Code web : 304026

Afin d'obtenir la racine des fichiers du périphérique externe, vous pouvez utiliser la méthode statique `File Environment.getExternalStorageDirectory()`. Cependant, il est conseillé d'écrire des fichiers uniquement à un emplacement précis : `/Android/data/<votre_package>/files/`. En effet, les fichiers qui se trouvent à cet emplacement seront automatiquement supprimés dès que l'utilisateur effacera votre application.

Partager des fichiers

Il arrive aussi que votre utilisateur veuille partager la musique qu'il vient de concevoir avec d'autres applications du téléphone, pour la mettre en sonnerie par exemple. Ce sont des fichiers qui ne sont pas spécifiques à votre application ou que l'utilisateur ne souhaitera pas supprimer à la désinstallation de l'application. On va donc faire en sorte de sauvegarder ces fichiers à des endroits spécifiques. Une petite sélection de répertoires : pour la musique on mettra les fichiers dans `/Music/`, pour les téléchargements divers on utilisera `/Download/` et pour les sonneries de téléphone on utilisera

/Ringtones/.

Application

Énoncé

Très simple, on va faire en sorte d'écrire votre pseudo dans deux fichiers : un en stockage interne, l'autre en stockage externe. N'oubliez pas de vérifier qu'il est possible d'écrire sur le support externe !

Détails techniques

Il existe une constante qui indique que le périphérique est en lecture seule (et que par conséquent il est impossible d'écrire dessus), c'est la constante `Environment.MEDIA_MOUNTED_READ_ONLY`.

Si un fichier n'existe pas, vous pouvez le créer avec `boolean createNewFile()`.

Ma solution

```
1  import java.io.File;
2  import java.io.FileInputStream;
3  import java.io.FileNotFoundException;
4  import java.io.FileOutputStream;
5  import java.io.IOException;
6
7  import android.app.Activity;
8  import android.os.Bundle;
9  import android.os.Environment;
10 import android.view.View;
11 import android.widget.Button;
12 import android.widget.Toast;
13
14 public class MainActivity extends Activity {
15   private String PRENOM = "prenom.txt";
16   private String userName = "Apollidore";
17   private File mFile = null;
18
19   private Button mWrite = null;
20   private Button mRead = null;
21
22   @Override
23   public void onCreate(Bundle savedInstanceState) {
24     super.onCreate(savedInstanceState);
25     setContentView(R.layout.activity_main);
26
27     // On crée un fichier qui correspond à l'emplacement exté
          rieur
```

```
28      mFile = new File(Environment.getExternalStorageDirectory().
            getPath() + "/Android/data/ " + getPackageName() + "/
            files/" + PRENOM);
29
30      mWrite = (Button) findViewById(R.id.write);
31      mWrite.setOnClickListener(new View.OnClickListener() {
32
33        public void onClick(View pView) {
34          try {
35            // Flux interne
36            FileOutputStream output = openFileOutput(PRENOM,
                  MODE_PRIVATE);
37
38            // On écrit dans le flux interne
39            output.write(userName.getBytes());
40
41            if(output != null)
42              output.close();
43
44            // Si le fichier est lisible et qu'on peut écrire
                  dedans
45            if(Environment.MEDIA_MOUNTED.equals(Environment.
                  getExternalStorageState())
46                && !Environment.MEDIA_MOUNTED_READ_ONLY.equals(
                      Environment.getExternalStorageState())) {
47              // On crée un nouveau fichier. Si le fichier existe
                    déjà, il ne sera pas créé
48              mFile.createNewFile();
49              output = new FileOutputStream(mFile);
50              output.write(userName.getBytes());
51              if(output != null)
52                output.close();
53            }
54          } catch (FileNotFoundException e) {
55            e.printStackTrace();
56          } catch (IOException e) {
57            e.printStackTrace();
58          }
59        }
60      });
61
62      mRead = (Button) findViewById(R.id.read);
63      mRead.setOnClickListener(new View.OnClickListener() {
64
65        public void onClick(View pView) {
66          try {
67            FileInputStream input = openFileInput(PRENOM);
68            int value;
69            // On utilise un StringBuffer pour construire la chaî
                  ne au fur et à mesure
```

```
70              StringBuffer lu = new StringBuffer();
71              // On lit les caractères les uns après les autres
72              while((value = input.read()) != -1) {
73                // On écrit dans le fichier le caractère lu
74                lu.append((char)value);
75              }
76              Toast.makeText(MainActivity.this, "Interne : " + lu.
                    toString(), Toast.LENGTH_SHORT).show();
77              if(input != null)
78                input.close();
79
80              if(Environment.MEDIA_MOUNTED.equals(Environment.
                    getExternalStorageState())) {
81                lu = new StringBuffer();
82                input = new FileInputStream(mFile);
83                while((value = input.read()) != -1)
84                  lu.append((char)value);
85
86                Toast.makeText(MainActivity.this, "Externe : " + lu
                      .toString(), Toast.LENGTH_SHORT).show();
87                if(input != null)
88                  input.close();
89              }
90
91            } catch (FileNotFoundException e) {
92              e.printStackTrace();
93            } catch (IOException e) {
94              e.printStackTrace();
95            }
96          }
97        });
98      }
99    }
```

En résumé

- Il est possible d'enregistrer les préférences de l'utilisateur avec la classe `SharedPreferences`.
- Pour permettre à l'utilisateur de sélectionner ses préférences, on peut définir une `PreferenceActivity` qui facilite le processus. On peut ainsi insérer des `CheckBox`, des `EditText`, etc.
- Il est possible d'enregistrer des données dans des fichiers facilement, comme en Java. Cependant, on trouve deux endroits accessibles : en interne (sur un emplacement mémoire réservé à l'application sur le téléphone) ou en externe (sur un emplacement mémoire amovible, comme par exemple une carte SD).
- Enregistrer des données sur le cache permet de sauvegarder des fichiers temporairement.

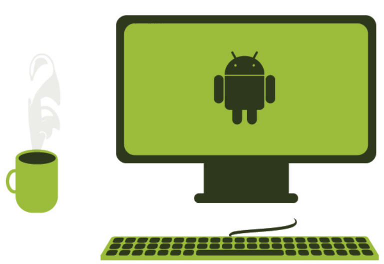
Chapitre 16

TP : un explorateur de fichiers

Difficulté :

Petit à petit, on se rapproche d'un contenu qui pourrait s'apparenter à celui des applications professionnelles. Bien entendu, il nous reste du chemin à parcourir, mais on commence à vraiment voir comment fonctionne Android !

Afin de symboliser notre entrée dans les entrailles du système, on va s'affairer ici à déambuler dans ses méandres. Notre objectif : créer un petit explorateur qui permettra de naviguer entre les fichiers contenus dans le terminal et faire en sorte de pouvoir exécuter certains de ces fichiers.



Objectifs

Contenu d'un répertoire

L'activité principale affiche le contenu du répertoire dans lequel on se situe. Afin de différencier rapidement les fichiers des répertoires, ces derniers seront représentés avec une couleur différente. La figure 16.1 vous donne un avant-goût de ce que l'on obtiendra.

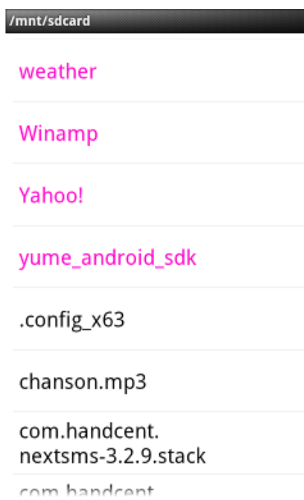


FIGURE 16.1 – Le dernier répertoire que contient le répertoire courant est « yume_android_sdk »

Notez aussi que le titre de l'activité change en fonction du répertoire dans lequel on se trouve. On voit sur la figure précédente que je me trouve dans le répertoire `sdcard`, lui-même situé dans `mnt`.

Navigation entre les répertoires

Si on clique sur un répertoire dans la liste, alors notre explorateur va entrer dedans et afficher la nouvelle liste des fichiers et répertoires. De plus, si l'utilisateur utilise le bouton `Retour Arrière`, alors il reviendra au répertoire parent du répertoire actuel. En revanche, si on se trouve à la racine de tous les répertoires, alors appuyer deux fois sur `Retour Arrière` fait sortir de l'application.

Préférences

Il faudra un menu qui permet d'ouvrir les préférences et où il sera possible de changer la couleur d'affichage des répertoires, comme à la figure 16.2.

FIGURE 16.2 – L'application contiendra un menu de préférences

Cliquer sur cette option ouvre une boîte de dialogue qui permet de sélectionner la couleur voulue, comme le montre la figure 16.3.

FIGURE 16.3 – Il sera possible de modifier la couleur d'affichage des répertoires

Action sur les fichiers

Cliquer sur un fichier fait en sorte de rechercher une application qui pourra le lire. Faire un clic long ouvre un menu contextuel qui permet soit de lancer le fichier comme avec un clic normal, soit de supprimer le fichier, ainsi que le montre la figure 16.4.

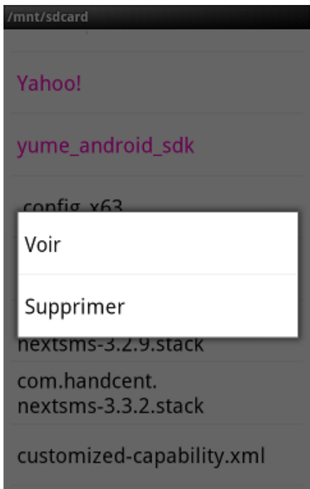


FIGURE 16.4 – Il est possible d'ouvrir ou de supprimer un fichier

Bien sûr, faire un clic long sur un répertoire ne propose pas d'exécuter ce dernier (on pourrait envisager de proposer de l'ouvrir, j'ai opté pour supprimer directement l'option).

Spécifications techniques

Activité principale

Un nouveau genre d'activité

La première chose à faire est de vérifier qu'il est possible de lire la carte SD avec les méthodes vues aux chapitres précédents. S'il est bien possible de lire la carte, alors on affiche la liste des fichiers du répertoire, ce qui se fera dans une `ListView`. Cependant, comme notre mise en page sera uniquement constituée d'une liste, nous allons procéder différemment par rapport à d'habitude. Au lieu d'avoir une activité qui affiche un layout qui contient une `ListView`, on va remplacer notre `Activity` par une `ListActivity`. Comme l'indique le nom, une `ListActivity` est une activité qui est principalement utilisée pour afficher une `ListView`. Comme il s'agit d'une classe qui dérive de `Activity`, il faut la traiter comme une activité normale, si ce n'est que vous n'avez pas besoin de préciser un layout avec `void setContentView (View view)`, puisqu'on sait qu'il n'y a qu'une liste dans la mise en page. Elle sera alors ajoutée automatiquement.

Il est possible de récupérer la `ListView` qu'affiche la `ListActivity` à l'aide de la méthode `ListView getListView ()`. Cette `ListView` est une `ListView` tout à fait banale que vous pouvez traiter comme celles vues dans le cours.

Adaptateur personnalisé

On associera les items à la liste à l'aide d'un adaptateur personnalisé. En effet, c'est la seule solution pour avoir deux couleurs dans les éléments de la liste. On n'oubliera pas d'optimiser cet adaptateur afin d'avoir une liste fluide. Ensuite, on voudra que les éléments soient triés de la manière suivante :

- Les répertoires en premier, les fichiers en second.
- Dans chacune de ces catégories, les éléments sont triés dans l'ordre alphabétique sans tenir compte de la casse.

Pour cela, on pourra utiliser la méthode `void sort (Comparator<? super T> comparator)` qui permet de trier des éléments en fonction de règles qu'on lui passe en paramètres. Ces règles implémentent l'interface `Comparator` de manière à pouvoir définir *comment* seront triés les objets. Votre implémentation de cette interface devra redéfinir la méthode `int compare(T lhs, T rhs)` dont l'objectif est de dire qui est le plus grand entre `lhs` et `rsh`. Si `lhs` est plus grand que `rhs`, on renvoie un entier supérieur à 0, si `lhs` est plus petit que `rhs`, on renvoie un entier inférieur à 0, et s'ils sont égaux, on renvoie 0. Vous devrez vérifier que cette méthode respecte la logique suivante :

- `compare(a,a)` renvoie 0 pour tout `a` parce que `a==a`.
- `compare(a,b)` renvoie l'opposé de `compare(b,a)` pour toutes les paires (a,b) (par exemple, si `a > b`, alors `compare(a,b)` renvoie un entier supérieur à 0 et `compare(b,a)` un entier inférieur à 0).
- Si `compare(a,b) > 0` et `compare(b,c) > 0`, alors `compare(a,c) > 0` quelque que soit la combinaison (`a`, `b`, `c`).

Je comprends que ce soit un peu compliqué à comprendre, alors voici un exemple qui trie les entiers :

```
1  import java.util.Comparator;
2  
3  public class EntierComparator implements Comparator<Integer> {
4    @Override
5    public int compare(Integer lhs, Integer rhs) {
6    // Si lhs est supérieur à rsh, alors on retourne 1
7    if(lhs > rhs)
8      return 1;
9    // Si lhs est inférieur à rsh, alors on retourne -1
10   if(lhs < rhs)
11     return -1;
12   // Si lhs est égal à rsh, alors on retourne 0
13   return 0;
14   }
15 }
```

Ensuite, dans le code, on peut l'utiliser pour trier un tableau d'entiers :

```
1  // Voici un tableau avec des entiers dans le mauvais ordre
2  Integer[] tableau = {0, -1, 5, 10, 9, 5, -10, 8, 21, 132};
3  
```

```
// On convertit le tableau en liste
List<Integer> entiers = new ArrayList<Integer>(Arrays.asList(
    tableau));

// On écrit tous les entiers dans le Logcat, ils sont dans le d
    ésordre !
for(Integer i : entiers)
  Log.d("Avant le tri", Integer.toString(i));

// On utilise une méthode qui va trier les éléments de la liste
Collections.sort(entiers, new EntierComparator());

// Désormais, les entiers seront triés !
for(Integer i : entiers)
  Log.d("Après le tri", Integer.toString(i));

//La liste contient désormais {-10, -1, 0, 5, 5, 8, 9, 10, 21,
    132}
```

Préférences

Nous n'avons qu'une préférence ici, qui chez moi a pour identifiant `repertoireColorPref` et qui contient la couleur dans laquelle nous souhaitons afficher les répertoires.

Comme il n'existe pas de vue qui permette de choisir une couleur, on va utiliser une vue développée par Google dans ses échantillons et qui n'est pas incluse dans le code d'Android. Tout ce qu'il faut faire, c'est créer un fichier Java qui s'appelle `ColorPickerView` et d'y insérer le code suivant :

```
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.ColorMatrix;
import android.graphics.Paint;
import android.graphics.RectF;
import android.graphics.Shader;
import android.graphics.SweepGradient;
import android.view.MotionEvent;
import android.view.View;

public class ColorPickerView extends View {
  public interface OnColorChangedListener {
    void colorChanged(int color);
  }

  private Paint mPaint;
  private Paint mCenterPaint;
  private final int[] mColors;
  private OnColorChangedListener mListener;
```

```
21
22     ColorPickerView(Context c, OnColorChangedListener l, int
           color) {
23       super(c);
24       mListener = l;
25       mColors = new int[] {0xFFFF0000, 0xFFFF00FF, 0xFF0000FF,
             0xFF00FFFF, 0xFF00FF00, 0xFFFFFF00, 0xFFFF0000};
26       Shader s = new SweepGradient(0, 0, mColors, null);
27
28       mPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
29       mPaint.setShader(s);
30       mPaint.setStyle(Paint.Style.STROKE);
31       mPaint.setStrokeWidth(32);
32
33       mCenterPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
34       mCenterPaint.setColor(color);
35       mCenterPaint.setStrokeWidth(5);
36     }
37
38     private boolean mTrackingCenter;
39     private boolean mHighlightCenter;
40
41     @Override
42     protected void onDraw(Canvas canvas) {
43       int centerX = getRootView().getWidth()/2 - (int)(mPaint.
             getStrokeWidth()/2);
44       float r = CENTER_X - mPaint.getStrokeWidth()*0.5f;
45
46       canvas.translate(centerX, CENTER_Y);
47
48       canvas.drawOval(new RectF(-r, -r, r, r), mPaint);
49       canvas.drawCircle(0, 0, CENTER_RADIUS, mCenterPaint);
50
51       if (mTrackingCenter) {
52         int c = mCenterPaint.getColor();
53         mCenterPaint.setStyle(Paint.Style.STROKE);
54
55         if (mHighlightCenter) {
56           mCenterPaint.setAlpha(0xFF);
57         } else {
58           mCenterPaint.setAlpha(0x80);
59         }
60         canvas.drawCircle(0, 0, CENTER_RADIUS + mCenterPaint.
               getStrokeWidth(), mCenterPaint);
61
62         mCenterPaint.setStyle(Paint.Style.FILL);
63         mCenterPaint.setColor(c);
64       }
65     }
66
```

```
67  @Override
68  protected void onMeasure(int widthMeasureSpec, int
        heightMeasureSpec) {
69    setMeasuredDimension(getRootView().getWidth(), CENTER_Y*2);
70  }
71
72  private static final int CENTER_X = 100;
73  private static final int CENTER_Y = 100;
74  private static final int CENTER_RADIUS = 32;
75
76  private int floatToByte(float x) {
77    int n = java.lang.Math.round(x);
78    return n;
79  }
80  private int pinToByte(int n) {
81    if (n < 0) {
82      n = 0;
83    } else if (n > 255) {
84      n = 255;
85    }
86    return n;
87  }
88
89  private int ave(int s, int d, float p) {
90    return s + java.lang.Math.round(p * (d - s));
91  }
92
93  private int interpColor(int colors[], float unit) {
94    if (unit <= 0) {
95      return colors[0];
96    }
97    if (unit >= 1) {
98      return colors[colors.length - 1];
99    }
100
101   float p = unit * (colors.length - 1);
102   int i = (int)p;
103   p -= i;
104
105   int c0 = colors[i];
106   int c1 = colors[i+1];
107   int a = ave(Color.alpha(c0), Color.alpha(c1), p);
108   int r = ave(Color.red(c0), Color.red(c1), p);
109   int g = ave(Color.green(c0), Color.green(c1), p);
110   int b = ave(Color.blue(c0), Color.blue(c1), p);
111
112   return Color.argb(a, r, g, b);
113 }
114
115 private int rotateColor(int color, float rad) {
```

```
116       float deg = rad * 180 / 3.1415927f;
117       int r = Color.red(color);
118       int g = Color.green(color);
119       int b = Color.blue(color);
120
121       ColorMatrix cm = new ColorMatrix();
122       ColorMatrix tmp = new ColorMatrix();
123
124       cm.setRGB2YUV();
125       tmp.setRotate(0, deg);
126       cm.postConcat(tmp);
127       tmp.setYUV2RGB();
128       cm.postConcat(tmp);
129
130       final float[] a = cm.getArray();
131
132       int ir = floatToByte(a[0] * r +  a[1] * g +  a[2] * b);
133       int ig = floatToByte(a[5] * r +  a[6] * g +  a[7] * b);
134       int ib = floatToByte(a[10] * r + a[11] * g + a[12] * b);
135
136       return Color.argb(Color.alpha(color), pinToByte(ir),
              pinToByte(ig), pinToByte(ib));
137     }
138
139     private static final float PI = 3.1415926f;
140
141     @Override
142     public boolean onTouchEvent(MotionEvent event) {
143       float x = event.getX() - CENTER_X;
144       float y = event.getY() - CENTER_Y;
145       boolean inCenter = java.lang.Math.sqrt(x*x + y*y) <=
              CENTER_RADIUS;
146
147       switch (event.getAction()) {
148         case MotionEvent.ACTION_DOWN:
149           mTrackingCenter = inCenter;
150           if (inCenter) {
151             mHighlightCenter = true;
152             invalidate();
153             break;
154           }
155         case MotionEvent.ACTION_MOVE:
156           if (mTrackingCenter) {
157             if (mHighlightCenter != inCenter) {
158               mHighlightCenter = inCenter;
159               invalidate();
160             }
161           } else {
162             float angle = (float)java.lang.Math.atan2(y, x);
163
```

```
164             float unit = angle/(2*PI);
165             if (unit < 0) {
166               unit += 1;
167             }
168             mCenterPaint.setColor(interpColor(mColors, unit));
169             invalidate();
170           }
171           break;
172         case MotionEvent.ACTION_UP:
173           mListener.colorChanged(mCenterPaint.getColor());
174           if (mTrackingCenter) {
175             mTrackingCenter = false;
176             invalidate();
177           }
178           break;
179       }
180       return true;
181     }
182 }
```

Vous pouvez copier ce code grâce au code web suivant :

▷ Copier ce code
Code web : 775031

Ce n'est pas grave si vous ne comprenez pas ce code compliqué, il permet juste d'afficher le joli rond de couleur et de sélectionner une couleur. En fait, la vue contient un listener qui s'appelle `OnColorChangedListener`. Ce listener se déclenche dès que l'utilisateur choisit une couleur. Afin de créer un objet de type `ColorPickerView`, on doit utiliser le constructeur `ColorPickerView(Context c, OnColorChangedListener listener, int color)` avec `listener` le listener qui sera déclenché dès qu'une couleur est choisie et `color` la couleur qui sera choisie par défaut au lancement de la vue.

Notre préférence, elle, sera une boîte de dialogue qui affichera ce `ColorPickerView`. Comme il s'agira d'une boîte de dialogue qui permettra de choisir une préférence, elle dérivera de `DialogPreference`.

Au moment de la construction de la boîte de dialogue, la méthode de *callback* `void onPrepareDialogBuilder(Builder builder)` est appelée, comme pour toutes les `AlertDialog`. On utilise `builder` pour construire la boîte, il est d'ailleurs facile d'y insérer une vue à l'aide de la méthode `AlertDialog.Builder setView(View view)`.

Notre préférence a un attribut de type `int` qui permet de retenir la couleur que choisit l'utilisateur. Elle peut avoir un attribut de type `OnColorChangedListener` ou implémenter elle-même `OnColorChangedListener`, dans tous les cas cette implémentation implique de redéfinir la fonction `void colorChanged(int color)` avec `color` la couleur qui a été choisie. Dès que l'utilisateur choisit une couleur, on change notre attribut pour désigner cette nouvelle couleur.

On n'enregistrera la bonne couleur qu'à la fermeture de la boîte de dialogue, celle-ci étant marquée par l'appel à la méthode `void onDialogClosed(boolean positiveResult)` avec `positiveResult` qui vaut `true` si l'utilisateur a cliqué sur `OK`.

Réagir au changement de préférence

Dès que l'utilisateur change de couleur, il faudrait que ce changement se répercute immédiatement sur l'affichage des répertoires. Il nous faut donc détecter les changements de configuration. Pour cela, on va utiliser l'interface `OnSharedPreferenceChangeListener`. Cette interface fait appel à la méthode de *callback* `void onSharedPreferenceChanged( SharedPreferences sharedPreferences, String key)` dès qu'un changement de préférence arrive, avec `sharedPreferences` l'ensemble des préférences et `key` la clé de la préférence qui vient d'être modifiée. On peut indiquer à `SharedPreferences` qu'on souhaite ajouter un listener à l'aide de la méthode `void registerOnSharedPreference ChangeListener (SharedPreferences.OnSharedPreference ChangeListener listener)`.

Options

Ouvrir le menu d'options ne permet d'accéder qu'à une option. Cliquer sur celle-ci enclenche un intent explicite qui ouvrira la `PreferenceActivity`.

Navigation

Il est recommandé de conserver un `File` qui représente le répertoire courant. On peut savoir si un fichier est un répertoire avec la méthode `boolean isDirectory()` et, s'il s'agit d'un répertoire, on peut voir la liste des fichiers qu'il contient avec `File[] listFiles()`.

Pour effectuer des retours en arrière, il faut détecter la pression du bouton adéquat. À chaque fois qu'on presse un bouton, la méthode de *callback* `boolean onKeyDown(int keyCode, KeyEvent event)` est lancée, avec `keyCode` un code qui représente le bouton pressé et `event` l'évènement qui s'est produit. Le code du bouton `Retour arrière` est `KeyEvent.KEYCODE_BACK`.

Il existe deux cas pour un retour en arrière :

- Soit on ne se trouve pas à la racine de la hiérarchie de fichier, auquel cas on peut revenir en arrière dans cette hiérarchie. Il faut passer au répertoire parent du répertoire actuel et ce répertoire peut se récupérer avec la méthode `File getParentFile()`.
- Soit on se trouve à la racine et il n'est pas possible de faire un retour en arrière. En ce cas, on propose à l'utilisateur de quitter l'application avec la méthode de `Context` que vous connaissez déjà, `void finish()`.

Visualiser un fichier

Nous allons utiliser des intents implicites qui auront pour action `ACTION_VIEW`. Le problème est de savoir comment associer un type et une donnée à un intent, depuis un fichier. Pour la donnée, il existe une méthode statique de la classe `Uri` qui permet d'obtenir l'URI d'un fichier : `Uri.fromFile(File file)`. Pour le type, c'est plus dé-

licat. Il faudra détecter l'extension du fichier pour associer un type qui corresponde. Par exemple, pour un fichier `.mp3`, on indiquera le type MIME `audio/mp3`. Enfin, si on veut moins s'embêter, on peut aussi passer le type MIME `audio/*` pour chaque fichier audio.

Pour rajouter une donnée *et* un type en même temps à un intent, on utilise la méthode `void setDataAndType(Uri data, String type)`, car, si on utilise la méthode `void setData(Uri)`, alors le champ `type` de l'intent est supprimé, et si on utilise `void setType(String)`, alors le champ `data` de l'intent est supprimé. Pour récupérer l'extension d'un fichier, il suffit de récupérer son nom avec `String getName()`, puis de récupérer une partie de ce nom : toute la partie qui se trouve après le point qui représente l'extension :

```
1 fichier.getName().substring(fichier.getName().indexOf(".") + 1)
```

`int indexOf(String str)` va trouver l'endroit où se trouve la première instance de `str` dans la chaîne de caractères, alors que `String substring(int beginIndex)` va extraire la sous-chaîne de caractères qui se situe à partir de `beginIndex` jusqu'à la fin de cette chaîne. Donc, si le fichier s'appelle `chanson.mp3`, la position du point est 7 (puisqu'on commence à 0), on prend donc la sous-chaîne à partir du caractère 8 jusqu'à la fin, ce qui donne « `mp3` ». C'est la même chose que si on avait fait :

```
1 "musique.mp3".subSequence(8, "musique.mp3".length())
```

N'oubliez pas de gérer le cas où vous n'avez pas d'activité qui puisse intercepter votre intent.

Ma solution

Interface graphique

Facile, il n'y en a pas ! Comme notre activité est constituée uniquement d'une `ListView`, pas besoin de lui attribuer une interface graphique avec `setContentView`.

Choisir une couleur avec `ColorPickerPreferenceDialog`

Tout le raisonnement a déjà été expliqué dans les spécifications techniques :

```
1 public class ColorPickerPreferenceDialog extends
      DialogPreference implements OnColorChangedListener{
2   private int mColor = 0;
3
4   public ColorPickerPreferenceDialog(Context context,
        AttributeSet attrs) {
5     super(context, attrs);
```

```
  }

  /**
   * Déclenché dès qu'on ferme la boîte de dialogue
  */
  protected void onDialogClosed(boolean positiveResult) {
    // Si l'utilisateur a cliqué sur « OK »
    if (positiveResult) {
      persistInt(mColor);
    // Ou getSharedPreferences().edit().putInt(getKey(), mColor
        ).commit();
    }
    super.onDialogClosed(positiveResult);
  }

  /**
  * Pour construire la boîte de dialogue
  */
  protected void onPrepareDialogBuilder(Builder builder) {
    // On récupère l'ancienne couleur ou la couleur par défaut
    int oldColor = getSharedPreferences().getInt(getKey(),
        Color.BLACK);
    // On insère la vue dans la boîte de dialogue
    builder.setView(new ColorPickerView(getContext(), this,
        oldColor));

    super.onPrepareDialogBuilder(builder);
  }

  /**
  * Déclenché à chaque fois que l'utilisateur choisit une
     couleur
  */
  public void colorChanged(int color) {
    mColor = color;
  }
}
```

Il faut ensuite ajouter cette boîte de dialogue dans le fichier XML des préférences :

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk
    /res/android" >
  <PreferenceCategory android:title="@string/couleurs_pref" >
    <sdz.chapitreTrois.explorateur.ColorPickerPreferenceDialog
      android:key="repertoireColorPref"
      android:title="Répertoires"
      android:summary="Choisir une couleur des répertoires"
      android:dialogTitle="Couleur des répertoires" />
  </PreferenceCategory>
</PreferenceScreen>
```

Il suffit ensuite de déclarer l'activité dans le Manifest :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sdz.chapitreTrois.explorateur"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="7"
    android:targetSdkVersion="7" />

  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name=".ExplorateurActivity"
      android:label="@string/title_activity_explorateur" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity
      android:name=".ExploreurPreference"
      android:label="@string/title_activity_exploreur_preference" >
    </activity>
  </application>
</manifest>
```

. . . puis de créer l'activité :

```
public class ExploreurPreference extends PreferenceActivity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.preference);
  }
}
```

L'activité principale

Attributs

Voici les différents attributs que j'utilise :

```
/**
```

```
2	 * Représente le texte qui s'affiche quand la liste est vide
3	 */
4	private TextView mEmpty = null;
5	
6	/**
7	 * La liste qui contient nos fichiers et répertoires
8	 */
9	private ListView mList = null;
10	
11	/**
12	 * Notre adaptateur personnalisé qui lie les fichiers à la
	    liste
13	 */
14	private FileAdapter mAdapter = null;
15	
16	/**
17	 * Représente le répertoire actuel
18	 */
19	private File mCurrentFile = null;
20	
21	/**
22	 * Couleur voulue pour les répertoires
23	 */
24	private int mColor = 0;
25	
26	/**
27	 * Indique si l'utilisateur est à la racine ou pas
28	 * Pour savoir s'il veut quitter
29	 */
30	private boolean mCountdown = false;
31	
32	/**
33	 * Les préférences partagées de cette application
34	 */
35	private SharedPreferences mPrefs = null;
```

Comme je fais implémenter `OnSharedPreferenceChangeListener` à mon activité, je dois redéfinir la méthode de *callback* :

```
1	/**
2	 * Se déclenche dès qu'une préférence a changé
3	 */
4	public void onSharedPreferenceChanged(SharedPreferences
	   sharedPreferences, String key) {
5	  mColor = sharedPreferences.getInt("repertoireColorPref",
	      Color.BLACK);
6	  mAdapter.notifyDataSetInvalidated();
7	}
```

L'adaptateur

J'utilise un `Adapter` que j'ai créé moi-même afin d'avoir des items de la liste de différentes couleurs :

```
/**
 * L'adaptateur spécifique à nos fichiers
 */

private class FileAdapter extends ArrayAdapter<File> {
  /**
   * Permet de comparer deux fichiers
   *
   */
  private class FileComparator implements Comparator<File> {
    public int compare(File lhs, File rhs) {
      // Si lhs est un répertoire et pas l'autre, il est plus
          petit
      if(lhs.isDirectory() && rhs.isFile())
        return -1;
      // Dans le cas inverse, il est plus grand
      if(lhs.isFile() && rhs.isDirectory())
        return 1;

      // Enfin, on ordonne en fonction de l'ordre alphabétique
          sans tenir compte de la casse
      return lhs.getName().compareToIgnoreCase(rhs.getName());
    }
  }

  public FileAdapter(Context context, int textViewResourceId,
      List<File> objects) {
    super(context, textViewResourceId, objects);
    mInflater = LayoutInflater.from(context);
  }

  private LayoutInflater mInflater = null;

  /**
   * Construit la vue en fonction de l'item
   */
  public View getView(int position, View convertView, ViewGroup
       parent) {
    TextView vue = null;

    if(convertView != null)
      // On recycle
      vue = (TextView) convertView;
    else
      // On inflate
```

```
42          vue = (TextView) mInflater.inflate(android.R.layout.
               simple_list_item_1 , null);
43
44        File item = getItem(position);
45        //Si c'est un répertoire , on choisit la couleur dans les pr
             éférences
46        if(item.isDirectory())
47          vue.setTextColor(mColor);
48        else
49          // Sinon , c'est du noir
50          vue.setTextColor(Color.BLACK);
51
52        vue.setText(item.getName());
53        return vue;
54      }
55
56      /**
57       * Pour trier rapidement les éléments de l'adaptateur
58       */
59      public void sort () {
60        super.sort(new FileComparator());
61      }
62    }
```

Méthodes secondaires

Ensuite, j'ai une méthode qui permet de vider l'adaptateur :

```
1   /**
2    * On enlève tous les éléments de la liste
3    */
4
5   public void setEmpty() {
6     // Si l'adaptateur n'est pas vide...
7     if(!mAdapter.isEmpty())
8       // Alors on le vide !
9       mAdapter.clear();
10  }
```

J'ai aussi développé une méthode qui me permet de passer d'un répertoire à l'autre :

```
1   /**
2    * Utilisé pour naviguer entre les répertoires
3    * @param pFile le nouveau répertoire dans lequel aller
4    */
5
6   public void updateDirectory(File pFile) {
7     // On change le titre de l'activité
8     setTitle(pFile.getAbsolutePath());
9
```

```java
    // L'utilisateur ne souhaite plus sortir de l'application
    mCountdown = false;

    // On change le répertoire actuel
    mCurrentFile = pFile;
    // On vide les répertoires actuels
    setEmpty();

    // On récupère la liste des fichiers du nouveau répertoire
    File[] fichiers = mCurrentFile.listFiles();

    // Si le répertoire n'est pas vide...
    if(fichiers != null) {
      // On les ajoute à  l'adaptateur
      for(File f : fichiers)
        mAdapter.add(f);
      // Puis on le trie
      mAdapter.sort();
    }
  }
```

Cette méthode est d'ailleurs utilisée par la méthode de *callback* `onKeyDown` :

```java
public boolean onKeyDown(int keyCode, KeyEvent event) {
  // Si on a appuyé sur le retour arrière
  if(keyCode == KeyEvent.KEYCODE_BACK) {
    // On prend le parent du répertoire courant
    File parent = mCurrentFile.getParentFile();
    // S'il y a effectivement un parent
    if(parent != null)
      updateDirectory(parent);
    else {
      // Sinon, si c'est la première fois qu'on fait un retour arrière
      if(mCountdown != true) {
        // On indique à l'utilisateur qu'appuyer dessus une seconde fois le fera sortir
        Toast.makeText(this, "Nous sommes déjà à la racine ! Cliquez une seconde fois pour quitter", Toast.LENGTH_SHORT).show();
        mCountdown  = true;
      } else
        // Si c'est la seconde fois, on sort effectivement
        finish();
    }
    return true;
  }
  return super.onKeyDown(keyCode, event);
}
```

Gestion de l'intent pour visualiser un fichier

```java
/**
 * Utilisé pour visualiser un fichier
 * @param pFile le fichier à visualiser
 */
private void seeItem(File pFile) {
  // On crée un intent
  Intent i = new Intent(Intent.ACTION_VIEW);

  String ext = pFile.getName().substring(pFile.getName().
      indexOf(".") + 1).toLowerCase();
  if(ext.equals("mp3"))
    i.setDataAndType(Uri.fromFile(pFile), "audio/mp3");
    /** Faites en autant que vous le désirez */

  try {
    startActivity(i);
    // Et s'il n'y a pas d'activité qui puisse gérer ce type de
        fichier
  } catch (ActivityNotFoundException e) {
    Toast.makeText(this, "Oups, vous n'avez pas d'application
        qui puisse lancer ce fichier", Toast.LENGTH_SHORT).show
        ();
    e.printStackTrace();
  }
}
```

Les menus

Rien d'étonnant ici, normalement vous connaissez déjà tout. À noter que j'ai utilisé deux layouts pour le menu contextuel de manière à pouvoir le changer selon qu'il s'agit d'un répertoire ou d'un fichier :

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  getMenuInflater().inflate(R.menu.activity_explorateur, menu);
  return true;
}

@Override
public boolean onOptionsItemSelected (MenuItem item)
{
  switch(item.getItemId())
  {
    case R.id.menu_options:
      // Intent explicite
      Intent i = new Intent(this, ExploreurPreference.class);
      startActivity(i);
      return true;
  }
```

```
18      return super.onOptionsItemSelected(item);
19    }
20
21    @Override
22    public void onCreateContextMenu(ContextMenu menu, View vue,
          ContextMenuInfo menuInfo) {
23      super.onCreateContextMenu(menu, vue, menuInfo);
24
25      MenuInflater inflater = getMenuInflater();
26      // On récupère des informations sur l'item par apport à l'
            adaptateur
27      AdapterView.AdapterContextMenuInfo info = (AdapterView.
            AdapterContextMenuInfo) menuInfo;
28
29      // On récupère le fichier concerné par le menu contextuel
30      File fichier = mAdapter.getItem(info.position);
31      // On a deux menus, s'il s'agit d'un répertoire ou d'un
            fichier
32      if(!fichier.isDirectory())
33        inflater.inflate(R.menu.context_file, menu);
34      else
35        inflater.inflate(R.menu.context_dir, menu);
36    }
37
38    @Override
39    public boolean onContextItemSelected(MenuItem item) {
40      AdapterView.AdapterContextMenuInfo info = (AdapterView.
            AdapterContextMenuInfo) item.getMenuInfo();
41      // On récupère la position de l'item concerné
42      File fichier = mAdapter.getItem(info.position);
43      switch (item.getItemId()) {
44        case R.id.deleteItem:
45          mAdapter.remove(fichier);
46          fichier.delete();
47          return true;
48
49        case R.id.seeItem:
50          seeItem(fichier);
51          return true;
52      }
53      return super.onContextItemSelected(item);
54    }
```

onCreate

Voici la méthode principale où se situent toutes les initialisations :

```
1    public void onCreate(Bundle savedInstanceState) {
2      super.onCreate(savedInstanceState);
```

```
3
4     // On récupère la ListView de notre activité
5     mList = (ListView) getListView();
6
7     // On vérifie que le répertoire externe est bien accessible
8     if(!Environment.MEDIA_MOUNTED.equals(Environment.
          getExternalStorageState())) {
9       // S'il ne l'est pas, on affiche un message
10      mEmpty = (TextView) mList.getEmptyView();
11      mEmpty.setText("Vous ne pouvez pas accéder aux fichiers");
12    } else {
13      // S'il l'est, on déclare qu'on veut un menu contextuel sur
            les éléments de la liste
14      registerForContextMenu(mList);
15
16      // On récupère les préférences de l'application
17      mPrefs = PreferenceManager.getDefaultSharedPreferences(this
          );
18      // On indique que l'activité est à l'écoute des changements
            de préférences
19      mPrefs.registerOnSharedPreferenceChangeListener(this);
20      // On récupère la couleur voulue par l'utilisateur, par dé
          faut il s'agira du rouge
21      mColor = mPrefs.getInt("repertoireColorPref", Color.RED);
22
23      // On récupère la racine de la carte SD pour qu'elle soit
          le répertoire consulté au départ
24      mCurrentFile = Environment.getExternalStorageDirectory();
25
26      // On change le titre de l'activité pour y mettre le chemin
            actuel
27      setTitle(mCurrentFile.getAbsolutePath());
28
29      // On récupère la liste des fichiers dans le répertoire
          actuel
30      File[] fichiers = mCurrentFile.listFiles();
31
32      // On transforme le tableau en une structure de données de
          taille variable
33      ArrayList<File> liste = new ArrayList<File>();
34      for(File f : fichiers)
35        liste.add(f);
36
37      mAdapter = new FileAdapter(this, android.R.layout.
          simple_list_item_1, liste);
38      // On ajoute l'adaptateur à la liste
39      mList.setAdapter(mAdapter);
40      // On trie la liste
41      mAdapter.sort();
42
```

```
43      // On ajoute un Listener sur les items de la liste
44      mList.setOnItemClickListener(new OnItemClickListener() {
45
46        // Que se passe-t-il en cas de clic sur un élément de la
             liste ?
47        public void onItemClick(AdapterView<?> adapter, View view
             , int position, long id) {
48          File fichier = mAdapter.getItem(position);
49          // Si le fichier est un répertoire...
50          if(fichier.isDirectory())
51            // On change de répertoire courant
52            updateDirectory(fichier);
53          else
54            // Sinon, on lance l'item
55            seeItem(fichier);
56        }
57      });
58    }
59  }
```

▷ Télécharger le projet
Code web : 413822

Améliorations envisageables

Quand la liste est vide ou le périphérique externe est indisponible

On se trouve en face d'un écran blanc pas très intéressant... Ce qui pourrait être plus excitant, c'est un message qui indique à l'utilisateur qu'il n'a pas accès à ce périphérique externe. On peut faire ça en indiquant un layout pour notre `ListActivity` ! Oui, je sais, je vous ai dit de ne pas le faire, parce que notre activité contient principalement une liste, mais là on pousse le concept encore plus loin. Le layout qu'on utilisera doit contenir au moins une `ListView` pour représenter celle de notre `ListActivity`, mais notre application sera bien incapable de la trouver si vous ne lui précisez pas où elle se trouve. Vous pouvez le faire en mettant comme identifiant à la `ListView` `android:id="@android:id/list"`. Si vous voulez qu'un widget ou un layout s'affiche quand la liste est vide, vous devez lui attribuer l'identifiant `android:id="@android:id/empty"`. Pour ma correction, j'ai le XML suivant :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res
     /android"
3   android:orientation="vertical"
4   android:layout_width="fill_parent"
5   android:layout_height="fill_parent"
6   android:paddingLeft="8dp"
7   android:paddingRight="8dp">
8
```

```
 9    <ListView android:id="@android:id/list"
10      android:layout_width="fill_parent"
11      android:layout_height="0dip"
12      android:layout_weight="1"
13      android:drawSelectorOnTop="false"/>
14
15    <TextView android:id="@android:id/empty"
16      android:layout_width="fill_parent"
17      android:layout_height="0dip"
18      android:layout_weight="1"
19      android:text="@string/empty"/>
20  </LinearLayout>
```

Détection automatique du type MIME

Parce que faire une longue liste de « Si on a cette extension pour ce fichier, alors le type MIME, c'est celui-là » est quand même long et contraignant, je vous propose de détecter automatiquement le type MIME d'un objet. Pour cela, on utilisera un objet de type `MimeTypeMap`. Afin de récupérer cet objet, on passe par la méthode statique `MimeTypeMap MimeTypeMap.getSingleton()`.

Petite digression pour vous dire que le *design pattern* singleton a pour objectif de faire en sorte que vous ne puissiez avoir qu'une seule instance d'un objet. C'est pourquoi on utilise la méthode `getSingleton()` qui renvoie toujours le même objet. Il est impossible de construire autrement un objet de type `MimeTypeMap`.

Ensuite c'est simple, il suffit de donner à la méthode String `getMimeTypeFromExtension(String extension)` l'extension de notre fichier. On obtient ainsi :

```
1  MimeTypeMap mime = MimeTypeMap.getSingleton();
2  String ext = fichier.getName().substring(fichier.getName().
       indexOf(".") + 1).toLowerCase();
3  String type = mime.getMimeTypeFromExtension(ext);
```

Détecter les changements d'état du périphérique externe

C'est bien beau tout ça, mais si l'utilisateur se décide tout à coup à changer la carte SD en pleine utilisation, nous ferons face à un gros plantage ! Alors comment contrer ce souci ? C'est simple. Dès que l'état du périphérique externe change, un broadcast intent est transmis pour le signaler à tout le système. Il existe tout un tas d'actions différentes associées à un changement d'état, je vous propose de ne gérer que le cas où le périphérique externe est enlevé, auquel cas l'action est `ACTION_MEDIA_REMOVED`. Notez au passage que l'action pour dire que la carte fonctionne à nouveau est `ACTION_MEDIA_MOUNTED`.

Comme nous l'avons vu dans le cours, il faudra déclarer notre `broadcast receiver` dans le Manifest :

```
<receiver android:name=".ExplorerReceiver"
  android:exported="false">
  <intent-filter>
    <action android:name="android.intent.action.MEDIA_REMOVED"
        />
    <action android:name="android.intent.action.MEDIA_MOUNTED"
        />
  </intent-filter>
</receiver>
```

Ensuite, dans le `receiver` en lui-même, on fait en sorte de viser la liste des éléments s'il y a un problème avec le périphérique externe, ou au contraire de la repeupler dès que le périphérique fonctionne correctement à nouveau. À noter que dans le cas d'un `broadcast Intent` avec l'action `ACTION_MEDIA_MOUNTED`, l'intent aura dans son champ `data` l'emplacement de la racine du périphérique externe :

```
public class ExplorerReceiver extends BroadcastReceiver {
  private ExplorateurActivity mActivity = null;

  public ExplorerReceiver(ExplorateurActivity mActivity) {
    super();
    this.mActivity = mActivity;
  }

  @Override
  public void onReceive(Context context, Intent intent) {
    if(intent.getAction().equals(Intent.ACTION_MEDIA_REMOVED))
      mActivity.setEmpty();
    else if(intent.getAction().equals(Intent.
        ACTION_MEDIA_MOUNTED))
      mActivity.updateDirectory(new File(intent.getData().
          toString()));
  }
}
```

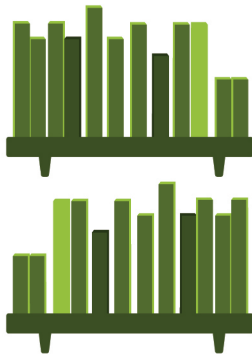
Chapitre 17

Les bases de données

Difficulté :

Ce que nous avons vu précédemment est certes utile, mais ne répondra pas à tous nos besoins. Nous avons besoin d'un moyen efficace de stocker des données complexes et d'y accéder. Or, il nous faudrait des années pour concevoir un système de ce style. Imaginez le travail s'il vous fallait développer de A à Z une bibliothèque multimédia qui puisse chercher en moins d'une seconde parmi plus de 100 000 titres une chanson bien précise ! C'est pour cela que nous avons besoin des bases de données, qui sont optimisées pour ce type de traitements.

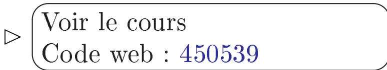
Les bases de données pour Android sont fournies à l'aide de SQLite. L'avantage de SQLite est qu'il s'agit d'un SGBD (Système de Gestion de Base de Données) très compact et par conséquent très efficace pour les applications embarquées, mais pas uniquement puisqu'on le trouve dans Skype, Adobe Reader, Firefox, etc.



Généralités

Vous comprendrez peut-être ce chapitre même si vous n'avez jamais manipulé de bases de données auparavant. Tant mieux, mais cela ne signifie pas que vous serez capables de manipuler correctement les bases de données pour autant. C'est une vraie science que d'agencer les bases de données et il faut beaucoup plus de théorie que nous n'en verrons ici pour modéliser puis réaliser une base de données cohérente et efficace.

Il vous est possible d'apprendre à utiliser les bases de données et surtout MySQL grâce au cours « Administrez vos bases de données avec MySQL » rédigé par Taguan sur le Site du Zéro.


▷ Voir le cours
Code web : 450539

Sur les bases de données

Une base de données est un dispositif permettant de stocker un ensemble d'informations de manière structurée. L'agencement adopté pour organiser les informations s'appelle le **schéma**.

L'unité de base de cette structure s'appelle la **table**. Une table regroupe des ensembles d'informations qui sont composés de manière similaire. Une entrée dans une table s'appelle un **enregistrement,** ou un **tuple**. Chaque entrée est caractérisée par plusieurs renseignements distincts, appelés des **champs** ou **attributs**.

Par exemple, une table peut contenir le prénom, le nom et le métier de plusieurs utilisateurs, on aura donc pour chaque utilisateur les mêmes informations. Il est possible de représenter une table par un tableau, où les champs seront symbolisés par les colonnes du tableau et pour lequel chaque ligne représentera une entrée différente. Regardez la figure 17.1, cela devrait être plus clair.

Nom	Prénom	Métier
Daku	Tenshi	Auteur
John	John	Éditeur
Andro	Wiiid	Validateur
Shakespeare	William	Auteur
...	...	...

Figure 17.1 – Cette table contient quatre tuples qui renseignent toutes des informations du même gabarit pour chaque attribut

Une manière simple d'identifier les éléments dans une table est de leur attribuer une **clé**. C'est-à-dire qu'on va choisir une combinaison de champs qui permettront de récupérer de manière unique un enregistrement. Dans la table présentée à la figure 17.2, l'attribut `Nom` peut être une clé puisque toutes les entrées ont un `Nom` différent. Le problème est qu'il peut arriver que deux utilisateurs aient le même nom, c'est pourquoi on peut aussi envisager la combinaison `Nom` et `Prénom` comme clé.

Nom	Prénom	Métier
Daku	Tenshi	Auteur
John	John	Éditeur
Andro	Wiiid	Validateur
Shakespeare	William	Auteur
...	...	...

FIGURE 17.2 – On choisit comme clé la combinaison Nom-Prénom

Il n'est pas rare qu'une base de données ait plusieurs tables. Afin de lier des tables, il est possible d'insérer dans une table une clé qui appartient à une autre table, auquel cas on parle de **clé étrangère** pour la table qui accueille la clé, comme à la figure 17.3.

Métier	Salaire moyen
Auteur	-10
Éditeur	9987
Validateur	2,5
...	...

Nom	Prénom	Métier
Daku	Tenshi	Auteur
John	John	Éditeur
Andro	Wiiid	Validateur
Shakespeare	William	Auteur
...	...	...

FIGURE 17.3 – Dans notre première table, Métier est une clé étrangère, car elle est clé primaire de la seconde table

Il est possible d'effectuer des opérations sur une base de données, comme créer des tables, supprimer des entrées, etc. L'opération qui consiste à lire des informations qui se trouvent dans une base de données s'appelle la **sélection**.

Pour effectuer des opérations sur plusieurs tables, on passe par une **jointure**, c'est-à-dire qu'on combine des attributs qui appartiennent à plusieurs tables pour les présenter conjointement.

Afin d'effectuer toutes ces opérations, on passe par un **langage de requête**. Celui dont nous avons besoin s'appelle **SQL**. Nous verrons un rappel des opérations principales dans ce chapitre.

Enfin, une base de données est destinée à recueillir des informations simples, c'est pourquoi on évite d'y insérer des données lourdes comme des fichiers ou des données brutes. Au lieu de mettre directement des images ou des vidéos, on préfèrera insérer un URI qui dirige vers ces fichiers.

Sur SQLite

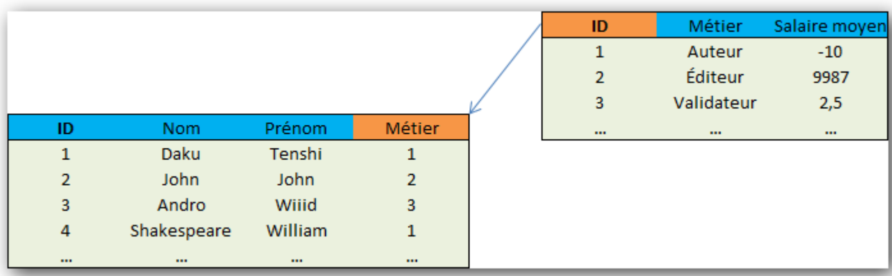
Contrairement à MySQL par exemple, SQLite ne nécessite pas de serveur pour fonctionner, ce qui signifie que son exécution se fait dans le même processus que celui de

l'application. Par conséquent, une opération massive lancée dans la base de données aura des conséquences visibles sur les performances de votre application. Ainsi, il vous faudra savoir maîtriser son implémentation afin de ne pas pénaliser le restant de votre exécution.

Sur SQLite pour Android

SQLite a été inclus dans le cœur même d'Android, c'est pourquoi chaque application peut avoir sa propre base. De manière générale, les bases de données sont stockées dans les répertoires de la forme /data/data/<package>/databases. Il est possible d'avoir plusieurs bases de données par application, cependant chaque fichier créé l'est selon le mode MODE_PRIVATE, par conséquent les bases ne sont accessibles qu'au sein de l'application elle-même. Notez que ce n'est pas pour autant qu'une base de données ne peut pas être partagée avec d'autres applications.

Enfin, pour des raisons qui seront expliquées dans un chapitre ultérieur, il est préférable de faire en sorte que la clé de chaque table soit un identifiant qui s'incrémente automatiquement. Notre schéma devient donc la figure 17.4.



ID	Métier	Salaire moyen
1	Auteur	-10
2	Éditeur	9987
3	Validateur	2,5
...	...	...

ID	Nom	Prénom	Métier
1	Daku	Tenshi	1
2	John	John	2
3	Andro	Wiiid	3
4	Shakespeare	William	1
...	...	...	...

FIGURE 17.4 – Un identifiant a été ajouté

Création et mise à jour

La solution la plus évidente est d'utiliser une classe qui nous aidera à maîtriser toutes les relations avec la base de données. Cette classe dérivera de SQLiteOpenHelper. Au moment de la création de la base de données, la méthode de *callback* void onCreate(SQLiteDatabase db) est automatiquement appelée, avec le paramètre db qui représentera la base. C'est dans cette méthode que vous devrez lancer les instructions pour créer les différentes tables et éventuellement les remplir avec des données initiales.

Pour créer une table, il vous faudra réfléchir à son nom et à ses attributs. Chaque attribut sera défini à l'aide d'un type de données. Ainsi, dans la table Metier de notre exemple, nous avons trois attributs :

- ID, qui est un entier auto-incrémental pour représenter la clé ;

– `Métier`, qui est une chaîne de caractères ;
– `Salaire`, qui est un nombre réel.

Pour SQLite, c'est simple, il n'existe que cinq types de données :

– `NULL` pour les données `NULL`.
– `INTEGER` pour les entiers (sans virgule).
– `REAL` pour les nombres réels (avec virgule).
– `TEXT` pour les chaînes de caractères.
– `BLOB` pour les données brutes, par exemple si vous voulez mettre une image dans votre base de données (ce que vous ne ferez jamais, n'est-ce pas ?).

La création de table se fait avec une syntaxe très naturelle :

```
CREATE TABLE nom_de_la_table (
  nom_du_champ_1 type {contraintes},
  nom_du_champ_2 type {contraintes},
  ...);
```

Pour chaque attribut, on doit déclarer au moins deux informations :

– Son nom, afin de pouvoir l'identifier ;
– Son type de donnée.

Mais il est aussi possible de déclarer des contraintes pour chaque attribut à l'emplacement de `{contraintes}`. On trouve comme contraintes :

– `PRIMARY KEY` pour désigner la clé primaire sur un attribut ;
– `NOT NULL` pour indiquer que cet attribut ne peut valoir `NULL` ;
– `CHECK` afin de vérifier que la valeur de cet attribut est cohérente ;
– `DEFAULT` sert à préciser une valeur par défaut.

Ce qui peut donner par exemple :

```
CREATE TABLE nom_de_la_table (
  nom_du_champ_1 INTEGER PRIMARY KEY,
  nom_du_champ_2 TEXT NOT NULL,
  nom_du_champ_3 REAL NOT NULL CHECK (nom_du_champ_3 > 0),
  nom_du_champ_4 INTEGER DEFAULT 10);
```

Il existe deux types de requêtes SQL. Celles qui appellent une réponse, comme la sélection, et celles qui n'appellent pas de réponse. Afin d'exécuter une requête SQL pour laquelle on ne souhaite pas de réponse ou on ignore la réponse, il suffit d'utiliser la méthode `void execSQL(String sql)`. De manière générale, on utilisera `execSQL(String)` dès qu'il ne s'agira pas de faire un `SELECT`, `UPDATE`, `INSERT` ou `DELETE`. Par exemple, pour notre table `Metier` :

```
public class DatabaseHandler extends SQLiteOpenHelper {
  public static final String METIER_KEY = "id";
  public static final String METIER_INTITULE = "intitule";
  public static final String METIER_SALAIRE = "salaire";

  public static final String METIER_TABLE_NAME = "Metier";
  public static final String METIER_TABLE_CREATE =
```

```
8       "CREATE TABLE " + METIER_TABLE_NAME + " (" +
9         METIER_KEY + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
10        METIER_INTITULE + " TEXT, " +
11        METIER_SALAIRE + " REAL);";
12
13    public DatabaseHandler(Context context, String name,
          CursorFactory factory, int version) {
14      super(context, name, factory, version);
15    }
16
17    @Override
18    public void onCreate(SQLiteDatabase db) {
19      db.execSQL(METIER_TABLE_CREATE);
20    }
21  }
```

Comme vous l'aurez remarqué, une pratique courante avec la manipulation des bases de données est d'enregistrer les attributs, tables et requêtes dans des constantes de façon à les retrouver et les modifier plus facilement. Tous ces attributs sont `public` puisqu'il est possible qu'on manipule la base en dehors de cette classe.

Si vous installez la base de données sur un périphérique externe, il vous faudra demander la permission `WRITE_EXTERNAL_STORAGE`, sinon votre base de données sera en lecture seule. Vous pouvez savoir si une base de données est en lecture seule avec la méthode `boolean isReadOnly()`.

Le problème du code précédent, c'est qu'il ne fonctionnera pas, et ce pour une raison très simple : il faut aussi implémenter la méthode `void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)` qui est déclenchée à chaque fois que l'utilisateur met à jour son application. `oldVersion` est le numéro de l'ancienne version de la base de données que l'application utilisait, alors que `newVersion` est le numéro de la nouvelle version. En fait, Android rajoute automatiquement dans la base une table qui contient la dernière valeur connue de la base. À chaque lancement, Android vérifiera la dernière version de la base par rapport à la version actuelle dans le code. Si le numéro de la version actuelle est supérieur à celui de la dernière version, alors cette méthode est lancée.

En général, le contenu de cette méthode est assez constant puisqu'on se contente de supprimer les tables déjà existantes pour les reconstruire suivant le nouveau schéma :

```
1  public static final String METIER_TABLE_DROP = "DROP TABLE IF
       EXISTS " + METIER_TABLE_NAME + ";";
2
3  @Override
4  public void onUpgrade(SQLiteDatabase db, int oldVersion, int
       newVersion) {
5    db.execSQL(METIER_TABLE_DROP);
6    onCreate(db);
7  }
```

Opérations usuelles

Récupérer la base

Si vous voulez accéder à la base de données n'importe où dans votre code, il vous suffit de construire une instance de votre `SQLiteOpenHelper` avec le constructeur `SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)`, où `name` est le nom de la base, `factory` est un paramètre qu'on va oublier pour l'instant — qui accepte très bien les `null` — et `version` la version voulue de la base de données.

On utilise `SQLiteDatabase getWritableDatabase()` pour récupérer ou créer une base sur laquelle vous voulez lire et/ou écrire. La dernière méthode qui est appelée avant de fournir la base à `getWritableDatabase()` est la méthode de *callback* `void onOpen(SQLiteDatabase db)`, c'est donc l'endroit où vous devriez effectuer des opérations si vous le souhaitez.

Cependant, le système fera appel à une autre méthode avant d'appeler `onOpen(SQLiteDatabase)`. Cette méthode dépend de l'état de la base et de la version qui a été fournie à la création du `SQLiteOpenHelper` :

- S'il s'agit de la première fois que vous appelez la base, alors ce sera la méthode `onCreate(SQLiteDatabase)` qui sera appelée.
- Si la version fournie est plus récente que la dernière version fournie, alors on fait appel à `onUpgrade(SQLiteDatabase, int, int)`.
- En revanche, si la version fournie est plus ancienne, on considère qu'on effectue un retour en arrière et c'est `onDowngrade(SQLiteDatabase, int, int)` qui sera appelée.

L'objet `SQLiteDatabase` fourni est un objet en cache, constant. Si des opérations se déroulent sur la base après que vous avez récupéré cet objet, vous ne les verrez pas sur l'objet. Il faudra en recréer un pour avoir le nouvel état de la base.

Vous pouvez aussi utiliser la méthode `SQLiteDatabase getReadableDatabase()` pour récupérer la base, la différence étant que la base sera en lecture seule, mais uniquement s'il y a un problème qui l'empêche de s'ouvrir normalement. Avec `getWritableDatabase()`, si la base ne peut pas être ouverte en écriture, une exception de type `SQLiteException` sera lancée. Donc, si vous souhaitez ne faire que lire dans la base, utilisez en priorité `getReadableDatabase()`.

Ces deux méthodes peuvent prendre du temps à s'exécuter.

Enfin, il faut fermer une base comme on ferme un flux avec la méthode `void close()`.

Récupérer un `SQLiteDatabase` avec l'une des deux méthodes précédentes équivaut à faire un `close()` sur l'instance précédente de `SQLiteDatabase`.

Réfléchir, puis agir

Comme je l'ai déjà dit, chacun fait ce qu'il veut dès qu'il doit manipuler une base de données, ce qui fait qu'on se retrouve parfois avec du code incompréhensible ou difficile à mettre à jour. Une manière efficace de gérer l'interfaçage avec une base de données est de passer par un DAO (Data Access Object), un objet qui incarne l'accès aux données de la base.

En fait, cette organisation implique d'utiliser deux classes :

– Une classe (dans mon cas `Metier`) qui représente les informations et qui peut contenir un enregistrement d'une table. Par exemple, on aura une classe `Metier` pour symboliser les différentes professions qui peuvent peupler cette table.
– Une classe `contrôleur`, le DAO pour ainsi dire, qui effectuera les opérations sur la base.

La classe `Metier`

Très simple, il suffit d'avoir un attribut pour chaque attribut de la table et d'ajouter des méthodes pour y accéder et les modifier :

```
1  public class Metier {
2    // Notez que l'identifiant est un long
3    private long id;
4    private String intitule;
5    private float salaire;
6
7    public Metier(long id, String intitule, float salaire) {
8      super();
9      this.id = id;
10     this.intitule = intitule;
11     this.salaire = salaire;
12   }
13
14   public long getId() {
15     return id;
16   }
17
18   public void setId(long id) {
19     this.id = id;
20   }
21
22   public String getIntitule() {
23     return intitule;
24   }
25
26   public void setIntitule(String intitule) {
27     this.intitule = intitule;
28   }
29
```

```
30    public float getSalaire() {
31      return salaire;
32    }
33
34    public void setSalaire(float salaire) {
35      this.salaire = salaire;
36    }
37
38  }
```

La classe DAO

On doit y inclure au moins les méthodes CRUD (Create, Read, Update, Delete), autrement dit les méthodes qui permettent l'ajout d'entrées dans la base, la récupération d'entrées, la mise à jour d'enregistrements ou encore la suppression de tuples. Bien entendu, ces méthodes sont à adapter en fonction du contexte et du métier. De plus, on rajoute les constantes globales déclarées précédemment dans la base :

```
1   public class MetierDAO {
2     public static final String TABLE_NAME = "metier";
3     public static final String KEY = "id";
4     public static final String INTITULE = "intitule";
5     public static final String SALAIRE = "salaire";
6
7     public static final String TABLE_CREATE = "CREATE TABLE " +
          TABLE_NAME + " (" + KEY + " INTEGER PRIMARY KEY
          AUTOINCREMENT, " + INTITULE + " TEXT, " + SALAIRE + " REAL
          );";
8
9     public static final String TABLE_DROP =  "DROP TABLE IF
          EXISTS " + TABLE_NAME + ";";
10
11    /**
12     * @param m le métier à ajouter à la base
13     */
14    public void ajouter(Metier m) {
15      // CODE
16    }
17
18    /**
19     * @param id l'identifiant du métier à supprimer
20     */
21    public void supprimer(long id) {
22      // CODE
23    }
24
25    /**
26     * @param m le métier modifié
27     */
```

```
28    public void modifier(Metier m) {
29      // CODE
30    }
31
32    /**
33     * @param id l'identifiant du métier à récupérer
34     */
35    public Metier selectionner(long id) {
36      // CODE
37    }
38  }
```

Il ne s'agit bien entendu que d'un exemple, dans la pratique on essaie de s'adapter au contexte quand même, là je n'ai mis que des méthodes génériques.

Comme ces opérations se déroulent sur la base, nous avons besoin d'un accès à la base. Pour cela, et comme j'ai plusieurs tables dans mon schéma, j'ai décidé d'implémenter toutes les méthodes qui permettent de récupérer ou de fermer la base dans une classe abstraite :

```
1   public abstract class DAOBase {
2     // Nous sommes à la première version de la base
3     // Si je décide de la mettre à jour, il faudra changer cet
          attribut
4     protected final static int VERSION = 1;
5     // Le nom du fichier qui représente ma base
6     protected final static String NOM = "database.db";
7
8     protected SQLiteDatabase mDb = null;
9     protected DatabaseHandler mHandler = null;
10
11    public DAOBase(Context pContext) {
12      this.mHandler = new DatabaseHandler(pContext, NOM, null,
            VERSION);
13    }
14
15    public SQLiteDatabase open() {
16      // Pas besoin de fermer la dernière base puisque
            getWritableDatabase s'en charge
17      mDb = mHandler.getWritableDatabase();
18      return mDb;
19    }
20
21    public void close() {
22      mDb.close();
23    }
24
25    public SQLiteDatabase getDb() {
26      return mDb;
27    }
28  }
```

Ainsi, pour pouvoir utiliser ces méthodes, la définition de ma classe `MetierDAO` devient :

```
public class MetierDAO extends DAOBase
```

Ajouter

Pour ajouter une entrée dans la table, on utilise la syntaxe suivante :

```
INSERT INTO nom_de_la_table
  (nom_de_la_colonne1, nom_de_la_colonne2, ..) VALUES (valeur1,
      valeur2, ..)
```

La partie `(nom_de_la_colonne1, nom_de_la_colonne2, ...)` permet d'associer une valeur à une colonne précise à l'aide de la partie `(valeur1, valeur2, ...)`. Ainsi la colonne 1 aura la valeur 1 ; la colonne 2, la valeur 2 ; etc.

```
INSERT INTO Metier (Salaire, Metier) VALUES (50.2, "
    Caricaturiste")
```

Pour certains SGBD, l'instruction suivante est aussi possible afin d'insérer une entrée vide dans la table.

```
INSERT INTO Metier;
```

Cependant, avec SQLite ce n'est pas possible, il faut préciser au moins une colonne, quitte à lui passer comme valeur `NULL`.

```
INSERT INTO Metier (Salaire) VALUES (NULL);
```

En Java, pour insérer une entrée, on utilisera la méthode `long insert(String table, String nullColumnHack, ContentValues values)`, qui renvoie le numéro de la ligne ajoutée où :

- `table` est l'identifiant de la table dans laquelle insérer l'entrée.
- `nullColumnHack` est le nom d'une colonne à utiliser au cas où vous souhaiteriez insérer une entrée vide. Prenez n'importe laquelle.
- `values` est un objet qui représente l'entrée à insérer.

Les `ContentValues` sont utilisés pour insérer des données dans la base. Ainsi, on peut dire qu'ils fonctionnent un peu comme les `Bundle` par exemple, puisqu'on peut y insérer des couples identifiant-valeur, qui représenteront les attributs des objets à insérer dans la base. L'identifiant du couple doit être une chaîne de caractères qui représente une des colonnes de la table visée. Ainsi, pour insérer le métier « Caricaturiste », il me suffit de faire :

```
ContentValues value = new ContentValues();
value.put(MetierDAO.INTITULE, m.getIntitule());
value.put(MetierDAO.SALAIRE, m.getSalaire());
mDb.insert(MetierDAO.TABLE_NAME, null, value);
```

Je n'ai pas besoin de préciser de valeur pour l'identifiant puisqu'il s'incrémente tout seul.

Supprimer

La méthode utilisée pour supprimer est quelque peu différente : `int delete(String table, String whereClause, String[] whereArgs)`. L'entier renvoyé est le nombre de lignes supprimées. Dans cette méthode :

- `table` est l'identifiant de la table.
- `whereClause` correspond au `WHERE` en SQL. Par exemple, pour sélectionner la première valeur dans la table `Metier`, on mettra pour `whereClause` la chaîne « `id = 1` ». En pratique, on préférera utiliser la chaîne « `id = ?` » et je vais vous expliquer pourquoi tout de suite.
- `whereArgs` est un tableau des valeurs qui remplaceront les « ? » dans `whereClause`. Ainsi, si `whereClause` vaut « `LIKE ? AND salaire > ?` » et qu'on cherche les métiers qui ressemblent à « ingénieur avec un salaire supérieur à 1000 € », il suffit d'insérer dans `whereArgs` un `String[]` du genre `{"ingenieur", "1000"}`.

Ainsi dans notre exemple, pour supprimer une seule entrée, on fera :

```
1 public void supprimer(long id) {
2   mDb.delete(TABLE_NAME, KEY + " = ?", new String[] {String.
        valueOf(id)});
3 }
```

Mise à jour

Rien de très surprenant ici, la syntaxe est très similaire à la précédente : `int update(String table, ContentValues values, String whereClause, String[] whereArgs)`. On ajoute juste le paramètre `values` pour représenter les changements à effectuer dans le ou les enregistrements cibles. Donc, si je veux mettre à jour le salaire d'un métier, il me suffit de mettre à jour l'objet associé et d'insérer la nouvelle valeur dans un `ContentValues` :

```
1 ContentValues value = new ContentValues();
2 value.put(SALAIRE, m.getSalaire());
3 mDb.update(TABLE_NAME, value, KEY  + " = ?", new String[] {
      String.valueOf(m.getId())});
```

Sélection

Ici en revanche, la méthode est plus complexe et revêt trois formes différentes en fonction des paramètres qu'on veut lui passer. La première forme est celle-ci :

```
1 Cursor query (boolean distinct, String table, String[] columns,
       String selection, String[] selectionArgs, String groupBy,
      String having, String orderBy, String limit)
```

La deuxième forme s'utilise sans l'attribut `limit` et la troisième sans les attributs `limit` *et* `distinct`. Ces paramètres sont vraiment explicites puisqu'ils représentent à chaque

fois des mots-clés du SQL ou des attributs que nous avons déjà rencontrés. Voici leur signification :

- `distinct`, si vous ne voulez pas de résultats en double.
- `table` est l'identifiant de la table.
- `columns` est utilisé pour préciser les colonnes à afficher.
- `selection` est l'équivalent du `whereClause` précédent.
- `selectionArgs` est l'équivalent du `whereArgs` précédent.
- `group by` permet de grouper les résultats.
- `having` est utile pour filtrer parmi les groupes.
- `order by` permet de trier les résultats. Mettre `ASC` pour trier dans l'ordre croissant et `DESC` pour l'ordre décroissant.
- `limit` pour fixer un nombre maximal de résultats voulus.

Pour être franc, utiliser ces méthodes m'agace un peu, c'est pourquoi je préfère utiliser `Cursor rawQuery(String sql, String[] selectionArgs)` où je peux écrire la requête que je veux dans `sql` et remplacer les « ? » dans `selectionArgs`. Ainsi, si je veux tous les métiers qui rapportent en moyenne plus de 1€, je ferai :

```
1 Cursor c = mDb.rawQuery("select " + INTITULE + " from " +
      TABLE_NAME + " where salaire > ?", new String[]{"1"});
```

Les curseurs

Manipuler les curseurs

Les curseurs sont des objets qui contiennent les résultats d'une recherche dans une base de données. Ce sont en fait des objets qui fonctionnent comme les tableaux que nous avons vus précédemment, ils contiennent les colonnes et lignes qui ont été renvoyées par la requête.

Ainsi, pour la requête suivante sur notre table `Metier` :

```
1 SELECT id, intitule, salaire from Metier;
```

. . . on obtient le résultat visible à la figure 17.5, dans un curseur.

ID	Métier	Salaire moyen
1	Auteur	-10
2	Éditeur	9987
3	Validateur	2,5

FIGURE 17.5 – On a trois lignes et trois colonnes

Les lignes

Ainsi, pour parcourir les résultats d'une requête, il faut procéder ligner par ligne. Pour naviguer parmi les lignes, on peut utiliser les méthodes suivantes :

- `boolean moveToFirst()` pour aller à la première ligne.
- `boolean moveToLast()` pour aller à la dernière.
- `boolean moveToPosition(int position)` pour aller à la `position` voulue, sachant que vous pouvez savoir le nombre de lignes avec la méthode `int getCount()`.

Cependant, il y a mieux. En fait, un `Cursor` est capable de retenir la position du dernier élément que l'utilisateur a consulté, il est donc possible de naviguer d'avant en arrière parmi les lignes grâce aux méthodes suivantes :

- `boolean moveToNext()` pour aller à la ligne suivante. Par défaut on commence à la ligne -1, donc, en utilisant un `moveToNext()` sur un tout nouveau `Cursor`, on passe à la première ligne. On aurait aussi pu accéder à la première ligne avec `moveToFirst()`, bien entendu.
- `boolean moveToPrevious()` pour aller à l'entrée précédente.

Vous remarquerez que toutes ces méthodes renvoient des booléens. Ces booléens valent `true` si l'opération s'est déroulée avec succès, sinon `false` (auquel cas la ligne demandée n'existe pas).

Pour récupérer la position actuelle, on utilise `int getPosition()`. Vous pouvez aussi savoir si vous êtes après la dernière ligne avec `boolean isAfterLast()`.

Par exemple, pour naviguer entre toutes les lignes d'un curseur, on fait :

```
1 while (cursor.moveToNext()) {
2   // Faire quelque chose
3 }
4 cursor.close();
```

Ou encore

```
1 for(cursor.moveToFirst(); !cursor.isAfterLast(); cursor.
      moveToNext()) {
2   // Votre code
3 }
4 cursor.close();
```

Les colonnes

Vous savez déjà à l'avance que vous avez trois colonnes, dont la première contient un entier, la deuxième, une chaîne de caractères, et la troisième, un réel. Pour récupérer le contenu d'une de ces colonnes, il suffit d'utiliser une méthode du style `X getX(int columnIndex)` avec `X` le typage de la valeur à récupérer et `columnIndex` la colonne dans laquelle se trouve cette valeur. On peut par exemple récupérer un `Metier` complet avec :

```
1 long id = cursor.getLong(0);
2 String intitule = cursor.getString(1);
```

```
double salaire = cursor.getDouble(2);
Metier m = new Metier (id, intitule, salaire);
```

Il ne vous est pas possible de récupérer le nom ou le type des colonnes, il vous faut donc le savoir à l'avance.

L'adaptateur pour les curseurs

Comme n'importe quel adaptateur, un `CursorAdapter` fera la transition entre des données et un `AdapterView`. Cependant, comme on trouve rarement *une seule* information dans un curseur, on préférera utiliser un `SimpleCursorAdapter`, qui est un équivalent au `SimpleAdapter` que nous avons déjà étudié.

Pour construire ce type d'adaptateur, on utilisera le constructeur suivant :

```
SimpleCursorAdapter (Context context, int layout, Cursor c,
    String[] from, int[] to)
```

... où :

- `layout` est l'identifiant de la mise en page des vues dans l'`AdapterView`.
- `c` est le curseur. On peut mettre `null` si on veut ajouter le curseur *a posteriori*.
- `from` indique une liste de noms des colonnes afin de lier les données au layout.
- `to` contient les `TextView` qui afficheront les colonnes contenues dans `from`.

Tout cela est un peu compliqué à comprendre, je le conçois. Alors nous allons faire un layout spécialement pour notre table `Metier`.

Avant tout, sachez que pour utiliser un `CursorAdapter` ou n'importe quelle classe qui dérive de `CursorAdapter`, votre curseur *doit* contenir une colonne qui s'appelle `_id`. Si ce n'est pas le cas, vous n'avez bien entendu pas à recréer tout votre schéma, il vous suffit d'adapter vos requêtes pour que la colonne qui permet l'identification s'appelle `_id`, ce qui donne avec la requête précédente : `SELECT id as _id, intitule, salaire from Metier;`.

Le layout peut par exemple ressembler au code suivant, que j'ai enregistré dans `cursor_row.xml`.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <TextView
    android:id="@+id/intitule"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="50" />
```

```
12
13      <TextView
14      android:id="@+id/salaire"
15      android:layout_width="fill_parent"
16      android:layout_height="fill_parent"
17      android:layout_weight="50" />
18
19  </LinearLayout>
```

Ensuite, pour utiliser le constructeur, c'est très simple, il suffit de faire :

```
1  SimpleCursorAdapter adapter = new SimpleCursorAdapter (context ,
       R.layout.cursor_row , cursor , new String[]{MetierDAO.
      Intitule , MetierDAO.Salire}, new int[]{R.id.intitule , R.id.
      salaire})
```

En résumé

- Android intègre au sein même de son système une base de données SQLite qu'il partage avec toutes les applications du système (avec des droits très spécifique à chacune pour qu'elle n'aille pas voir chez le voisin).
- La solution la plus évidente pour utiliser une base de données est de mettre en place une classe qui maitrisera les accès entre l'application et la base de données.
- En fonction des besoins de votre application, il est utile de mettre en place une série d'opérations usuelles accessibles à partir d'un DAO. Ces méthodes sont l'ajout, la suppression, la mise à jour et la sélection de données.
- Les curseurs sont des objets qui contiennent les résultats d'une recherche dans une base de données. Ce sont en fait des objets qui fonctionnent comme les tableaux que nous avons vus précédemment sur les chapitres des adaptateurs, ils contiennent les colonnes et les lignes qui ont été renvoyées par la requête.

Quatrième partie

Concepts avancés

Chapitre 18

Le travail en arrière-plan

Difficulté :

L'un de vos objectifs sera de travailler sur la réactivité de vos applications, c'est-à-dire de faire en sorte qu'elles ne semblent pas molles ou ne se bloquent pas sans raison apparente pendant une durée significative. En effet, l'utilisateur est capable de détecter un ralentissement s'il dure plus de 100 ms, ce qui est un laps de temps très court. Pis encore, Android lui-même peut déceler quand votre application n'est pas assez réactive, auquel cas il lancera une boîte de dialogue qui s'appelle ANR (Application Not Responding) et qui incitera l'utilisateur à quitter l'application. Il existe deux évènements qui peuvent lancer des ANR : l'application ne répond pas à une impulsion de l'utilisateur sur l'interface graphique en moins de cinq secondes ; un `Broadcast Receiver` s'exécute en plus de dix secondes.

C'est pourquoi nous allons voir ici comment faire exécuter du travail en arrière-plan, de façon à exécuter du code en parallèle de votre interface graphique, pour ne pas la bloquer quand on veut effectuer de grosses opérations qui risqueraient d'affecter l'expérience de l'utilisateur.

La gestion du multitâche par Android

Comme vous le savez, un programme informatique est constitué d'instructions destinées au processeur. Ces instructions sont présentées sous la forme d'un code, et lors de l'exécution de ce code, les instructions sont traitées par le processeur dans un ordre précis.

Tous les programmes Android s'exécutent dans ce qu'on appelle un **processus**. On peut définir un processus comme une instance d'un programme informatique qui est en cours d'exécution. Il contient non seulement le code du programme, mais aussi des variables qui représentent son état courant. Parmi ces variables s'en trouvent certaines qui permettent de définir la plage mémoire qui est mise à la disposition du processus.

Pour être exact, ce n'est pas le processus en lui-même qui va exécuter le code, mais l'un de ses constituants. Les constituants destinés à exécuter le code s'appellent des **threads** (« fils d'exécution » en français). Dans le cas d'Android, les threads sont contenus dans les processus. Un processus peut avoir un ou plusieurs threads, par conséquent un processus peut exécuter plusieurs portions du code en parallèle s'il a plusieurs threads. Comme un processus n'a qu'une plage mémoire, alors tous les threads se partagent les accès à cette plage mémoire. On peut voir à la figure 18.1 deux processus. Le premier possède deux threads, le second en possède un seul. On peut voir qu'il est possible de communiquer entre les threads ainsi qu'entre les processus.

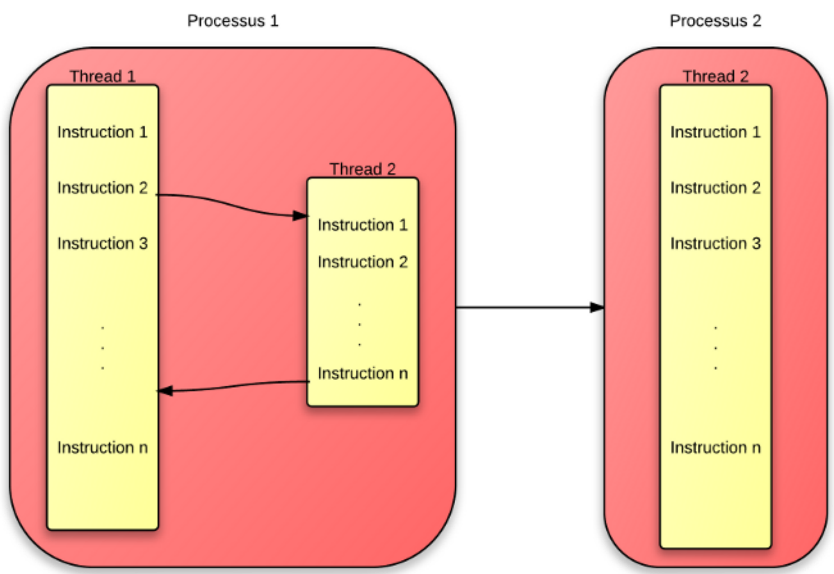


FIGURE 18.1 – Schéma de fonctionnement des threads

Vous vous rappelez qu'une application Android est constituée de composants, n'est-ce

pas ? Nous n'en connaissons que deux types pour l'instant, les activités et les receivers. Il peut y avoir plusieurs de ces composants dans une application. Dès qu'un composant est lancé (par exemple au démarrage de l'application ou quand on reçoit un `Broadcast Intent` dans un receiver), si cette application n'a pas de processus fonctionnel, alors un nouveau sera créé. Tout processus nouvellement créé ne possède qu'un thread. Ce thread s'appelle le **thread principal**.

En revanche, si un composant démarre alors qu'il y a déjà un processus pour cette application, alors le composant se lancera dans le processus en utilisant le même thread.

Processus

Par défaut, tous les composants d'une même application se lancent dans le même processus, et d'ailleurs c'est suffisant pour la majorité des applications. Cependant, si vous le désirez et si vous avez une raison bien précise de le faire, il est possible de définir dans quel processus doit se trouver tel composant de telle application à l'aide de la déclaration du composant dans le Manifest. En effet, l'attribut `android:process` permet de définir le processus dans lequel ce composant est censé s'exécuter, afin que ce composant ne suive pas le même cycle de vie que le restant de l'application. De plus, si vous souhaitez qu'un composant s'exécute dans un processus différent mais reste privé à votre application, alors rajoutez « `:` » à la déclaration du nom du processus.

```
1  <activity
2    android:name=".SensorsActivity"
3    android:label="@string/app_name" >
4    <intent-filter>
5      <action android:name="android.intent.action.MAIN" />
6      <category android:name="android.intent.category.LAUNCHER" /
         >
7    </intent-filter>
8  </activity>
9
10 <activity
11   android:name=".DetailActivity" >
12 </activity>
13
14 <receiver
15   android:name=".LowBatteryReceiver"
16   android:process=":sdz.chapitreQuatre.process.deux" >
17   <intent-filter>
18     <action android:name="android.intent.action.BATTERY_LOW" />
19   </intent-filter>
20 </receiver>
```

Ici, j'ai un receiver qui va s'enclencher dès que la batterie devient faible. Configuré de cette manière, mon receiver ne pourra démarrer que si l'application est lancée (comme j'ai rajouté « `:` », seule mon application pourra le lancer) ; cependant, si l'utilisateur ferme l'application alors que le receiver est en route, le receiver ne s'éteindra pas puisqu'il se trouvera dans un autre processus que le restant des composants.

Quand le système doit décider quel processus il doit tuer, pour libérer de la mémoire principalement, il mesure quelle est l'importance relative de chaque processus pour l'utilisateur. Par exemple, il sera plus enclin à fermer un processus qui ne contient aucune activité visible pour l'utilisateur, alors que d'autres ont des composants qui fonctionnent encore — une activité visible ou un receiver qui gère un évènement. On dit qu'une activité visible a une plus grande priorité qu'une activité non visible.

Threads

Quand une activité est lancée, le système crée un thread principal dans lequel s'exécutera l'application. C'est ce thread qui est en charge d'écouter les évènements déclenchés par l'utilisateur quand il interagit avec l'interface graphique. C'est pourquoi le second nom du thread principal est **thread UI** (UI pour *User Interface*, « interface utilisateur » en français).

Mais il est possible d'avoir plusieurs threads. Android utilise un *pool* de threads (comprendre une réserve de threads, pas une piscine de threads) pour gérer le multitâche. Un pool de threads comprend un certain nombre n de threads afin d'exécuter un certain nombre m de tâches (n et m n'étant pas forcément identiques) qui se trouvent dans un autre pool en attendant qu'un thread s'occupe d'elles. Logiquement, un pool est organisé comme une file, ce qui signifie qu'on empile les éléments les uns sur les autres et que nous n'avons accès qu'au sommet de cet empilement. Les résultats de chaque thread sont aussi placés dans un pool de manière à pouvoir les récupérer dans un ordre cohérent. Dès qu'un thread complète sa tâche, il va demander la prochaine tâche qui se trouve dans le pool jusqu'à ce qu'il n'y ait plus de tâches.

Avant de continuer, laissez-moi vous expliquer le fonctionnement interne de l'interface graphique. Dès que vous effectuez une modification sur une vue, que ce soit un widget ou un layout, cette modification ne se fait pas instantanément. À la place, un évènement est créé. Il génère un message, qui sera envoyé dans une pile de messages. L'objectif du thread UI est d'accéder à la pile des messages, de dépiler le premier message à traiter, de le traiter, puis de passer au suivant. De plus, ce thread s'occupe de toutes les méthodes de *callback* du système, par exemple `onCreate()` ou `onKeyDown()`. Si le système est occupé à un travail intensif, il ne pourra pas traiter les méthodes de *callback* ou les interactions avec l'utilisateur. De ce fait, un ARN est déclenché pour signaler à l'utilisateur qu'Android n'est pas d'accord avec ce comportement.

De la sorte, il faut respecter deux règles dès qu'on manipule des threads :

1. Ne jamais bloquer le thread UI.
2. Ne pas manipuler les vues standards en dehors du thread UI.

Enfin, on évite certaines opérations dans le thread UI, en particulier :

- Accès à un réseau, même s'il s'agit d'une courte opération en théorie.
- Certaines opérations dans la base de données, surtout les sélections multiples.
- Les accès fichiers, qui sont des opérations plutôt lentes.
- Enfin, les accès matériels, car certains demandent des temps de chargement vraiment trop longs.

Mais voyons un peu les techniques qui nous permettrons de faire tranquillement ces opérations.

Gérer correctement les threads simples

La base

En Java, un thread est représenté par un objet de type `Thread`, mais avant cela laissez-moi vous parler de l'interface `Runnable`. Cette interface représente les objets qui sont capables de faire exécuter du code au processeur. Elle ne possède qu'une méthode, `void run()`, dans laquelle il faut écrire le code à exécuter.

Ainsi, il existe deux façons d'utiliser les threads. Comme `Thread` implémente `Runnable`, alors vous pouvez très bien créer une classe qui dérive de `Thread` afin de redéfinir la méthode `void run()`. Par exemple, ce thread fait en sorte de chercher un texte dans un livre pour le mettre dans un `TextView` :

```
1  public class ChercherTexte extends Thread {
2    // La phrase à chercher dans le texte
3    public String a_chercher = "Être ou ne pas être";
4    // Le livre
5    public String livre;
6    // Le TextView dans lequel mettre le résultat
7    public TextView display;
8  
9    public void run() {
10     int caractere = livre.indexOf(a_chercher);
11     display.setText("Cette phrase se trouve au " + caractere +
           " ème caractère.");
12   }
13 }
```

Puis on ajoute le `Thread` à l'endroit désiré et on le lance avec `synchronized void start ()` :

```
1  public void onClick(View v) {
2    Thread t = new Thread();
3  
4    t.livre = hamlet;
5    t.display = v;
6  
7    t.start();
8  }
```

Une méthode synchronized a un verrou. Dès qu'on lance cette méthode, alors le verrou s'enclenche et il est impossible pour d'autres threads de lancer la même méthode.

Mais ce n'est pas la méthode à privilégier, car elle est contraignante à entretenir. À la place, je vous conseille de passer une instance anonyme de `Runnable` dans un `Thread` :

```
public void onClick(View v) {
  new Thread(new Runnable() {
    public void run() {
      int caractere = hamlet.indexOf("Être ou ne pas être");
      v.setText("Cette phrase se trouve au " + caractere + " è
          me caractère.");
    }
  }).start();
}
```

Le problème de notre exemple, c'est que l'opération coûteuse (la recherche d'un texte dans un livre) s'exécute dans un autre thread. C'est une bonne chose, c'est ce qu'on avait demandé, comme ça la recherche se fait sans bloquer le thread UI, mais on remarquera que la vue est aussi manipulée dans un autre thread, ce qui déroge à la seconde règle vue précédemment, qui précise que les vues doivent être manipulées dans le thread UI ! On risque de rencontrer des comportements inattendus ou impossibles à prédire !

Afin de remédier à ce problème, Android offre plusieurs manières d'accéder au thread UI depuis d'autres threads. Par exemple :

- La méthode d'`Activity void runOnUiThread(Runnable action)` spécifie qu'une action doit s'exécuter dans le thread UI. Si le thread actuel est le thread UI, alors l'action est exécutée immédiatement. Sinon, l'action est ajoutée à la pile des évènements du thread UI.
- Sur un `View`, on peut faire `boolean post(Runnable action)` pour ajouter le `Runnable` à la pile des messages du thread UI. Le `boolean` retourné vaut `true` s'il a été correctement placé dans la pile des messages.
- De manière presque similaire, `boolean postDelayed(Runnable action, long delayMillis)` permet d'ajouter un `Runnable` à la pile des messages, mais uniquement après qu'une certaine durée `delayMillis` s'est écoulée.

On peut par exemple voir :

```
public void onClick(View v) {
  new Thread(new Runnable() {
    public void run() {
      int caractere = hamlet.indexOf("Être ou ne pas être");
      v.post(new Runnable() {
        public void run() {
          v.setText("Cette phrase se trouve au " + caractere +
              " ème caractère.");
        }
      });
    }
  }).start();
}
```

Ou :

```
public void onClick(View v) {
  new Thread(new Runnable() {
    public void run() {
      int caractere = hamlet.indexOf("Être ou ne pas être");
      runOnUiThread(new Runnable() {
        public void run() {
          v.setText("Cette phrase se trouve au " + caractere +
              " ème caractère.");
        }
      });
    }
  }).start();
}
```

Ainsi, la longue opération s'exécute dans un thread différent, ce qui est bien, et la vue est manipulée dans le thread UI, ce qui est parfait !

Le problème ici est que ce code peut vite devenir difficile à maintenir. Vous avez vu, pour à peine deux lignes de code à exécuter, on a dix lignes d'enrobage ! Je vais donc vous présenter une solution qui permet un contrôle total tout en étant plus évidente à manipuler.

Les messages et les handlers

La classe `Thread` est une classe de bas niveau et en Java on préfère travailler avec des objets d'un plus haut niveau. Une autre manière d'utiliser les threads est d'utiliser la classe `Handler`, qui est d'un plus haut niveau.

En informatique, « de haut niveau » signifie « qui s'éloigne des contraintes de la machine pour faciliter sa manipulation pour les humains ».

La classe `Handler` contient un mécanisme qui lui permet d'ajouter des messages ou des `Runnable` à une file de messages. Quand vous créez un `Handler`, il sera lié à un thread, c'est donc dans la file de ce thread-là qu'il pourra ajouter des messages. Le thread UI possède lui aussi un handler et chaque handler que vous créerez communiquera par défaut avec ce handler-là.

Les handlers tels que je vais les présenter doivent être utilisés pour effectuer des calculs avant de mettre à jour l'interface graphique, et c'est tout. Ils peuvent être utilisés pour effectuer des calculs et ne pas mettre à jour l'interface graphique après, mais ce n'est pas le comportement attendu.

Mais voyons tout d'abord comment les handlers font pour se transmettre des messages. Ces messages sont représentés par la classe `Message`. Un message peut contenir un `Bundle` avec la méthode `void setData(Bundle data)`. Mais comme vous le savez, un Bundle, c'est lourd, il est alors possible de mettre des objets dans des attributs publics :

- `arg1` et `arg2` peuvent contenir des entiers.
- On peut aussi mettre un `Object` dans `obj`.

Bien que le constructeur de `Message` soit public, la meilleure manière d'en construire un est d'appeler la méthode statique `Message Message.obtain()` ou encore `Message Handler.obtainMessage()`. Ainsi, au lieu d'allouer de nouveaux objets, on récupère des anciens objets qui se trouvent dans le pool de messages. Notez que si vous utilisez la seconde méthode, le handler sera déjà associé au message, mais vous pouvez très bien le mettre *a posteriori* avec `void setTarget(Handler target)`.

```
1 Message msg = Handler.obtainMessage();
2 msg.arg1 = 25;
3 msg.obj = new String("Salut !");
```

Enfin, les méthodes pour planifier des messages sont les suivantes :

- `boolean post(Runnable r)` pour ajouter `r` à la queue des messages. Il s'exécutera sur le `Thread` auquel est rattaché le `Handler`. La méthode renvoie `true` si l'objet a bien été rajouté. De manière alternative, `boolean postAtTime(Runnable r, long uptimeMillis)` permet de lancer un `Runnable` au moment `longMillis` et `boolean postDelayed(Runnable r, long delayMillis)` permet d'ajouter un `Runnable` à lancer après un délai de `delayMillis`.
- `boolean sendEmptyMessage(int what)` permet d'envoyer un `Message` simple qui ne contient que la valeur `what`, qu'on peut utiliser comme un identifiant. On trouve aussi les méthodes `boolean sendEmptyMessageAtTime(int what, long uptimeMillis)` et `boolean sendEmptyMessageDelayed(int what, long delayMillis)`.
- Pour pousser un `Message` complet à la fin de la file des messages, utilisez `boolean sendMessage(Message msg)`. On trouve aussi `boolean sendMessageAtTime(Message msg, long uptimeMillis)` et `boolean sendMessageDelayed(Message msg, long delayMillis)`.

Tous les messages seront reçus dans la méthode de *callback* `void handleMessage(Message msg)` dans le thread auquel est attaché ce `Handler`.

```
1 public class MonHandler extends Handler {
2   @Override
3   public void handleMessage(Message msg) {
4     // Faire quelque chose avec le message
5   }
6 }
```

Application : une barre de progression

Énoncé

Une utilisation typique des handlers est de les incorporer dans la gestion des barres de progression. On va faire une petite application qui ne possède au départ qu'un bouton. Cliquer dessus lance un téléchargement et une boîte de dialogue s'ouvrira. Cette boîte contiendra une barre de progression qui affichera l'avancement du téléchargement,

comme à la figure 18.2. Dès que le téléchargement se termine, la boîte de dialogue se ferme et un `Toast` indique que le téléchargement est terminé. Enfin, si l'utilisateur s'impatiente, il peut très bien fermer la boîte de dialogue avec le bouton `Retour`.

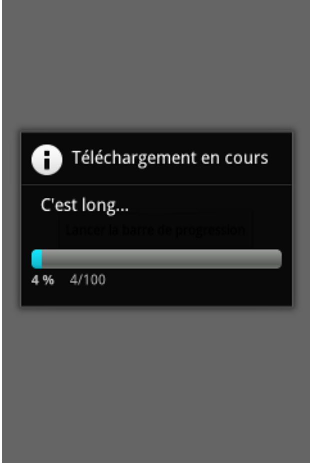


Figure 18.2 – Une barre de progression

Spécifications techniques

On va utiliser un `ProgressDialog` pour afficher la barre de progression. Il s'utilise comme n'importe quelle boîte de dialogue, sauf qu'il faut lui attribuer un style si on veut qu'il affiche une barre de progression. L'attribution se fait avec la méthode `setProgressStyle(int style)` en lui passant le paramètre `ProgressDialog.STYLE_HORIZONTAL`.

L'état d'avancement sera conservé dans un attribut. Comme on ne sait pas faire de téléchargement, l'avancement se fera au travers d'une boucle qui augmentera cet attribut. Bien entendu, on ne fera pas cette boucle dans le thread principal, sinon l'interface graphique sera complètement bloquée ! Alors on lancera un nouveau thread. On passera par un handler pour véhiculer des messages. On répartit donc les rôles ainsi :

- Dans le nouveau thread, on calcule l'état d'avancement, puis on l'envoie au handler à l'aide d'un message.
- Dans le handler, dès qu'on reçoit le message, on met à jour la progression de la barre.

Entre chaque incrémentation de l'avancement, allouez-vous une seconde de répit, sinon votre téléphone va faire la tête. On peut le faire avec :

```
1 try {
2   Thread.sleep(1000);
3 } catch (InterruptedException e) {
4   e.printStackTrace();
5 }
```

Enfin, on peut interrompre un `Thread` avec la méthode `void interrupt()`. Cependant, si votre thread est en train de dormir à cause de la méthode `sleep`, alors l'interruption `InterruptedException` sera lancée et le thread ne s'interrompra pas. À vous de réfléchir pour contourner ce problème.

Ma solution

```
import android.app.Activity;
import android.app.ProgressDialog;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class ProgressBarActivity extends Activity {
  private final static int PROGRESS_DIALOG_ID = 0;
  private final static int MAX_SIZE = 100;
  private final static int PROGRESSION = 0;

  private Button mProgressButton = null;
  private ProgressDialog mProgressBar = null;
  private Thread mProgress = null;

  private int mProgression = 0;

  // Gère les communications avec le thread de téléchargement
  final private Handler mHandler = new Handler(){
    @Override
    public void handleMessage(Message msg) {
      super.handleMessage(msg);
      // L'avancement se situe dans msg.arg1
      mProgressBar.setProgress(msg.arg1);
    }
  };

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_progress_bar);

    mProgressButton = (Button) findViewById(R.id.
        progress_button);
    mProgressButton.setOnClickListener(new View.OnClickListener
        () {
      @Override
      public void onClick(View v) {
        // Initialise la boîte de dialogue
```

```
41          showDialog(PROGRESS_DIALOG_ID);
42
43          // On remet le compteur à zéro
44          mProgression = 0;
45
46          mProgress = new Thread(new Runnable() {
47            public void run() {
48              try {
49                while (mProgression < MAX_SIZE) {
50                  // On télécharge un bout du fichier
51                  mProgression = download();
52
53                  // Repose-toi pendant une seconde
54                  Thread.sleep(1000);
55
56                  Message msg = mHandler.obtainMessage(
                        PROGRESSION, mProgression, 0);
57                  mHandler.sendMessage(msg);
58                }
59
60                // Le fichier a été téléchargé
61                if (mProgression >= MAX_SIZE) {
62                  runOnUiThread(new Runnable() {
63                    @Override
64                    public void run() {
65                      Toast.makeText(ProgressBarActivity.this,
                          ProgressBarActivity.this.getString(R.
                          string.over), Toast.LENGTH_SHORT).show()
                          ;
66                    }
67                  });
68
69                  // Ferme la boîte de dialogue
70                  mProgressBar.dismiss();
71                }
72              } catch (InterruptedException e) {
73                // Si le thread est interrompu, on sort de la
                    boucle de cette manière
74                e.printStackTrace();
75              }
76            }
77          }).start();
78        }
79      });
80    }
81
82    @Override
83    public Dialog onCreateDialog(int identifiant) {
84      if(mProgressBar == null) {
85        mProgressBar = new ProgressDialog(this);
```

```
86          // L'utilisateur peut annuler la boîte de dialogue
87          mProgressBar.setCancelable(true);
88          // Que faire quand l'utilisateur annule la boîte ?
89          mProgressBar.setOnCancelListener(new DialogInterface.
                OnCancelListener() {
90            @Override
91            public void onCancel(DialogInterface dialog) {
92              // On interrompt le thread
93              mProgress.interrupt();
94              Toast.makeText(ProgressBarActivity.this,
                    ProgressBarActivity.this.getString(R.string.
                    canceled), Toast.LENGTH_SHORT).show();
95              removeDialog(PROGRESS_DIALOG_ID);
96            }
97          });
98          mProgressBar.setTitle("Téléchargement en cours");
99          mProgressBar.setMessage("C'est long...");
100         mProgressBar.setProgressStyle(ProgressDialog.
                STYLE_HORIZONTAL);
101         mProgressBar.setMax(MAX_SIZE);
102       }
103       return mProgressBar;
104     }
105
106     public int download() {
107       if(mProgression <= MAX_SIZE) {
108         mProgression++;
109         return mProgression;
110       }
111       return MAX_SIZE;
112     }
113   }
```

Vous pouvez copier ce code grâce au code web suivant :

▷ Copier ce code
Code web : 158133

Sécuriser ses threads

Les threads ne sont pas des choses aisées à manipuler. À partir de notre application précédente, nous allons voir certaines techniques qui vous permettront de gérer les éventuels débordements imputés aux threads.

Il y a une fuite

Une erreur que nous avons commise est d'utiliser le handler en classe interne. Le problème de cette démarche est que quand on déclare une classe interne, alors chaque instance de cette classe contient une référence à la classe externe. Par conséquent, tant qu'il y a des messages sur la pile des messages qui sont liés au handler, l'activité ne

pourra pas être nettoyée par le système, et une activité, ça pèse lourd pour le système !

Une solution simple est de faire une classe externe qui dérive de `Handler`, et de rajouter une instance de cette classe en tant qu'attribut.

```java
import android.app.ProgressDialog;
import android.os.Handler;
import android.os.Message;

public class ProgressHandler extends Handler {
  @Override
  public void handleMessage(Message msg) {
    super.handleMessage(msg);
    ProgressDialog progressBar = (ProgressDialog)msg.obj;
    progressBar.setProgress(msg.arg1);
  }
}
```

Ne pas oublier d'inclure la boîte de dialogue dans le message puisque nous ne sommes plus dans la même classe ! Si vous vouliez vraiment rester dans la même classe, alors vous auriez pu déclarer `ProgressHandler` comme statique de manière à séparer les deux classes.

Gérer le cycle de l'activité

Il faut lier les threads au cycle des activités. On pourrait se dire qu'on veut parfois effectuer des tâches d'arrière-plan même quand l'activité est terminée, mais dans ce cas-là on ne passera pas par des threads mais par des `Services`, qui seront étudiés dans le prochain chapitre.

Le plus important est de gérer le changement de configuration. Pour cela, tout se fait dans `onRetainNonConfigurationInstance()`. On fait en sorte de sauvegarder le thread ainsi que la boîte de dialogue de manière à pouvoir les récupérer :

```java
public Object onRetainNonConfigurationInstance () {
  List<Object> list = new ArrayList<Object>();
  list.add(mProgressBar);
  list.add(mProgress);
  return list;
}
```

Enfin, vous pouvez aussi faire en sorte d'arrêter le thread dès que l'activité passe en pause ou quitte son cycle.

AsyncTask

Il faut avouer que tout cela est bien compliqué et nécessite de penser à tout, ce qui est source de confusion. Je vais donc vous présenter une alternative plus évidente à maîtriser, mais qui est encore une fois réservée à l'interaction avec le thread UI. `AsyncTask`

vous permet de faire proprement et facilement des opérations en parallèle du thread UI. Cette classe permet d'effectuer des opérations d'arrière-plan et de publier les résultats dans le thread UI sans avoir à manipuler de threads ou de handlers.

AsyncTask n'est pas une alternative radicale à la manipulation des threads, juste un substitut qui permet le travail en arrière-plan sans toucher les blocs de bas niveau comme les threads. On va surtout les utiliser pour les opérations courtes (quelques secondes tout au plus) dont on connaît précisément l'heure de départ et de fin.

On ne va pas utiliser directement `AsyncTask`, mais plutôt créer une classe qui en dérivera. Cependant, il ne s'agit pas d'un héritage évident puisqu'il faut préciser trois paramètres :

- Le paramètre `Params` permet de définir le typage des objets sur lesquels on va faire une opération.
- Le deuxième paramètre, `Progress`, indique le typage des objets qui indiqueront l'avancement de l'opération.
- Enfin, `Result` est utilisé pour symboliser le résultat de l'opération.

Ce qui donne dans le contexte :

```
1 public class MaClasse extends AsyncTask<Params, Progress,
      Result>
```

Ensuite, pour lancer un objet de type `MaClasse`, il suffit d'utiliser dessus la méthode `final AsyncTask<Params, Progress, Result> execute (Params... params)` sur laquelle il est possible de faire plusieurs remarques :

- Son prototype est accompagné du mot-clé `final`, ce qui signifie que la méthode ne peut être redéfinie dans les classes dérivées d'`AsyncTask`.
- Elle prend un paramètre de type `Params...` ce qui pourra en troubler plus d'un, j'imagine. Déjà, `Params` est tout simplement le type que nous avons défini auparavant dans la déclaration de `MaClasse`. Ensuite, les trois points signifient qu'il s'agit d'*arguments variables* et que par conséquent on peut en mettre autant qu'on veut. Si on prend l'exemple de la méthode `int somme(int... nombres)`, on peut l'appeler avec `somme(1)` ou `somme(1,5,-2)`. Pour être précis, il s'agit en fait d'un tableau déguisé, vous pouvez donc considérer `nombres` comme un `int[]`.

Une fois cette méthode exécutée, notre classe va lancer quatre méthodes de *callback*, dans cet ordre :

- `void onPreExecute()` est lancée dès le début de l'exécution, avant même que le travail commence. On l'utilise donc pour initialiser les différents éléments qui doivent être initialisés.
- Ensuite, on trouve `Result doInBackground(Params... params)`, c'est dans cette méthode que doit être effectué le travail d'arrière-plan. À la fin, on renvoie le résultat de l'opération et ce résultat sera transmis à la méthode suivante — on utilise souvent un `boolean` pour signaler la réussite ou l'échec de l'opération. Si vous voulez publier une progression pendant l'exécution de cette méthode, vous pouvez le faire

en appelant `final void publishProgress(Progress... values)` (la méthode de *callback* associée étant `void onProgressUpdate(Progress... values)`).
- Enfin, `void onPostExecute(Result result)` permet de conclure l'utilisation de l'`AsyncTask` en fonction du résultat `result` passé en paramètre.

De plus, il est possible d'annuler l'action d'un `AsyncTask` avec `final boolean cancel(boolean mayInterruptIfRunning)`, où `mayInterruptIfRunning` vaut `true` si vous autorisez l'exécution à s'interrompre. Par la suite, une méthode de *callback* est appelée pour que vous puissiez réagir à cet évènement : `void onCancelled()`.

Enfin, dernière chose à savoir, un `AsyncTask` n'est disponible que pour une unique utilisation, s'il s'arrête ou si l'utilisateur l'annule, alors il faut en recréer un nouveau.

Et cette fois on fait comment pour gérer les changements de configuration ?

Ah ! vous aimez avoir mal, j'aime ça. Accrochez-vous parce que ce n'est pas simple. Ce que nous allons voir est assez avancé et de bas niveau, alors essayez de bien comprendre pour ne pas faire de boulettes quand vous l'utiliserez par la suite.

On pourrait garder l'activité qui a lancé l'`AsyncTask` en paramètre, mais de manière générale il ne faut jamais garder de référence à une classe qui dérive de `Context`, par exemple `Activity`. Le problème, c'est qu'on est bien obligés par moment. Alors comment faire ?

Revenons aux bases de la programmation. Quand on crée un objet, on réserve dans la mémoire allouée par le processus un emplacement qui aura la place nécessaire pour mettre l'objet. Pour accéder à l'objet, on utilise une **référence** sous forme d'un identifiant déclaré dans le code :

```
1 String chaine = new String();
```

Ici, `chaine` est l'identifiant, autrement dit une référence qui pointe vers l'emplacement mémoire réservé pour cette chaîne de caractères.

Bien sûr, au fur et à mesure que le programme s'exécute, on va allouer de la place pour d'autres objets et, si on ne fait rien, la mémoire va être saturée. Afin de faire en sorte de libérer de la mémoire, un processus qui s'appelle le *garbage collector* (« ramasse-miettes » en français) va détruire les objets qui ne sont plus susceptibles d'être utilisés :

```
1 String chaine = new String("Rien du tout");
2
3 if(chaine.equals("Quelque chose") {
4   int dix = 10;
5 }
```

La variable `chaine` sera disponible avant, pendant et après le `if` puisqu'elle a été déclarée avant (donc de 1 à 5, voire plus loin encore), en revanche `dix` a été déclaré dans le `if`, il ne sera donc disponible que dedans (donc de 4 à 5). Dès qu'on sort du `if`, le *garbage collector* passe et désalloue la place réservée `dix` de manière à pouvoir

l'allouer à un autre objet.

Quand on crée un objet en Java, il s'agit toujours d'une **référence forte**, c'est-à-dire que l'objet est protégé contre le *garbage collector* tant qu'on est certain que vous l'utilisez encore. Ainsi, si on garde notre activité en référence forte en tant qu'attribut de classe, elle restera toujours disponible, et vous avez bien compris que ce n'était pas une bonne idée, surtout qu'une référence à une activité est bien lourde.

À l'opposé des références fortes se trouvent les **références faibles**. Les références faibles ne protègent pas une référence du *garbage collector*.

Ainsi, si vous avez une référence forte vers un objet, le *garbage collector* ne passera pas dessus. Si vous en avez deux, idem. Si vous avez deux références fortes et une référence faible, c'est la même chose, parce qu'il y a deux références fortes.

Si le *garbage collector* réalise que l'une des deux références fortes n'est plus valide, l'objet est toujours conservé en mémoire puisqu'il reste une référence forte. *En revanche*, dès que la seconde référence forte est invalidée, alors l'espace mémoire est libéré puisqu'il ne reste plus aucune référence forte, juste une petite référence faible qui ne protège pas du ramasse-miettes.

Ainsi, il suffit d'inclure une référence faible vers notre activité dans l'`AsyncTask` pour pouvoir garder une référence vers l'activité sans pour autant la protéger contre le ramasse-miettes.

Pour créer une référence faible d'un objet `T`, on utilise `WeakReference` de cette manière :

```
1  T strongReference = new T();
2  WeakReference<T> weakReference = new WeakReference<T>(
       strongReference);
```

Il n'est bien entendu pas possible d'utiliser directement un `WeakReference`, comme il ne s'agit que d'une référence faible, il vous faut donc récupérer une référence forte de cet objet. Pour ce faire, il suffit d'utiliser `T get()`. Cependant, cette méthode renverra `null` si l'objet a été nettoyé par le *garbage collector*.

Application

Énoncé

Faites exactement comme l'application précédente, mais avec un `AsyncTask` cette fois.

Spécifications techniques

L'`AsyncTask` est utilisé en tant que classe interne statique, de manière à ne pas avoir de fuites comme expliqué dans la partie consacrée aux threads.

Comme un `AsyncTask` n'est disponible qu'une fois, on va en recréer un à chaque fois que l'utilisateur appuie sur le bouton.

Il faut lier une référence faible à votre activité à l'`AsyncTask` pour qu'à chaque fois

que l'activité est détruite on reconstruise une nouvelle référence faible à l'activité dans l'`AsyncTask`. Un bon endroit pour faire cela est dans le `onRetainNonConfigurationInstance()`.

Ma solution

```
package sdz.chapitreQuatre.async.example;

import java.lang.ref.WeakReference;

import android.app.Activity;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class AsyncActivity extends Activity {
  // Taille maximale du téléchargement
  public final static int MAX_SIZE = 100;
  // Identifiant de la boîte de dialogue
  public final static int ID_DIALOG = 0;
  // Bouton qui permet de démarrer le téléchargement
  private Button mBouton = null;

  private ProgressTask mProgress = null;
  // La boîte en elle-même
  private ProgressDialog mDialog = null;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mBouton = (Button) findViewById(R.id.bouton);
    mBouton.setOnClickListener(new View.OnClickListener() {

      @Override
      public void onClick(View arg0) {
        // On recrée à chaque fois l'objet
        mProgress = new ProgressTask(AsyncActivity.this);
        // On l'exécute
        mProgress.execute();
      }
    });

```

```
44      // On recupère l'AsyncTask perdu dans le changement de
            configuration
45      mProgress = (ProgressTask) getLastNonConfigurationInstance
            ();
46
47      if(mProgress != null)
48        // On lie l'activité à l'AsyncTask
49        mProgress.link(this);
50    }
51
52    @Override
53    protected Dialog onCreateDialog (int id) {
54      mDialog = new ProgressDialog(this);
55      mDialog.setCancelable(true);
56      mDialog.setOnCancelListener(new DialogInterface.
            OnCancelListener() {
57        @Override
58        public void onCancel(DialogInterface arg0) {
59          mProgress.cancel(true);
60        }
61      });
62      mDialog.setTitle("Téléchargement en cours");
63      mDialog.setMessage("C'est long...");
64      mDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
65      mDialog.setMax(MAX_SIZE);
66      return mDialog;
67    }
68
69    @Override
70    public Object onRetainNonConfigurationInstance () {
71      return mProgress;
72    }
73
74    // Met à jour l'avancement dans la boîte de dialogue
75    void updateProgress(int progress) {
76      mDialog.setProgress(progress);
77    }
78
79    // L'AsyncTask est bien une classe interne statique
80    static class ProgressTask extends AsyncTask<Void, Integer,
          Boolean> {
81      // Référence faible à l'activité
82      private WeakReference<AsyncActivity> mActivity = null;
83      // Progression du téléchargement
84      private int mProgression = 0;
85
86      public ProgressTask (AsyncActivity pActivity) {
87        link(pActivity);
88      }
89
```

```
90      @Override
91      protected void onPreExecute () {
92        // Au lancement, on affiche la boîte de dialogue
93        if(mActivity.get() != null)
94          mActivity.get().showDialog(ID_DIALOG);
95      }
96
97      @Override
98      protected void onPostExecute (Boolean result) {
99        if (mActivity.get() != null) {
100         if(result)
101           Toast.makeText(mActivity.get(), "Téléchargement
                  terminé", Toast.LENGTH_SHORT).show();
102         else
103           Toast.makeText(mActivity.get(), "Echec du télé
                  chargement", Toast.LENGTH_SHORT).show();
104       }
105     }
106
107     @Override
108     protected Boolean doInBackground (Void... arg0) {
109       try {
110         while(download() != MAX_SIZE) {
111           publishProgress(mProgression);
112           Thread.sleep(1000);
113         }
114         return true;
115       }catch(InterruptedException e) {
116         e.printStackTrace();
117         return false;
118       }
119     }
120
121     @Override
122     protected void onProgressUpdate (Integer... prog) {
123       // À chaque avancement du téléchargement, on met à jour
             la boîte de dialogue
124       if (mActivity.get() != null)
125         mActivity.get().updateProgress(prog[0]);
126     }
127
128     @Override
129     protected void onCancelled () {
130       if(mActivity.get() != null)
131         Toast.makeText(mActivity.get(), "Annulation du télé
                chargement", Toast.LENGTH_SHORT).show();
132     }
133
134     public void link (AsyncActivity pActivity) {
135       mActivity = new WeakReference<AsyncActivity>(pActivity);
```

```
136        }
137
138        public int download() {
139          if(mProgression <= MAX_SIZE) {
140            mProgression++;
141            return mProgression;
142          }
143          return MAX_SIZE;
144        }
145      }
146    }
```

Vous pouvez copier ce code grâce au code web suivant :

▷ Copier ce code
Code web : 991656

Pour terminer, voici une liste de quelques comportements à adopter afin d'éviter les aléas des blocages :

- Si votre application fait en arrière-plan de gros travaux bloquants pour l'interface graphique (imaginez qu'elle doive télécharger une image pour l'afficher à l'utilisateur), alors il suffit de montrer l'avancement de ce travail avec une barre de progression de manière à faire patienter l'utilisateur.
- En ce qui concerne les jeux, usez et abusez des threads pour effectuer des calculs de position, de collision, etc.
- Si votre application a besoin de faire des initialisations au démarrage et que par conséquent elle met du temps à se charger, montrez un *splash screen* avec une barre de progression pour montrer à l'utilisateur que votre application n'est pas bloquée.

En résumé

- Par défaut, au lancement d'une application, le système l'attache à un nouveau processus dans lequel il sera exécuté. Ce processus contiendra tout un tas d'informations relatives à l'état courant de l'application qu'il contient et des threads qui exécutent le code.
- Vous pouvez décider de forcer l'exécution de certains composants dans un processus à part grâce à l'attribut `android:process` à rajouter dans l'un des éléments constituant le noeud `<application>` de votre manifest.
- Lorsque vous jouez avec les threads, vous ne devez jamais perdre à l'esprit deux choses :
  - Ne jamais bloquer le thread UI.
  - Ne pas manipuler les vues standards en dehors du thread UI.
- On préfèrera toujours privilégier les concepts de haut niveau pour faciliter les manipulations pour l'humain et ainsi donner un niveau d'abstraction aux contraintes machines. Pour les threads, vous pouvez donc privilégier les messages et les handlers à l'utilisation directe de la classe `Thread`.
- `AsyncTask` est un niveau d'abstraction encore supérieur aux messages et handlers.

Elle permet d'effectuer des opérations en arrière-plan et de publier les résultats dans le thread UI facilement grâce aux méthodes qu'elle met à disposition des développeurs lorsque vous en créez une.

Chapitre 19

Les services

Difficulté :

Nous savons désormais faire du travail en arrière-plan, mais de manière assez limitée quand même. En effet, toutes les techniques que nous avons vues étaient destinées aux opérations courtes et/ou en interaction avec l'interface graphique, or ce n'est pas le cas de toutes les opérations d'arrière-plan. C'est pourquoi nous allons voir le troisième composant qui peut faire partie d'une application : les services.

Contrairement aux threads, les services sont conçus pour être utilisés sur une longue période de temps. En effet, les threads sont des éléments sommaires qui n'ont pas de lien particulier avec le système Android, alors que les services sont des composants et sont par conséquent intégrés dans Android au même titre que les activités. Ainsi, ils vivent au même rythme que l'application. Si l'application s'arrête, le service peut réagir en conséquence, alors qu'un thread, qui n'est pas un composant d'Android, ne sera pas mis au courant que l'application a été arrêtée si vous ne lui dites pas. Il ne sera par conséquent pas capable d'avoir un comportement approprié, c'est-à-dire la plupart du temps de s'arrêter.

Qu'est-ce qu'un service ?

Tout comme les activités, les services possèdent un cycle de vie ainsi qu'un contexte qui contient des informations spécifiques sur l'application et qui constitue une interface de communication avec le restant du système. Ainsi, on peut dire que les services sont des composants très proches des activités (et beaucoup moins des receivers, qui eux ne possèdent pas de contexte). Cette configuration leur prodigue la même grande flexibilité que les activités. En revanche, à l'opposé des activités, les services ne possèdent pas d'interface graphique : c'est pourquoi on les utilise pour effectuer des travaux d'arrière-plan.

Un exemple typique est celui du lecteur de musique. Vous laissez à l'utilisateur l'opportunité de choisir une chanson à l'aide d'une interface graphique dans une activité, puis il est possible de manipuler la chanson dans une seconde activité qui nous montre un joli lecteur avec des commandes pour modifier le volume ou mettre en pause. Mais si l'utilisateur veut regarder une page web en écoutant la musique ? Comme une activité a besoin d'afficher une interface graphique, il est impossible que l'utilisateur regarde autre chose que le lecteur quand il écoute la musique. On pourrait éventuellement envisager de passer par un receiver, mais celui-ci devrait résoudre son exécution en dix secondes, ce n'est donc pas l'idéal pour un lecteur. La solution la plus évidente est bien sûr de faire jouer la musique par un service, comme ça votre client pourra utiliser une autre application sans pour autant que la musique s'interrompe. Un autre exemple est celui du lecteur d'e-mails qui va vérifier ponctuellement si vous avez reçu un nouvel e-mail.

Il existe deux types de services :

- Les plus courants sont les services *locaux* (on trouve aussi le terme *started* ou *unbound service*), où l'activité qui lance le service et le service en lui-même appartiennent à la même application.
- Il est aussi possible qu'un service soit lancé par un composant qui appartient à une autre application, auquel cas il s'agit d'un service *distant* (on trouve aussi le terme *bound* (lié/attaché) *service*). Dans ce cas de figure, il existe toujours une interface qui permet la communication entre le processus qui a appelé le service et le processus dans lequel s'exécute le service. Cette communication permet d'envoyer des requêtes ou récupérer des résultats par exemple. Le fait de communiquer entre plusieurs processus s'appelle l'IPC (InterProcess Communication). Il peut bien sûr y avoir plusieurs clients liés à un service.
- Il est aussi possible que le service expérimente les deux statuts à la fois. Ainsi, on peut lancer un service local et lui permettre d'accepter les connexions distantes par la suite.

Vous vous en doutez peut-être, mais un service se lance par défaut dans le même processus que celui du composant qui l'a appelé. Ce qui peut sembler plus étrange et qui pourrait vous troubler, c'est que les services s'exécutent dans le thread UI. D'ailleurs, ils ne sont pas conçus pour être exécutés en dehors de ce thread, alors n'essayez pas de le délocaliser. En revanche, si les opérations que vous allez mener dans le service risquent d'affecter l'interface graphique, vous pouvez très bien lancer un thread *dans* le service. Vous voyez la différence ? Toujours lancer un service depuis le thread principal ; mais vous pouvez très bien lancer des threads dans le service.

Gérer le cycle de vie d'un service

De manière analogue aux activités, les services traversent plusieurs étapes pendant leur vie et la transition entre ces étapes est matérialisée par des méthodes de *callback*. Heureusement, le cycle des services est plus facile à maîtriser que celui des activités puisqu'il y a beaucoup moins d'étapes. La figure 19.1 est un schéma qui résume ce fonctionnement.

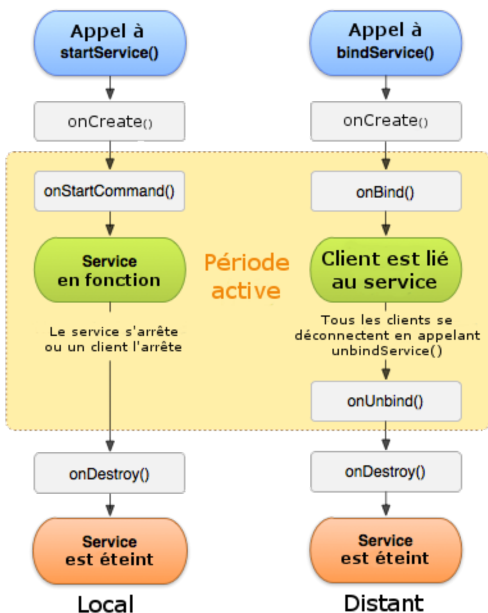


FIGURE 19.1 – Ce cycle est indépendant du cycle du composant qui a lancé le service

Vous voyez qu'on a deux cycles légèrement différents : si le service est local (lancé depuis l'application) ou distant (lancé depuis un processus différent).

Les services locaux

Ils sont lancés à partir d'une activité avec la méthode `ComponentName startService(Intent service)`. La variable retournée donne le package accompagné du nom du composant qui vient d'être lancé.

```
1 Intent intent = new Intent(Activite.this, UnService.class);
2 startService(intent);
```

Si le service n'existait pas auparavant, alors il sera créé. Or, la création d'un service est symbolisée par la méthode de *callback* `void onCreate()`. La méthode qui est appelée ensuite est `int onStartCommand(Intent intent, int flags, int startId)`.

Notez par ailleurs que, si le service existait déjà au moment où vous en demandez la création avec `startService()`, alors `onStartCommande()` est appelée directement sans passer par `onCreate()`.

En ce qui concerne les paramètres, on trouve `intent`, qui a lancé le service, `flags`, dont nous discuterons juste après, et enfin `startId`, pour identifier le lancement (s'il s'agit du premier lancement du service, `startId` vaut 1, s'il s'agit du deuxième lancement, il vaut 2, etc.).

Ensuite comme vous pouvez le voir, cette méthode retourne un entier. Cet entier doit en fait être une constante qui détermine ce que fera le système s'il est tué.

START_NOT_STICKY

Si le système tue le service, alors ce dernier ne sera pas recréé. Il faudra donc effectuer un nouvel appel à `startService()` pour relancer le service.

Ce mode vaut le coup dès qu'on veut faire un travail qui peut être interrompu si le système manque de mémoire et que vous pouvez le redémarrer explicitement par la suite pour recommencer le travail. Si vous voulez par exemple mettre en ligne des statistiques sur un serveur distant. Le processus qui lancera la mise en ligne peut se dérouler toutes les 30 minutes, mais, si le service est tué avant que la mise en ligne soit effectuée, ce n'est pas grave, on le fera dans 30 minutes.

START_STICKY

Cette fois, si le système doit tuer le service, alors il sera recréé mais sans lui fournir le dernier `Intent` qui l'avait lancé. Ainsi, le paramètre `intent` vaudra `null`. Ce mode de fonctionnement est utile pour les services qui fonctionnent par intermittence, comme par exemple quand on joue de la musique.

START_REDELIVER_INTENT

Si le système tue le service, il sera recréé et dans onStartCommand() le paramètre intent sera identique au dernier intent qui a été fourni au service. START_REDELIVER_INTENT est indispensable si vous voulez être certains qu'un service effectuera un travail complètement.

Revenons maintenant au dernier paramètre de onStartCommand(), flags. Il nous permet en fait d'en savoir plus sur la nature de l'intent qui a lancé le service :

- 0 s'il n'y a rien de spécial à dire.
- START_FLAG_REDELIVERY si l'intent avait déjà été délivré et qu'il l'est à nouveau parce que le service avait été interrompu.
- Enfin vous trouverez aussi START_FLAG_RETRY si le service redémarre alors qu'il s'était terminé de manière anormale.

Enfin, il faut faire attention parce que flags n'est pas un paramètre simple à maîtriser. En effet, il peut très bien valoir START_FLAG_REDELIVERY et START_FLAG_RETRY en même temps ! Alors comment ce miracle peut-il se produire ? Laissez-moi le temps de faire une petite digression qui vous servira à chaque fois que vous aurez à manipuler des flags, aussi appelés drapeaux.

Vous savez écrire les nombres sous la forme décimale : « 0, 1, 2, 3, 4 » et ainsi de suite. On parle de numération décimale, car il y a dix unités de 0 à 9. Vous savez aussi écrire les nombres sous la forme hexadécimale : « 0, 1, 2, 3, ..., 8, 9, A, B, C, D, E, F, 10, 11, 12, ..., 19, 1A, 1B », et ainsi de suite. Ici, il y a seize unités de 0 à F, on parle donc d'hexadécimal. Il existe une infinité de systèmes du genre, ici nous allons nous intéresser au système binaire qui n'a que deux unités : 0 et 1. On compte donc ainsi : « 0, 1, 10, 11, 100, 101, 110, 111, 1000 », etc.

Nos trois flags précédents valent en décimal (et dans l'ordre de la liste précédente) 0, 1 et 2, ce qui fait en binaire 0, 1 et 10. Ainsi, si flags contient START_FLAG_REDELIVERY et START_FLAG_RETRY, alors il vaudra 1 + 2 = 3, soit en binaire 1 + 10 = 11. Vous pouvez voir qu'en fait chaque 1 correspond à la présence d'un flag : le premier à droite dénote la présence de START_FLAG_REDELIVERY (car START_FLAG_REDELIVERY vaut 1) et le plus à gauche celui de START_FLAG_RETRY (car START_FLAG_RETRY vaut 10).

On remarque tout de suite que le binaire est pratique puisqu'il permet de savoir quel flag est présent en fonction de l'absence ou non d'un 1. Mais comment demander à Java quels sont les 1 présents dans flags ? Il existe deux opérations de base sur les nombres binaires : le « ET » (« & ») et le « OU » (« | »). Le « ET » permet de demander « Est-ce que ce flag est présent dans flags ou pas ? », car il permet de vérifier que deux bits sont similaires. Imaginez, on ignore la valeur de flags (qui vaut « YX », on va dire) et on se demande s'il contient START_FLAG_REDELIVERY (qui vaut 1, soit 01 sur deux chiffres). On va alors poser l'opération comme vous le faites d'habitude :

```
flags     YX
       &  01
          ------
Résultat 0X
```

Le résultat fait « 0X » et en fonction de X on saura si `flags` contient ou non `START_FLAG_REDELIVERY` :

- Si X vaut 0, alors `flags` ne contient pas `START_FLAG_REDELIVERY`.
- Si X vaut 1, alors il contient `START_FLAG_REDELIVERY`.

Il suffit maintenant de vérifier la valeur du résultat : s'il vaut 0, c'est que le flag n'est pas présent !

En Java, on peut le savoir de cette manière :

```
1 if((flags & Service.START_FLAG_REDELIVERY) != 0)
2   // Ici, START_FLAG_REDELIVERY est présent dans flags
3 else
4   // Ici, START_FLAG_REDELIVERY n'est pas présent dans flags
```

Je vais maintenant vous parler du « OU ». Il permet d'ajouter un flag à un nombre binaire s'il n'était pas présent auparavant :

```
flags     YX
       |  10
       ------
Résultat  1X
```

Quelle que soit la valeur précédente de `flags`, il contient désormais `START_FLAG_RETRY`. Ainsi, si on veut vérifier qu'il ait `START_FLAG_REDELIVERY` et `START_FLAG_RETRY`, on fera :

```
1 if((flags & (Service.START_FLAG_REDELIVERY | Service.
     START_FLAG_RETRY) != 0)
2   // Les deux flags sont présents
3 else
4   // Il manque un des deux flags (voire les deux)
5 }
```

J'espère que vous avez bien compris le concept de flags parce qu'on le retrouve souvent en programmation. Les flags permettent même d'optimiser quelque peu certains calculs pour les fous furieux, mais cela ne rentre pas dans le cadre de ce cours.

Une fois sorti de la méthode `onStartCommand()`, le service est lancé. Un service continuera à fonctionner jusqu'à ce que vous l'arrêtiez ou qu'Android le fasse de lui-même pour libérer de la mémoire RAM, comme pour les activités. Au niveau des priorités, les services sont plus susceptibles d'être détruits qu'une activité située au premier plan, mais plus prioritaires que les autres processus qui ne sont pas visibles. La priorité a néanmoins tendance à diminuer avec le temps : plus un service est lancé depuis longtemps, plus il a de risques d'être détruit. De manière générale, on va apprendre à concevoir nos services de manière à ce qu'ils puissent gérer la destruction et le redémarrage.

Pour arrêter un service, il est possible d'utiliser `void stopSelf()` depuis le service ou `boolean stopService(Intent service)` depuis une activité, auquel cas il faut fournir `service` qui décrit le service à arrêter.

Cependant, si votre implémentation du service permet de gérer une accumulation de requêtes (un pool de requêtes), vous pourriez vouloir faire en sorte de ne pas interrompre le service avant que toutes les requêtes aient été gérées, même les nouvelles. Pour éviter ce cas de figure, on peut utiliser `boolean stopSelfResult(int startId)` où `startId` correspond au même `startId` qui était fourni à `onStartCommand()`. On l'utilise de cette manière : vous lui passez un `startId` et, s'il est identique au dernier `startId` passé à `onStartCommand()`, alors le service s'interrompt. Sinon, c'est qu'il a reçu une nouvelle requête et qu'il faudra la gérer avant d'arrêter le service.

Comme pour les activités, si on fait une initialisation qui a lieu dans `onCreate()` et qui doit être détruite par la suite, alors on le fera dans le `onDestroy()`. De plus, si un service est détruit par manque de RAM, alors le système ne passera pas par la méthode `onDestroy()`.

Les services distants

Comme les deux types de services sont assez similaires, je ne vais présenter ici que les différences.

On utilisera cette fois `boolean bindService(Intent service, ServiceConnection conn, int flags)` afin d'assurer une connexion persistante avec le service. Le seul paramètre que vous ne connaissez pas est `conn` qui permet de recevoir le service quand celui-ci démarrera et permet de savoir s'il meurt ou s'il redémarre.

Un `ServiceConnection` est une interface pour surveiller l'exécution du service distant et il incarne le pendant client de la connexion. Il existe deux méthodes de *callback* que vous devrez redéfinir :

1. `void onServiceConnected(ComponentName name, IBinder service)` qui est appelée quand la connexion au service est établie, avec un `IBinder` qui correspond à un canal de connexion avec le service.
2. `void onServiceDisconnected(ComponentName name)` qui est appelée quand la connexion au service est perdue, en général parce que le processus qui accueille le service a planté ou a été tué.

Mais qu'est-ce qu'un `IBinder` ? Comme je l'ai déjà dit, il s'agit d'un pont entre votre service et l'activité, mais au niveau du service. Les `IBinder` permettent au client de demander des choses au service. Alors, comment créer cette interface ? Tout d'abord, il faut savoir que le `IBinder` qui sera donné à `onServiceConnected(ComponentName, IBinder)` est envoyé par la méthode de *callback* `IBinder onBind(Intent intent)` dans `Service`. Maintenant, il suffit de créer un `IBinder`. Nous allons voir la méthode

la plus simple, qui consiste à permettre à l'`IBinder` de renvoyer directement le `Service` de manière à pouvoir effectuer des commandes dessus.

```
public class MonService extends Service {
  // Attribut de type IBinder
  private final IBinder mBinder = new MonBinder();

  // Le Binder est représenté par une classe interne
  public class MonBinder extends Binder {
    // Le Binder possède une méthode pour renvoyer le Service
    MonService getService() {
      return MonService.this;
    }
  }

  @Override
  public IBinder onBind(Intent intent) {
    return mBinder;
  }
}
```

Le service sera créé s'il n'était pas déjà lancé (appel à `onCreate()` donc), mais ne passera pas par `onStartCommand()`.

Pour se détacher d'un service, un client peut utiliser la méthode de `Context`, avec `conn` l'interface de connexion fournie précédemment à `bindService()`, comme ceci :

```
void unbindService(ServiceConnection conn)
```

Ainsi, voici une implémentation typique d'un service distant :

```
// Retient l'état de la connexion avec le service
private boolean mBound = false;
// Le service en lui-même
private MonService mService;
// Interface de connexion au service
private ServiceConnection mConnexion = new ServiceConnection()
   {
  // Se déclenche quand l'activité se connecte au service
  public void onServiceConnected(ComponentName className,
      IBinder service) {
    mService = ((MonService.MonBinder)service).getService();
  }

  // Se déclenche dès que le service est déconnecté
  public void onServiceDisconnected(ComponentName className) {
    mService = null;
  }
};

@Override
protected void onStart() {
```

```
    super.onStart();
    Intent mIntent = new Intent(this, MonService.class);
    bindService(mIntent, mConnexion, BIND_AUTO_CREATE);
    mBound = true;
  }

  @Override
  protected void onStop() {
    super.onStop();
    if(mBound) {
      unbindService(mConnexion);
      mBound = false;
    }
  }
```

À noter aussi que, s'il s'agit d'un service distant, alors il aura une priorité supérieure ou égale à la priorité de son client le plus important (avec la plus haute priorité). Ainsi, s'il est lié à un client qui se trouve au premier plan, il y a peu de risques qu'il soit détruit.

Créer un service

Dans le Manifest

Tout d'abord, il faut déclarer le service dans le Manifest. Il peut prendre quelques attributs que vous connaissez déjà tels que `android:name` qui est indispensable pour préciser son identifiant, `android:icon` pour indiquer un drawable qui jouera le rôle d'icône, `android:permission` pour créer une permission nécessaire à l'exécution du service ou encore `android:process` afin de préciser dans quel processus se lancera ce service. Encore une fois, `android:name` est le seul attribut indispensable :

```
<service android:name="MusicService"
  android:process=":player" >
 ...
</service>
```

De cette manière, le service se lancera dans un processus différent du reste de l'application et ne monopolisera pas le thread UI. Vous pouvez aussi déclarer des filtres d'intents pour savoir quels intents implicites peuvent démarrer votre service.

En Java

Il existe deux classes principales depuis lesquelles vous pouvez dériver pour créer un service.

Le plus générique : `Service`

La classe `Service` permet de créer un service de base. Le code sera alors exécuté dans le thread principal, alors ce sera à vous de créer un nouveau thread pour ne pas engorger le thread UI.

Le plus pratique : `IntentService`

En revanche la classe `IntentService` va créer elle-même un thread et gérer les requêtes que vous lui enverrez dans une file. À chaque fois que vous utiliserez `startService()` pour lancer ce service, la requête sera ajoutée à la file et tous les éléments de la file seront traités par ordre d'arrivée. Le service s'arrêtera dès que la file sera vide. Usez et abusez de cette classe, parce que la plupart des services n'ont pas besoin d'exécuter toutes les requêtes en même temps, mais plutôt les unes après les autres. En plus, elle est plus facile à gérer puisque vous aurez juste à implémenter `void onHandleIntent(Intent intent)` qui recevra toutes les requêtes dans l'ordre sous la forme d'`intent`, ainsi qu'un constructeur qui fait appel au constructeur d'`IntentService` :

```
1  public class ExampleService extends IntentService {
2    public ExampleService() {
3      // Il faut passer une chaîne de caractères au
          superconstructeur
4      super("UnNomAuHasard");
5    }
6
7    @Override
8    protected void onHandleIntent(Intent intent) {
9      // Gérer la requête
10   }
11 }
```

Vous pouvez aussi implémenter les autres méthodes de *callback*, mais faites toujours appel à leur superimplémentation, sinon votre service échouera lamentablement :

```
1  @Override
2  public int onStartCommand(Intent intent, int flags, int startId
      ) {
3    // Du code
4    return super.onStartCommand(intent, flags, startId);
5  }
```

On veut un exemple, on veut un exemple !

Je vous propose de créer une activité qui va envoyer un chiffre à un `IntentService` qui va afficher la valeur de ce chiffre dans la console. De plus, l'`IntentService` fera un long

traitement pour que chaque fois que l'activité envoie un chiffre les intents s'accumulent, ce qui fera que les messages seront retardés dans la console.

J'ai une activité toute simple qui se lance au démarrage de l'application :

```
package sdz.chapitreQuatre.intentservice.example;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {
  private Button mBouton = null;
  private TextView mAffichageCompteur = null;

  private int mCompteur = 0;

  public final static String EXTRA_COMPTEUR = "sdz.
      chapitreQuatre.intentservice.example.compteur";

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mAffichageCompteur = (TextView) findViewById(R.id.affichage
        );

    mBouton = (Button) findViewById(R.id.bouton);
    mBouton.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Intent i = new Intent(MainActivity.this,
            IntentServiceExample.class);
        i.putExtra(EXTRA_COMPTEUR, mCompteur);

        mCompteur ++;
        mAffichageCompteur.setText("" + mCompteur);

        startService(i);
      }
    });
  }
}
```

Cliquer sur le bouton incrémente le compteur et envoie un intent qui lance un service qui s'appelle `IntentServiceExample`. L'intent est ensuite reçu et traité :

```
package sdz.chapitreQuatre.intentservice.example;
```

```
2
3 import android.app.IntentService;
4 import android.content.Intent;
5 import android.util.Log;
6
7 public class IntentServiceExample extends IntentService {
8   private final static String TAG = "IntentServiceExample";
9
10   public IntentServiceExample() {
11     super(TAG);
12   }
13
14   @Override
15   protected void onHandleIntent(Intent intent) {
16     Log.d(TAG, "Le compteur valait : " + intent.getIntExtra(
         MainActivity.EXTRA_COMPTEUR, -1));
17     int i = 0;
18     // Cette boucle permet de rajouter artificiellement du
         temps de traitement
19     while(i < 100000000)
20       i++;
21   }
22 }
```

Allez-y maintenant, cliquez sur le bouton. La première fois, le chiffre s'affichera immédiatement dans la console, mais si vous continuez vous verrez que le compteur augmente, et pas l'affichage, tout simplement parce que le traitement prend du temps et que l'affichage est retardé entre chaque pression du bouton. Cependant, chaque intent est traité, dans l'ordre d'envoi.

Les notifications et services de premier plan

Distribuer des autorisations

Les `PendingIntents` sont des `Intents` avec un objectif un peu particulier. Vous les créez dans votre application, et ils sont destinés à une autre application, jusque là rien de très neuf sous le soleil ! Cependant, en donnant à une autre application un `PendingIntent`, vous lui donnez les droits d'effectuer une opération comme s'il s'agissait de votre application (avec les mêmes permissions et la même identité).

En d'autres termes, vous avez deux applications : celle de départ, celle d'arrivée. Vous donnez à l'application d'arrivée tous les renseignements et toutes les autorisations nécessaires pour qu'elle puisse demander à l'application de départ d'exécuter une action à sa place.

Comment peut-on indiquer une action à effectuer ?

Vous connaissez déjà la réponse, j'en suis sûr ! On va insérer dans le `PendingIntent`... un autre `Intent`, qui décrit l'action qui sera à entreprendre. Le seul but du `PendingIntent` est d'être véhiculé entre les deux applications (ce n'est donc pas surprenant que cette classe implémente `Parcelable`), pas de lancer un autre composant.

Il existe trois manières d'appeler un `PendingIntent` en fonction du composant que vous souhaitez démarrer. Ainsi, on utilisera l'une des méthodes statiques suivantes :

```
PendingIntent PendingIntent.getActivity(Context context, int
    requestCode, Intent intent, int flags);

PendingIntent PendingIntent.getBroadcast(Context context, int
    requestCode, Intent intent, int flags);

PendingIntent PendingIntent.getService(Context context, int
    requestCode, Intent intent, int flags);
```

Comme vous l'aurez remarqué, les paramètres sont toujours les mêmes :

- `context` est le contexte dans lequel le `PendingIntent` devrait démarrer le composant.
- `requestCode` est un code qui n'est pas utilisé.
- `intent` décrit le composant à lancer (dans le cas d'une activité ou d'un service) ou l'`Intent` qui sera diffusé (dans le cas d'un `broadcast`).
- `flags` est également assez peu utilisé.

Le `PendingIntent` sera ensuite délivré au composant destinataire comme n'importe quel autre `Intent` qui aurait été appelé avec `startActivityForResult()` : le résultat sera donc accessible dans la méthode de *callback* `onActivityResult()`.

Voici un exemple qui montre un `PendingIntent` qui sera utilisé pour revenir vers l'activité principale :

```
package sdz.chapitreQuatre.pending.example;

import android.app.Activity;
import android.app.PendingIntent;
import android.content.ComponentName;
import android.content.Intent;
import android.os.Bundle;

public class MainActivity extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Intent explicite qui sera utilisé pour lancer à nouveau
        MainActivity
    Intent intent = new Intent();
    // On pointe vers l'activité courante en précisant le
```

```
        package, puis l'activité
18      intent.setComponent(new ComponentName("sdz.chapitreQuatre.
            pending.example", "sdz.chapitreQuatre.pending.example.
            MainActivity"));
19
20      PendingIntent mPending = PendingIntent.getService(this, 0,
            intent, 0);
21    }
22  }
```

Notifications

Une fois lancé, un service peut avertir l'utilisateur des évènements avec les `Toasts` ou des notifications dans la barre de statut, comme à la figure 19.2.

FIGURE 19.2 – Ma barre de statut contient déjà deux notifications représentées par deux icônes à gauche

Comme vous connaissez les `Toasts` mieux que certaines personnes chez Google, je ne vais parler que des notifications.

Une notification n'est pas qu'une icône dans la barre de statut, en fait elle traverse trois étapes :

1. Tout d'abord, à son arrivée, elle affiche une icône ainsi qu'un texte court que Google appelle bizaremment un « texte de téléscripteur ».
2. Ensuite, seule l'icône est lisible dans la barre de statut après quelques secondes.
3. Puis il est possible d'avoir plus de détails sur la notification en ouvrant la liste des notifications, auquel cas on peut voir une icône, un titre, un texte et un horaire de réception.

Si l'utilisateur déploie la liste des notifications et appuie sur l'une d'elles, Android actionnera un `PendingIntent` qui est contenu dans la notification et qui sera utilisé pour lancer un composant (souvent une activité, puisque l'utilisateur s'attendra à pouvoir effectuer quelque chose). Vous pouvez aussi configurer la notification pour qu'elle s'accompagne d'un son, d'une vibration ou d'un clignotement de la LED.

Les notifications sont des instances de la classe `Notification`. Cette classe permet de définir les propriétés de la notification, comme l'icône, le message associé, le son à jouer, les vibrations à effectuer, etc.

Il existe un constructeur qui permet d'ajouter les éléments de base à une notification : `Notification(int icon, CharSequence tickerText, long when)` où `icon` est une référence à un `Drawable` qui sera utilisé comme icône, `tickerText` est le texte de type téléscripteur qui sera affiché dans la barre de statut, alors que `when` permet d'indiquer la

date et l'heure qui accompagneront la notification. Par exemple, pour une notification lancée dès qu'on appuie sur un bouton, on pourrait avoir :

```
// L'icône sera une petite loupe
int icon = R.drawable.ic_action_search;
// Le premier titre affiché
CharSequence tickerText = "Titre de la notification";
// Daté de maintenant
long when = System.currentTimeMillis();
```

La figure 19.3 représente la barre de statut avant la notification.

FIGURE 19.3 – Avant la notification

La figure 19.4 représente la barre de statut au moment où l'on reçoit la notification.

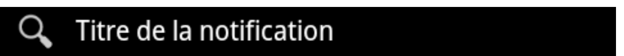


FIGURE 19.4 – Au moment de la notification

Ajouter du contenu à une notification

Une notification n'est pas qu'une icône et un léger texte dans la barre de statut, il est possible d'avoir plus d'informations quand on l'affiche dans son intégralité et elle *doit* afficher du contenu, au minimum un titre et un texte, comme à la figure 19.5.

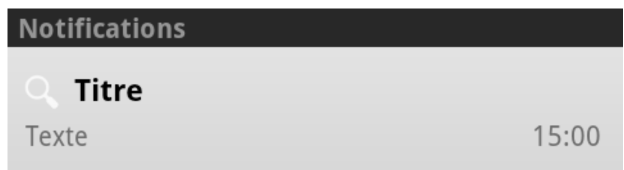


FIGURE 19.5 – La notification contient au moins un titre et un texte

De plus, il faut définir ce qui va se produire dès que l'utilisateur cliquera sur la notification. Nous allons rajouter un `PendingIntent` à la notification, et dès que l'utilisateur cliquera sur la notification, l'intent à l'intérieur de la notification sera déclenché.

Notez bien que, si l'intent lance une activité, alors il faut lui rajouter le flag `FLAG_ACTIVITY_NEW_TASK`. Ces trois composants, titre, texte et `PendingIntent` sont à définir avec la méthode `void setLatestEventInfo(Context context, CharSequence contentTitle, CharSequence contentText, PendingIntent contentIntent)`, où `contentTitle` sera le titre affiché et `contentText`, le texte. Par exemple, pour une notification qui fait retourner dans la même activité que celle qui a lancé la notification :

```
// L'icône sera une petite loupe
int icon = R.drawable.ic_action_search;
// Le premier titre affiché
CharSequence tickerText = "Titre de la notification";
// Daté de maintenant
long when = System.currentTimeMillis();

// La notification est créée
Notification notification = new Notification(icon, tickerText,
    when);

// Intent qui lancera vers l'activité MainActivity
Intent notificationIntent = new Intent(MainActivity.this,
    MainActivity.class);
notificationIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

PendingIntent contentIntent = PendingIntent.getActivity(
    MainActivity.this, 0, notificationIntent, 0);

notification.setLatestEventInfo(MainActivity.this, "Titre", "
    Texte", contentIntent);
```

Enfin, il est possible de rajouter des flags à une notification afin de modifier son comportement :

- FLAG_AUTO_CANCEL pour que la notification disparaisse dès que l'utilisateur appuie dessus.
- FLAG_ONGOING_EVENT pour que la notification soit rangée sous la catégorie « En cours » dans l'écran des notifications, comme à la figure 19.6. Ainsi, l'utilisateur saura que le composant qui a affiché cette notification est en train de faire une opération.

Figure 19.6 – La notification est rangée sous la catégorie « En cours » dans l'écran des notifications

Les flags s'ajoutent à l'aide de l'attribut `flags` qu'on trouve dans chaque notification :

```
notification.flags = FLAG_AUTO_CANCEL | FLAG_ONGOING_EVENT;
```

Gérer vos notifications

Votre application n'est pas la seule à envoyer des notifications, toutes les applications peuvent le faire ! Ainsi, pour gérer toutes les notifications de toutes les applications, Android fait appel à un gestionnaire de notifications, représenté par la classe `NotificationManager`. Comme il n'y a qu'un `NotificationManager` pour tout le système, on ne va pas en construire un nouveau, on va plutôt récupérer celui du système avec une méthode qui appartient à la classe suivante :
`Context` : `Object getSystemService(Context.NOTIFICATION_SERVICE)`. Alors réfléchissons : cette méthode appartient à `Context`, pouvez-vous en déduire quels sont les composants qui peuvent invoquer le `NotificationManager` ? Eh bien, les `Broadcast Receiver` n'ont pas de contexte, alors ce n'est pas possible. En revanche, les activités et les services peuvent le faire !

Il est ensuite possible d'envoyer une notification avec la méthode `void notify(int id, Notification notification)` où `id` sera un identifiant unique pour la notification et où on devra insérer la `notification`.

Ainsi, voici le code complet de notre application qui envoie une notification pour que l'utilisateur puisse la relancer en cliquant sur une notification :

```
import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity {
  public int ID_NOTIFICATION = 0;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Button b = (Button) findViewById(R.id.launch);
    b.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        // L'icône sera une petite loupe
        int icon = R.drawable.ic_action_search;
        // Le premier titre affiché
        CharSequence tickerText = "Titre de la notification";
        // Daté de maintenant
        long when = System.currentTimeMillis();

```

```
30            // La notification est créée
31            Notification notification = new Notification(icon,
                  tickerText, when);
32
33            Intent notificationIntent = new Intent(MainActivity.
                  this, MainActivity.class);
34            notificationIntent.addFlags(Intent.
                  FLAG_ACTIVITY_NEW_TASK);
35            PendingIntent contentIntent = PendingIntent.getActivity
                  (MainActivity.this, 0, notificationIntent, 0);
36
37            notification.setLatestEventInfo(MainActivity.this, "
                  Titre", "Texte", contentIntent);
38
39            // Récupération du Notification Manager
40            NotificationManager manager = (NotificationManager)
                  getSystemService(Context.NOTIFICATION_SERVICE);
41
42            manager.notify(ID_NOTIFICATION, notification);
43          }
44        });
45      }
46    }
```

Les services de premier plan

Pourquoi avons-nous appris tout cela ? Cela n'a pas grand-chose à voir avec les services ! En fait, tout ce que nous avons appris pourra être utilisé pour manipuler des services de premier plan.

Mais cela n'a pas de sens, pourquoi voudrait-on que nos services soient au premier plan ?

Et pourquoi pas ? En fait, parler d'un service de premier plan est un abus de langage, parce que ce type de services reste un service, il n'a pas d'interface graphique, en revanche il a la même priorité qu'une activité consultée par un utilisateur, c'est-à-dire la priorité maximale. Il est donc peu probable que le système le ferme.

Il faut cependant être prudent quand on les utilise. En effet, ils ne sont pas destinés à tous les usages. On ne fait appel aux services de premier plan que si l'utilisateur sait pertinemment qu'il y a un travail en cours qu'il ne peut pas visualiser, tout en lui laissant des contrôles sur ce travail pour qu'il puisse intervenir de manière permanente. C'est pourquoi on utilise une notification qui sera une passerelle entre votre service et l'utilisateur. Cette notification devra permettre à l'utilisateur d'ouvrir des contrôles dans une activité pour arrêter le service.

Par exemple, un lecteur multimédia qui joue de la musique depuis un service devrait

s'exécuter sur le premier plan, de façon à ce que l'utilisateur soit conscient de son exécution. La notification pourrait afficher le titre de la chanson, son avancement et permettre à l'utilisateur d'accéder aux contrôles dans une activité.

Pour qu'un service se lance au premier plan, on appelle `void startForeground(int id, Notification notification)`. Comme vous pouvez le voir, vous devez fournir un identifiant pour la notification avec `id`, ainsi que la `notification` à afficher.

```
1  import android.app.Activity;
2  import android.app.Notification;
3  import android.app.NotificationManager;
4  import android.app.PendingIntent;
5  import android.content.Context;
6  import android.content.Intent;
7  import android.os.Bundle;
8  import android.view.View;
9  import android.widget.Button;
10
11 public class MainActivity extends Activity {
12   public int ID_NOTIFICATION = 0;
13
14   @Override
15   public void onCreate(Bundle savedInstanceState) {
16     super.onCreate(savedInstanceState);
17     setContentView(R.layout.activity_main);
18
19     Button b = (Button) findViewById(R.id.launch);
20     b.setOnClickListener(new View.OnClickListener() {
21       @Override
22       public void onClick(View view) {
23         // L'icône sera une petite loupe
24         int icon = R.drawable.ic_action_search;
25         // Le premier titre affiché
26         CharSequence tickerText = "Titre de la notification";
27         // Daté de maintenant
28         long when = System.currentTimeMillis();
29
30         // La notification est créée
31         Notification notification = new Notification(icon,
               tickerText, when);
32
33         Intent notificationIntent = new Intent(MainActivity.
               this, MainActivity.class);
34         notificationIntent.addFlags(Intent.
               FLAG_ACTIVITY_NEW_TASK);
35         PendingIntent contentIntent = PendingIntent.getActivity
               (MainActivity.this, 0, notificationIntent, 0);
36
37         notification.setLatestEventInfo(MainActivity.this, "
               Titre", "Texte", contentIntent);
38
```

```
39           startForeground(ID_NOTIFICATION, notification)
40         }
41       });
42     }
43 }
```

Vous pouvez ensuite enlever le service du premier plan avec `void stopForeground(boo lean removeNotification)`, ou vous pouvez préciser si vous voulez que la notification soit supprimée avec `removeNotification` (sinon le service sera arrêté, mais la notification persistera). Vous pouvez aussi arrêter le service avec les méthodes traditionnelles, auquel cas la notification sera aussi supprimée.

Pour aller plus loin : les alarmes

Il arrive parfois qu'on ait besoin de lancer des travaux à intervalles réguliers. C'est même indispensable pour certaines opérations : vérifier les e-mails de l'utilisateur, programmer une sonnerie tous les jours à la même heure, etc. Avec notre savoir, il existe déjà des solutions, mais rien qui permette de le faire de manière élégante !

La meilleure manière de faire est d'utiliser les alarmes. Une alarme est utilisée pour déclencher un `Intent` à intervalles réguliers.

Encore une fois, toutes les applications peuvent envoyer des alarmes, Android a donc besoin d'un système pour gérer toutes les alarmes, les envoyer au bon moment, etc. Ce système s'appelle `AlarmManager` et il est possible de le récupérer avec `Object context.getSystemService(Context.ALARM_SERVICE)`, un peu comme pour `NotificationManager`.

Il existe deux types d'alarme : les uniques et celles qui se répètent.

Les alarmes uniques

Pour qu'une alarme ne se déclenche qu'une fois, on utilise la méthode `void set(int type, long triggerAtMillis, PendingIntent operation)` sur l'`AlarmManager`.

On va commencer par le paramètre `triggerAtMillis`, qui définit à quel moment l'alarme se lancera. Le temps doit y être exprimé en millisecondes comme d'habitude, alors on utilisera la classe `Calendar`, que nous avons vue précédemment.

Ensuite, le paramètre `type` permet de définir le comportement de l'alarme vis à vis du paramètre `triggerAtMillis`. Est-ce que `triggerAtMillis` va déterminer le moment où l'alarme doit se déclencher (le 30 mars à 08 :52) ou dans combien de temps elle doit se déclencher (dans 25 minutes et 55 secondes) ? Pour définir une date exacte on utilisera la constante `RTC`, sinon pour un compte à rebours on utilisera `ELAPSED_REALTIME`. De plus, est-ce que vous souhaitez que l'alarme réveille l'appareil ou qu'elle se déclenche d'elle-même quand l'appareil sera réveillé d'une autre manière ? Si vous souhaitez que l'alarme réveille l'appareil rajoutez _WAKEUP aux constantes que nous venons de voir. On obtient ainsi `RTC_WAKEUP` et `ELAPSED_REALTIME_WAKEUP`.

Enfin, `operation` est le `PendingIntent` qui contient l'`Intent` qui sera enclenché dès que l'alarme se lancera.

Ainsi, pour une alarme qui se lance maintenant, on fera :

```
AlarmManager manager = (AlarmManager) context.getSystemService(
    Context.ALARM_SERVICE);
manager.set(RTC, System.currentTimeMillis(), pending);
```

Pour une alarme qui se lancera pour mon anniversaire (notez-le dans vos agendas !), tout en réveillant l'appareil :

```
Calendar calendar = Calendar.getInstance();
// N'oubliez pas que les mois commencent à 0, contrairement aux
      jours !
// Ne me faites pas de cadeaux en avril surtout !
calendar.set(1987, 4, 10, 17, 35);

manager.set(RTC_WAKEUP, calendar.getTimeInMillis(), pending);
```

Et pour une alarme qui se lance dans 20 minutes et 50 secondes :

```
calendar.set(Calendar.MINUTE, 20);
calendar.set(Calendar.SECOND, 50);

manager.set(ELAPSED_REALTIME, calendar.getTimeInMillis(),
    pending);
```

Les alarmes récurrentes

Il existe deux méthodes pour définir une alarme récurrente. La première est `void setRepeating(int type, long triggerAtMillis, long intervalMillis, PendingIntent operation)` qui prend les mêmes paramètres que précédemment à l'exception de `intervalMillis` qui est l'intervalle entre deux alarmes. Vous pouvez écrire n'importe quelle durée, cependant il existe quelques constantes qui peuvent vous aider :

- `INTERVAL_FIFTEEN_MINUTES` représente un quart d'heure.
- `INTERVAL_HALF_HOUR` représente une demi-heure.
- `INTERVAL_HOUR` représente une heure.
- `INTERVAL_HALF_DAY` représente 12 heures.
- `INTERVAL_DAY` représente 24 heures.

Vous pouvez bien entendu faire des opérations, par exemple `INTERVAL_HALF_DAY = INTERVAL_DAY / 2`. Pour obtenir une semaine, on peut faire `INTERVAL_DAY * 7`.

Si une alarme est retardée (parce que l'appareil est en veille et que le mode choisi ne réveille pas l'appareil par exemple), une requête manquée sera distribuée dès que possible. Par la suite, les alarmes seront à nouveau distribuées en fonction du plan originel.

Le problème de cette méthode est qu'elle est assez peu respectueuse de la batterie, alors si le délai de répétition est inférieur à une heure, on utilisera plutôt `void setInexactRepeating(int type, long triggerAtMillis, long intervalMillis, PendingIntent operation)`, auquel cas l'alarme n'est pas déclenchée au moment précis si c'est impossible.

Une alarme ne persiste pas après un redémarrage du périphérique. Si vous souhaitez que vos alarmes se réactivent à chaque démarrage du périphérique, il vous faudra écouter le `Broadcast Intent` appelé `ACTION_BOOT_COMPLETED`.

Annuler une alarme

Pour annuler une alarme, il faut utiliser la méthode `void cancel(PendingIntent operation)` où `operation` est le même `PendingIntent` qui accompagnait l'alarme. Si plusieurs alarmes utilisent le même `PendingIntent`, alors elles sont toutes annulées.

Il faut que tous les champs du `PendingIntent` soient identiques, à l'exception du champ `Données`. De plus, les deux `PendingIntent` doivent avoir le même identifiant.

En résumé

- Les services sont des composants très proches des activités puisqu'ils possèdent un contexte et un cycle de vie similaire mais ne possèdent pas d'interface graphique. Ils sont donc destinés à des travaux en arrière-plan.
- Il existe deux types de services :
  - Les services locaux où l'activité qui lance le service et le service en lui-même appartiennent à la même application.
  - Les services distants où le service est lancé par l'activité d'une activité d'une autre application du système.
- Le cycle de vie du service est légèrement différent selon qu'il soit lancé de manière locale ou distante.
- La création d'un service se déclare dans le `manifest` dans un premier temps et se crée dans le code Java en étendant la classe `Service` ou `IntentService` dans un second temps.
- Bien qu'il soit possible d'envoyer des notifications à partir d'une activité, les services sont particulièrement adaptés pour les lancer à la fin du traitement pour lequel ils sont destinés, par exemple.
- Les alarmes sont utiles lorsque vous avez besoin d'exécuter du code à un intervalle régulier.

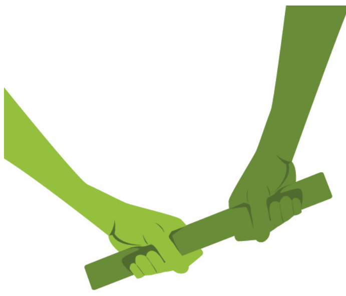
Chapitre 20

Le partage de contenus entre applications

Difficulté :

L'avantage des bases de données, c'est qu'elles facilitent le stockage de données complexes et structurées. Cependant, elles posent un problème : il n'est pas possible d'accéder à la base de données d'une application qui ne nous appartient pas. Néanmoins, il peut arriver qu'on ait vraiment besoin de partager du contenu entre plusieurs applications. Un exemple simple et courant est de pouvoir consulter les contacts de l'utilisateur qui sont enregistrés dans l'application « Carnet d'adresses ». Ces accès aux données d'une application différente de la nôtre se font à l'aide des **fournisseurs de contenu** ou *content providers* en anglais.

Les fournisseurs de contenu sont le quatrième et dernier composant des applications que nous verrons.



Techniquement, un fournisseur de contenu est découpé en deux éléments distincts :

- **Le fournisseur de contenu**, qui sera utilisé dans l'application qui distribue son contenu aux autres applications.
- **Un client**, qui permettra aux autres applications de demander au fournisseur les informations voulues.

Ensemble, les fournisseurs et les clients offrent une interface standardisée permettant l'échange sécurisé de données, ainsi que les communications inter-processus, de façon à faciliter les transactions entre applications. Ils permettent entre autres d'effectuer des copier/coller de données complexes depuis votre application vers d'autres applications.

Pour être tout à fait franc, il n'est pas rare qu'une application ne développe pas son propre fournisseur de contenu, car ils ne sont nécessaires que pour des besoins bien spécifiques, mais il se pourrait bien qu'un jour vous rencontriez ce type de difficultés.

Je reprends ici la même base de données qui représente les membres du Site du Zéro qui participent à l'écriture de ce cours. N'hésitez pas à aller relire complètement le chapitre sur les bases de données afin de vous familiariser avec cette architecture et vous remémorer les différentes techniques et termes techniques, ce chapitre-là étant intimement lié au présent chapitre.

Côté client : accéder à des fournisseurs

Les fournisseurs de contenu permettent l'encapsulation de données, et, pour accéder à ces données, il faudra utiliser les fameuses URI. Ici, nous ne saurons pas où ni comment les données sont stockées. Dans une base de données, dans un fichier, sur un serveur distant ? Cela ne nous regarde pas, du moment que les données nous sont mises à disposition.

Cependant, quel que soit le type de stockage, les données nous sont toujours présentées de la même manière. En effet, et un peu comme une base de données relationnelle, un fournisseur de contenu présente les données à une application extérieure dans une ou plusieurs tables. Chaque entrée dans la table est représentée dans une ligne et chaque colonne représente un attribut.

Une chose importante à savoir avant de faire appel à un fournisseur de contenu : vous devez savoir par avance la structure des tables (ses attributs et les valeurs qu'ils peuvent prendre), car vous en aurez besoin pour exploiter correctement ce fournisseur de contenu. Il n'y a pas moyen d'obtenir ce genre d'informations, il faut que le développeur du fournisseur vous communique cette information.

Un fournisseur n'a pas besoin d'avoir une clé primaire. S'il en a une, elle peut l'appeler `_ID`, même si ce n'est pas nécessaire. Si vous le faites, alors Android pourra faire quelques traitements automatiquement. Par exemple, si vous voulez lier des données depuis un fournisseur vers une `ListView`, il vaut mieux que la clé primaire s'appelle `_ID` de façon à ce que le `ListView` puisse deviner tout seul qu'il s'agit d'une clé primaire.

Examinons les éléments architecturaux des fournisseurs de contenu, ainsi que la relation qui existe entre les fournisseurs de contenu et les autres abstractions qui permettent l'accès aux données.

Accéder à un fournisseur

Il est possible d'accéder aux données d'une autre application avec un objet client `ContentResolver`. Cet objet a des méthodes qui appellent d'autres méthodes, qui ont le même nom, mais qui se trouvent dans un objet fournisseur, c'est-à-dire l'objet qui met à disposition le contenu pour les autres applications. Les objets fournisseurs sont de type `ContentProvider`. Aussi, si votre `ContentResolver` a une méthode qui s'appelle `myMethod`, alors le `ContentProvider` aura aussi une méthode qui s'appelle `myMethod`, et quand vous appelez `myMethod` sur votre `ContentResolver`, il fera en sorte d'appeler `myMethod` sur le `ContentProvider`.

Pourquoi je n'irais pas appeler ces méthodes moi-même ? Cela irait plus vite et ce serait plus simple !

Parce que ce n'est pas assez sécurisé ! Avec ce système, Android est certain que vous avez reçu les autorisations nécessaires à l'exécution de ces opérations.

Vous vous rappelez ce qu'on avait fait pour les bases de données ? On avait écrit des méthodes qui permettent de créer, lire, mettre à jour ou détruire des informations dans une base de données. Eh bien, ces méthodes, appelées méthodes CRUD (Create, Read, Update, Destroy), sont fournies par le `ContentResolver`. Ainsi, si mon `ContentResolver` demande poliment à un `ContentProvider` de lire des entrées dans la base de données de l'application dans laquelle se trouve ce `ContentProvider`, il appellera sur lui-même la méthode `lireCesDonnées` pour que soit appelée sur le `ContentProvider` la même méthode `lireCesDonnées`.

L'objet de type `ContentResolver` dans le processus de l'application cliente et l'objet de type `ContentProvider` de l'application qui fournit les données gèrent automatiquement les communications inter-processus, ce qui est bien parce que ce n'est pas une tâche aisée du tout. `ContentProvider` sert aussi comme une couche d'abstraction entre le référentiel de données et l'apparence extérieure des données en tant que tables.

Pour accéder à un fournisseur, votre application a besoin de certaines permissions. Vous ne pouvez bien entendu pas utiliser n'importe quel fournisseur sans l'autorisation de son application mère ! Nous verrons comment utiliser ou créer une permission par la suite.

Pour récupérer le gestionnaire des fournisseurs de contenu, on utilise la méthode de `Context` appelée `ContentResolver getContentResolver ()`. Vous aurez ensuite besoin d'une URI pour déterminer à quel fournisseur de contenu vous souhaitez accéder.

L'URI des fournisseurs de contenu

Le **schéma** d'une URI qui représente un fournisseur de contenu est `content`. Ainsi, ce type d'URI commence par `content://`.

Après le schéma, on trouve l'**information**. Comme dans le cas des URL sur internet, cette information sera un chemin. Ce chemin est dit hiérarchique : plus on rajoute d'informations, plus on devient précis sur le contenu voulu. La première partie du chemin s'appelle l'**autorité**. Elle est utilisée en tant qu'identifiant unique afin de pouvoir différencier les fournisseurs dans le registre des fournisseurs que tient Android. Un peu comme un nom de domaine sur internet. Si vous voulez aller sur le Site du Zéro , vous utiliserez le nom de domaine `www.siteduzero.com`. Ici, le schéma est `http` (dans le cas d'une URL, le schéma est le protocole de communication utilisé pour recevoir et envoyer des informations) et l'autorité est `www.siteduzero.com`, car elle permet de retrouver le site de manière unique. Il n'y a aucun autre site auquel vous pourrez accéder en utilisant l'adresse `www.siteduzero.com`.

Si on veut rentrer dans une partie spécifique du Site du Zéro, on va ajouter des composantes au chemin et chaque composante permet de préciser un peu plus l'emplacement ciblé : `http:/www.siteduzero.com/forum/android/demande_d_aide.html` (cette URL est bien entendu totalement fictive).

Comme vous pouvez le voir, les composantes sont séparées par des « / ». Ces composantes sont appelées des **segments**. On retrouve ainsi le segment `forum` qui nous permet de savoir qu'on se dirige vers les forums, puis `android` qui permet de savoir qu'on va aller sur un forum dédié à Android, et enfin `demande_d_aide.html` qui permet de se diriger vers le forum Android où on peut demander de l'aide.

Les URI pour les fournisseurs de contenu sont similaires. L'autorité seule est totalement nécessaire et chaque segment permet d'affiner un peu la recherche. Par exemple, il existe une API pour accéder aux données associées aux contacts enregistrés dans le téléphone : `ContactsContract`. Elle possède plusieurs tables, dont `ContactsContract.Data` qui contient des données sur les contacts (numéros de téléphone, adresses e-mail, comptes Facebook, etc.), `ContactsContract.RawContacts` qui contient les contacts en eux-mêmes, et enfin `ContactsContract.Contacts` qui fait le lien entre ces deux tables, pour lier un contact à ses données personnelles.

Pour accéder à `ContactsContract`, on peut utiliser l'URI `content://com.android.contacts/`. Si je cherche uniquement à accéder à la table `Contact`, je peux utiliser l'URI `content://com.android.contacts/contact`. Néanmoins, je peux affiner encore

plus la recherche en ajoutant un autre segment qui indiquera l'identifiant du contact recherché : `content://com.android.contacts/contact/18`.

Ainsi, si j'effectue une recherche avec `content://com.android.contacts/contact` sur mon téléphone, j'aurai 208 résultats, alors que si j'utilise `content://com.android.contacts/contact/18` je n'aurai qu'un résultat, celui d'identifiant 18.

De ce fait, le schéma sera `content://` et l'autorité sera composée du nom du package. Le premier segment indiquera la table dans laquelle il faut chercher et le deuxième la composante de la ligne à récupérer : `content://sdz.chapitreQuatre.Provider/Client/5`. Ici, je récupère la cinquième entrée de ma table `Client` dans mon application `Provider` qui se situe dans le package `sdz.chapitreQuatre`.

On ne pourra retrouver une ligne que si l'on a défini un identifiant en lui donnant comme nom de colonne `_ID`. Dans l'exemple précédent, on cherche dans la table `Client` celui qui a pour valeur 5 dans la colonne `_ID`.

Android possède nativement un certain nombre de fournisseurs de contenu qui sont décrits dans `android.provider`. Vous trouverez une liste de ces fournisseurs sur la documentation. On trouve parmi ces fournisseurs des accès aux données des contacts, des appels, des médias, etc. Chacune de ces classes possède une constante appelée `CONTENT_URI` qui est en fait l'URI pour accéder au fournisseur qu'elles incarnent. Ainsi, pour accéder au fournisseur de contenu de `ContactsContract.Contacts`, on pourra utiliser l'URI `ContactsContract.Contacts.CONTENT_URI`.

▷ Liste des fournisseurs
Code web : 456894

Vous remarquerez que l'autorité des fournisseurs de contenu d'Android ne respecte pas la tradition qui veut qu'on ait le nom du package ainsi que le nom du fournisseur. Google peut se le permettre mais pas vous, alors n'oubliez pas la bonne procédure à suivre.

On trouve par exemple :

Pour avoir accès à ces contenus natifs, il faut souvent demander une permission. Si vous voulez par exemple accéder aux contacts, n'oubliez pas de demander la permission adaptée : `android.permission.READ_CONTACTS`.

Il existe des API pour vous aider à construire les URI pour les fournisseurs de contenu. Vous connaissez déjà `Uri.Builder`, mais il existe aussi `ContentUris` rien que pour les fournisseurs de contenu.
Il contient par exemple la méthode statique `Uri ContentUris.withAppendedId(Uri contentUri, long id)` avec `contentUri` l'URI et `id` l'identifiant de la ligne à récupérer :

```
1 Uri client = ContentUris.withAppendedId(Uri.parse("content://
    sdz.chapitreQuatre.Provider/Client/"), 5);
```

Nom	Description	Interface
Contact	Permet l'accès aux données des contacts de l'utilisateur.	La base est `ContactsContract`, mais il existe une vingtaine de façons d'accéder à ces informations.
Magasin multimédia	Liste les différents médias disponibles sur le support, tels que les images, vidéos, fichiers audios, etc.	La base est `MediaStore`, mais il existe encore une fois un bon nombre de dérivés, par exemple `MediaStore.Audio.Artists` liste tous les artistes dans votre magasin.
Navigateur	Les données de navigation telles que l'historique ou les archives des recherches.	On a `Browser.SearchColumns` pour les historiques des recherches et `Browser.BookmarkColumns` pour les favoris de l'utilisateur.
Appel	Appels passés, reçus et manqués par l'utilisateur.	On peut trouver ces appels dans `CallLog.Calls`.
Dictionnaire	Les mots que connaît le dictionnaire utilisateur.	Ces mots sont gérés avec `UserDictionary.Words`.

Effectuer des opérations sur un fournisseur de contenu

Vous verrez ici d'énormes ressemblances avec la manipulation des bases de données, c'est normal, les deux API se fondent sur les mêmes principes. Il existe deux objets sur lesquels on peut effectuer les requêtes. Soit directement sur le `ContentResolver`, auquel cas vous devrez fournir à chaque fois l'URI du fournisseur de contenu visé. Soit, si vous effectuez les opérations sur le même fournisseur à chaque fois, vous pouvez utiliser plutôt un `ContentProviderClient`, afin de ne pas avoir à donner l'URI à chaque fois. On peut obtenir un `ContentProviderClient` en faisant `ContentProviderClient acquireContentProviderClient(String name)` sur un `ContentResolver`, `name` étant l'autorité du fournisseur.

Il n'est pas nécessaire de fermer un `ContentResolver`, cependant il faut appliquer `boolean release()` sur un `ContentProviderClient` pour aider le système à libérer de la mémoire. Exemple :

```
ContentProviderClient client = getContentResolver().
    acquireContentProviderClient("content://sdz.chapitreQuatre.
    Provider/Client/");

  // ...

client.release();
```

Les méthodes à utiliser entre les deux objets sont similaires, ils ont les mêmes paramètres, même si `ContentProviderClient` n'a pas besoin qu'on précise d'URI systématiquement. Je ne présenterai d'ailleurs que les méthodes de `ContentProvider`, retenez simplement qu'il suffit de ne pas passer le paramètre de type URI pour utiliser la méthode sur un `ContentProviderClient`.

Ajouter des données

Il existe deux méthodes pour ajouter des données : `Uri insert(Uri url, ContentValues values)`, qui permet d'insérer une valeur avec un `ContentValues` que nous avons appris à utiliser avec les bases de données. L'URI retournée représente la nouvelle ligne insérée.

```
ContentValues values = new ContentValues();

values.put(DatabaseHandler.METIER_INTITULE, "Autre");
values.put(DatabaseHandler.METIER_SALAIRE, 0);

contentResolver.insert(MetierProvider.CONTENT_URI, values);
```

Il est aussi possible d'utiliser `int bulkInsert(Uri url, ContentValues[] initialValues)` pour insérer plusieurs valeurs à la fois. Cette méthode retourne le nombre de lignes créées.

Récupérer des données

Il n'existe qu'une méthode cette fois : `Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)` avec les mêmes paramètres que d'habitude :

- `uri` indique le fournisseur de contenu.
- `project` est un tableau des colonnes de la table à récupérer.
- `selection` correspond à la valeur du `WHERE`.
- `selectionArgs` permet de remplacer les « ? » dans `selection` par des valeurs.
- `sortOrder` peut valoir « ASC » pour ranger les lignes retournées dans l'ordre croissant et « DESC » pour l'ordre décroissant.

Les résultats sont présentés dans un `Cursor`.

Par exemple, pour récupérer tous les utilisateurs dont le nom est « Apol » on peut faire :

```
Cursor c = contentResolver.query(Uri.parse("content://sdz.
    chapitreTrois.Membre"), null, "nom = ?", new String[] {"Apol
    "}, null);
```

Mettre à jour des données

On utilise `int update(Uri uri, ContentValues values, String where, String[] selectionArgs)` qui retourne le nombre de lignes mises à jour. Par exemple, pour changer le nom du métier « Autre » en « Les autres encore », on fera :

```
ContentValues values = new ContentValues();

values.put(DatabaseHandler.METIER_INTITULE, "Les autres encore");

int nombre = contentResolver.update(Uri.parse("content://sdz.chapitreTrois.Membre"), values, "metier = ?", new String[]{"Autre"});
```

Supprimer des données

Pour cela, il existe `int delete(Uri url, String where, String[] selectionArgs)` qui retourne le nombre de lignes mises à jour. Ainsi, pour supprimer les membres de nom « Apol » et de prénom « Lidore », on fera :

```
int nombre = contentResolver.delete(Uri.parse("content://sdz.chapitreTrois.Membre"), "nom = ?, prenom = ?", new String[]{"Apol", "Lidore"});
```

Créer un fournisseur

Maintenant que vous savez exploiter les fournisseurs de contenu, on va apprendre à en créer pour que vous puissiez mettre vos bases de données à disposition d'autres applications. Comme je l'ai déjà dit, il n'est pas rare qu'une application n'ait pas de fournisseur, parce qu'on les utilise uniquement pour certaines raisons particulières :

- Vous voulez permettre à d'autres applications d'accéder à des données complexes ou certains fichiers.
- Vous voulez permettre à d'autres applications de pouvoir copier des données complexes qui vous appartiennent.
- Enfin, vous voulez peut-être aussi permettre à d'autres applications de faire des recherches sur vos données complexes.

Une autre raison de ne construire un fournisseur que si nécessaire est qu'il ne s'agit pas d'une tâche triviale : la quantité de travail peut être énorme et la présence d'un fournisseur peut compromettre votre application si vous ne vous protégez pas.

Si vous voulez juste une base de données, n'oubliez pas que vous pouvez très bien le faire sans fournisseur de contenu. Je dis cela parce que certains ont tendance à confondre les deux concepts.

La préparation de la création d'un fournisseur de contenu se fait en plusieurs étapes.

L'URI

Vous l'avez bien compris, pour identifier les données à récupérer, l'utilisateur aura besoin d'une URI. Elle contiendra une autorité afin de permettre la récupération du fournisseur de contenu et un chemin pour permettre d'affiner la sélection et choisir une table, un fichier ou encore une ligne dans une table.

Le schéma

Il permet d'identifier quel type de contenu désigne l'URI. Vous le savez déjà, dans le cas des fournisseurs de contenu, ce schéma sera `content://`.

L'autorité

Elle sera utilisée comme identifiant pour Android. Quand on déclare un fournisseur dans le Manifest, elle sera inscrite dans un registre qui permettra de la distinguer parmi tous les fournisseurs quand on y fera appel. De manière standard, on utilise le nom du package dans l'autorité afin d'éviter les conflits avec les autres fournisseurs. Ainsi, si le nom de mon package est `sdz.chapitreQuatre.example`, alors pour le fournisseur j'utiliserai `sdz.chapitreQuatre.example.provider`.

Le chemin

Il n'y a rien d'obligatoire, mais en général le premier segment de chemin est utilisé pour identifier une table et le second est utilisé comme un identifiant. De ce fait, si on a deux tables `table1` et `table2`, on peut envisager d'y accéder avec `sdz.chapitreQuatre.example.provider/table1` et `sdz.chapitreQuatre.example.provider/table2`. Ensuite, pour avoir le cinquième élément de `table1`, on fait `sdz.chapitreQuatre.example.provider/table1/5` .

Vous pouvez avoir plusieurs segments ou faire en sorte qu'un segment ne corresponde pas à une table, c'est votre choix.

UriMatcher

Comme il existe beaucoup d'URI, il va falloir une technique pour toutes les gérer. C'est pourquoi je vais vous apprendre à utiliser `UriMatcher` qui analysera tout seul les URI et prendra les décisions pour vous.

On crée un `UriMatcher` toujours de la même manière :

```
UriMatcher membreMatcher = new UriMatcher(UriMatcher.NO_MATCH);
```

Cependant on n'utilisera qu'un seul `UriMatcher` par classe, alors on le déclarera en tant qu'attribut de type `static final` :

```
private static final UriMatcher membreMatcher = new UriMatcher(
    UriMatcher.NO_MATCH);
```

On va ensuite ajouter les différentes URI que pourra accepter le fournisseur, et on associera à chacune de ces URI un identifiant. De cette manière, on donnera des URI à notre `UriMatcher` et il déterminera tout seul le type de données associé.

Pour ajouter une URI, on utilise `void addURI(String authority, String path, int code)`, avec l'autorité dans `authority`, `path` qui incarne le chemin (on peut mettre # pour symboliser un nombre et * pour remplacer une quelconque chaîne de caractères) et enfin `code` l'identifiant associé à l'URI. De plus, comme notre `UriMatcher` est statique, on utilise ces ajouts dans un bloc `static` dans la déclaration de notre classe :

```
class maClass {
    // Autorité de ce fournisseur
    public static final String AUTHORITY = "sdz.chapitreQuatre.
        provider.MembreProvider";

    private static final int DIR = 0;
    private static final int ITEM = 1;

    private static final UriMatcher membreMatcher = new
        UriMatcher(UriMatcher.NO_MATCH);

    static {
        // Correspondra à content://sdz.chapitreQuatre.provider
            .MembreProvider/metier
        membreMatcher.addURI(AUTHORITY, "metier", DIR);
        // Correspondra à content://sdz.chapitreQuatre.provider
            .MembreProvider/metier/un_nombre
        membreMatcher.addURI(AUTHORITY, "metier/#", ITEM);
    }

    // ...

}
```

Enfin, on vérifie si une URI correspond aux filtres installés avec `int match(Uri uri)`, la valeur retournée étant l'identifiant de l'URI analysée :

```
switch(membreMatcher.match(Uri.parse("content://sdz.
    chapitreQuatre.provider.MembreProvider/metier/5") {
  case -1:
  // Si l'URI passée ne correspond à aucun filtre
  break;

  case 0:
  // Si l'URI passée est content://sdz.chapitreQuatre.provider.
      MembreProvider/metier
  break;

  case 1:
  // C'est le cas ici ! Notre URI est de la forme content://sdz
      .chapitreQuatre.provider.MembreProvider/metier/#
```

```
12    break;
13 }
```

Le type MIME

Android a besoin de connaître le type MIME des données auxquelles donne accès votre fournisseur de contenu, afin d'y accéder sans avoir à préciser leur structure ou leur implémentation. On a de ce fait besoin d'une méthode qui indique ce type (`String getType(Uri uri)`) dont le retour est une chaîne de caractères qui contient ce type MIME.

Cette méthode devra être capable de retourner deux formes de la même valeur en fonction de ce que veut l'utilisateur : une seule valeur ou une collection de valeurs. En effet, vous vous souvenez, un type MIME qui n'est pas officiel doit prendre sous Android la forme `vnd.android.cursor.X` avec `X` qui vaut `item` pour une ligne unique et `dir` pour une collection de lignes. Il faut ensuite une chaîne qui définira le type en lui-même, qui doit respecter la forme `vnd.<nom unique>.<type>`.

Voici ce que j'ai choisi :

```
vnd.android.cursor.item/vnd.sdz.chapitreQuatre.example.provider.
   table1
vnd.android.cursor.dir/vnd.sdz.chapitreQuatre.example.provider.
   table1
```

C'est ici que l'`UriMatcher` prendra tout son intérêt :

```
1  public static final String AUTHORITY = "sdz.chapitreQuatre.
       provider.MembreProvider";
2  public static final String TABLE_NAME = "metier";
3
4  public static final String TYPE_DIR =
5      "vnd.android.cursor.dir/vnd." + AUTHORITY + "." +
           TABLE_NAME;
6  public static final String TYPE_ITEM =
7      "vnd.android.cursor.item/vnd." + AUTHORITY + "." +
           TABLE_NAME;
8
9  public String getType(Uri uri) {
10     // Regardez dans l'exemple précédent, pour toute une table
           on avait la valeur 0
11     if (membreMatcher.match(uri) == 0) {
12         return(TYPE_DIR);
13     }
14
15     // Et si l'URI correspondait à une ligne précise dans une
           table, elle valait 1
16     return(TYPE_ITEM);
17 }
```

Le stockage

Comment allez-vous stocker les données ? En général, on utilise une base de données, mais vous pouvez très bien opter pour un stockage sur support externe. Je vais me concentrer ici sur l'utilisation des bases de données.

On va avoir une classe qui représente la base de données et, à l'intérieur de cette classe, des *classes internes constantes* qui représenteront chaque table. Une classe constante est une classe déclarée avec les modificateurs `static final`. Cette classe contiendra des attributs constants (donc qui possèdent aussi les attributs `static final`) qui définissent les URI, le nom de la table, le nom de ses colonnes, les types MIME ainsi que toutes les autres données nécessaires à l'utilisation du fournisseur. L'objectif de cette classe, c'est d'être certains que les applications qui feront appel au fournisseur pourront le manipuler aisément, même si certains changements sont effectués au niveau de la valeur des URI, du nom des colonnes ou quoi que ce soit d'autre. De plus, les classes constantes aident les développeurs puisque les constantes ont des noms mnémoniques plus pratiques à utiliser que si on devait retenir toutes les valeurs.

Bien entendu, comme les développeurs n'auront pas accès au code en lui-même, c'est à vous de bien documenter le code pour qu'ils puissent utiliser vos fournisseurs de contenu.

```
1  import android.content.UriMatcher;
2  import android.net.Uri;
3  import android.provider.BaseColumns;
4
5  public class MembreDatabase {
6      // Autorité de ce fournisseur
7      public static final String AUTHORITY = "sdz.chapitreQuatre.
           provider.MembreProvider";
8      // Nom du fichier qui représente la base
9      public static final String NAME = "membre.db";
10     // Version de la base
11     public static final int VERSION = 1;
12
13     private static final int DIR = 0;
14     private static final int ITEM = 1;
15
16     private static final UriMatcher membreMatcher = new
           UriMatcher(UriMatcher.NO_MATCH);
17
18     public static final class Metier implements BaseColumns {
19         static {
20           membreMatcher.addURI(AUTHORITY, "metier", DIR);
21           membreMatcher.addURI(AUTHORITY, "metier/#", ITEM);
22         }
23
24         // Nom de la table
25         public static final String TABLE_NAME = "metier";
26
```

```
27          // URI
28          public static final Uri CONTENT_URI =
29              Uri.parse("content://" + AUTHORITY + "/" +
                    TABLE_NAME);
30
31          // Types MIME
32          public static final String TYPE_DIR =
33              "vnd.android.cursor.dir/vnd." + AUTHORITY + "." +
                    TABLE_NAME;
34          public static final String TYPE_ITEM =
35              "vnd.android.cursor.item/vnd." + AUTHORITY + "." +
                    TABLE_NAME;
36
37          // Attributs de la table
38          public static final String INTITULE = "intitule";
39          public static final String SALAIRE = "salaire";
40      }
41  }
```

Comme vous pouvez le voir, ma classe `Metier` dérive de `BaseColumns`. Il s'agit d'une petite classe qui définit deux attributs indispensables : `_ID` (qui représente l'identifiant d'une ligne) et `_COUNT` (qui représente le nombre de lignes dans une requête).

Le Manifest

Chaque fournisseur de contenu s'enregistre sur un appareil à l'aide du Manifest. On aura besoin de préciser une autorité ainsi qu'un identifiant et la combinaison des deux se doit d'être unique. Cette combinaison n'est que la base utilisée pour constituer les requêtes de contenu. Le nœud doit être de type `provider`, puis on verra ensuite deux attributs : `android:name` pour le nom du composant (comme pour tous les composants) et `android:authorities` pour l'autorité.

```
1  <provider android:name=".MembreProvider"
2      android:authorities="sdz.chapitreQuatre.provider.
           MembreProvider" />
```

La programmation

On fait dériver une classe de `ContentProvider` pour gérer les requêtes qui vont s'effectuer sur notre fournisseur de contenu. Chaque opération qu'effectuera une application sur votre fournisseur de contenu sera à gérer dans la méthode idoine. Je vais donc vous présenter le détail de chaque méthode.

`boolean onCreate()`

Cette méthode de *callback* est appelée automatiquement dès qu'un `ContentResolver` essaie d'y accéder pour la première fois.

Le plus important ici est d'éviter les opérations qui prennent du temps, puisqu'il s'agit du démarrage, sinon celui-ci durera trop longtemps. Je pense par exemple à éviter les initialisations qui pourraient prendre du temps (comme créer, ouvrir, mettre à jour ou analyser la base de données), de façon à permettre aux applications de se lancer plus vite, d'éviter les efforts inutiles si le fournisseur n'est pas nécessaire, d'empêcher les erreurs de base de données (comme par exemple un disque plein), ou d'arrêter le lancement de l'application. De ce fait, faites en sorte de ne jamais appeler `getReadableDatabase()` ou `getWritableDatabase()` dans cette méthode.

La meilleure chose à faire, est d'implémenter `onOpen(SQLiteDatabase)` comme nous avons appris à le faire, pour initialiser la base de données quand elle est ouverte pour la première fois (dès que le fournisseur reçoit une quelconque requête concernant la base).

Par exemple, vous pouvez créer un `SQLiteOpenHelper` dans `onCreate()`, mais ne créez les tables que la première fois que vous ouvrez vraiment la base. Rappelez-vous que la première fois que vous appelez `getWritableDatabase()` on fera automatiquement appel à `onCreate()` de `SQLiteOpenHelper`.

N'oubliez pas de retourner `true` si tout s'est bien déroulé.

```
1 public boolean onCreate() {
2   // Je crée mon Handler comme nous l'avons vu dans le chapitre
        sur les bases de données
3   mHandler = new DatabaseHandler(getContext(), VERSION);
4
5   // Et si tout s'est bien passé, je retourne true
6   return((mHandler == null) ? false : true);
7
8   // Et voilà, on n'a pas ouvert ni touché à la base !
9 }
```

`Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)`

Permet d'effectuer des recherches sur la base. Elle doit retourner un `Cursor` qui contient le résultat de la recherche ou doit lancer une exception en cas de problème. S'il n'y a pas de résultat qui correspond à la recherche, alors il faut renvoyer un `Cursor` vide, et non `null`, qui est plutôt réservé aux erreurs.

```
1 public Cursor query(Uri url, String[] projection, String
      selection, String[] selectionArgs, String sort) {
2   SQLiteQueryBuilder builder = new SQLiteQueryBuilder();
3
4   builder.setTables(DatabaseHandler.METIER_TABLE_NAME);
5
6   Cursor c = builder.query(mHandler.getReadableDatabase(),
        projection, selection, selectionArgs, null, null, sort);
7   c.setNotificationUri(getContext().getContentResolver(), url);
8   return(c);
9 }
```

`Uri insert(Uri uri, ContentValues values)`

On l'utilise pour insérer des données dans le fournisseur. Elle doit retourner l'URI de la nouvelle ligne. Comme vous le savez déjà, ce type d'URI doit être constitué de l'URI qui caractérise la table suivie de l'identifiant de la ligne.

Afin d'alerter les éventuels observateurs qui suivent le fournisseur, on indique que l'ensemble des données a changé avec la méthode `void notifyChange(Uri uri, ContentObserver observer)`, `uri` indiquant les données qui ont changé et `observer` valant `null`.

```
1 public Uri insert (Uri url, ContentValues initialValues) {
2   long id = mHandler.getWritableDatabase().insert(
        DatabaseHandler.METIER_TABLE_NAME,    DatabaseHandler.
        METIER_KEY,    initialValues);
3   if (id > -1) {
4     Uri uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
5     getContext().getContentResolver().notifyChange(uri, null);
6     return uri;
7   }
8   return null;
9 }
```

`int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)`

Met à jour des données dans le fournisseur. Il faut retourner le nombre de lignes modifiées. N'oubliez pas d'alerter les observateurs avec `notifyChange()` encore une fois.

```
1 public int update (Uri url, ContentValues values, String where,
       String[] whereArgs) {
2   int count = mHandler.getWritableDatabase().update(
        DatabaseHandler.METIER_TABLE_NAME, values, where,
        whereArgs);
3   getContext().getContentResolver().notifyChange(url, null);
4   return count;
5 }
```

`int delete(Uri uri, String selection, String[] selectionArgs)`

Supprime des éléments du fournisseur et doit retourner le nombre de lignes supprimées. Pour alerter les observateurs, utilisez encore une fois `void notifyChange(Uri uri, ContentObserver observer)`.

```
1 public int delete(Uri url, String where, String[] whereArgs) {
2   int count = mHandler.getWritableDatabase().delete(
        DatabaseHandler.METIER_TABLE_NAME, where, whereArgs);
3   getContext().getContentResolver().notifyChange(url, null);
4   return count;
```

```
}
```

String getType(Uri uri)

Retourne le type MIME des données concernées par `uri`. Vous connaissez déjà cette méthode par cœur !

```java
public String getType(Uri uri) {
  if (membreMatcher.match(uri) == 0) {
    return(TYPE_DIR);
  }

  return(TYPE_ITEM);
}
```

En résumé

- Les fournisseurs de contenu permettent de rendre accessibles les données d'une application sans connaitre son moyen de stockage.
- Pour accéder à un fournisseur de contenu, vous êtes obligés de passer par un objet client `ContentResolver`.
- L'URI pour un fournisseur de contenu est sous la forme suivante : un schéma et l'information :
  - Le schéma d'un fournisseur de contenu est `content` et s'écrit `content://`.
  - L'information est un chemin qui devient de plus en plus précis à force de rentrer dans des parties spécifiques. La partie de l'information qui permet de pointer de manière unique vers le bon fournisseur de contenu est l'autorité. Quant à l'affinement de la requête par des « / », cela s'appelle des segments.
- Il n'est pas rare qu'une application n'offre pas de fournisseur de contenus. Il y a plusieurs raisons pour lesquelles vous pourrez en développer un :
  - Vous voulez permettre à d'autres applications d'accéder à des données complexes ou certains fichiers.
  - Vous voulez permettre à d'autres applications de pouvoir copier des données complexes qui vous appartiennent.
  - Vous voulez peut-être aussi permettre à d'autres applications de faire des recherches sur vos données complexes.

Chapitre 21

Créer un AppWidget

Difficulté :

Une des forces d'Android est son côté personnalisable. Un des exemples les plus probants est qu'il est tout à fait possible de choisir les éléments qui se trouvent sur l'écran d'accueil. On y trouve principalement des icônes, mais les utilisateurs d'Android sont aussi friands de ce qu'on appelle les « AppWidgets », applications miniatures destinées à être utilisées dans d'autres applications. Ces AppWidgets permettent d'améliorer une application à peu de frais en lui ajoutant un compagnon permanent. De plus, mettre un AppWidget sur son écran d'accueil permet à l'utilisateur de se rappeler l'existence de votre produit et par conséquent d'y accéder plus régulièrement. Par ailleurs, les AppWidgets peuvent accorder un accès direct à certaines fonctionnalités de l'application sans avoir à l'ouvrir, ou même ouvrir l'application ou des portions de l'application.

Un AppWidget est divisé en plusieurs unités, toutes nécessaires pour fonctionner. On retrouve tout d'abord une interface graphique qui détermine quelles sont les vues qui le composent et leurs dispositions. Ensuite, un élément gère le cycle de vie de l'AppWidget et fait le lien entre l'AppWidget et le système. Enfin, un dernier élément est utilisé pour indiquer les différentes informations de configuration qui déterminent certains aspects du comportement de l'AppWidget. Nous allons voir tous ces éléments, comment les créer et les manipuler.

L'interface graphique

La première chose à faire est de penser à l'interface graphique qui représentera la mise en page de l'AppWidget. Avant de vous y mettre, n'oubliez pas de réfléchir un peu. Si votre AppWidget est l'extension d'une application, faites en sorte de respecter la même charte graphique de manière à assurer une véritable continuité dans l'utilisation des deux programmes. Le pire serait qu'un utilisateur ne reconnaisse pas votre application en voyant un AppWidget et n'arrive pas à associer les deux dans sa tête.

Vous allez comme d'habitude devoir créer un layout dans le répertoire `res/layout/`. Cependant, il ne peut pas contenir toutes les vues qui existent. Voici les layouts acceptés :

- `FrameLayout`
- `LinearLayout`
- `RelativeLayout`

... ainsi que les widgets acceptés :

- `AnalogClock`
- `Button`
- `Chronometer`
- `ImageButton`
- `ImageView`
- `ProgressBar`
- `TextView`

Ne confondez pas les widgets, ces vues qui ne peuvent pas contenir d'autres vues, et les AppWidgets. Pour être franc, vous trouverez le terme « widget » utilisé pour désigner des AppWidgets, ce qui est tout à fait correct, mais pour des raisons pédagogiques je vais utiliser AppWidget.

Pourquoi uniquement ces vues-là ?

Toutes les vues ne sont pas égales. Ces vues-là sont des `RemoteViews`, c'est-à-dire qu'on peut y avoir accès quand elles se trouvent dans un autre processus que celui dans lequel on travaille. Au lieu de désérialiser une hiérarchie de vues comme on le fait d'habitude,

on désérialisera un layout dans un objet de type `RemoteViews`. Il est ainsi possible de configurer les vues dans notre receiver pour les rendre accessibles à une autre activité, celle de l'écran d'accueil par exemple.

L'une des contreparties de cette technique est que vous ne pouvez pas implémenter facilement la gestion des évènements avec un `OnClickListener` par exemple. À la place, on va attribuer un `PendingIntent` à notre `RemoteViews` de façon à ce qu'il sache ce qu'il doit faire en cas de clic, mais nous le verrons plus en détail bientôt.

Enfin, sachez qu'on ne retient pas de référence à des `RemoteViews`, tout comme on essaie de ne jamais faire de référence à des `context`.

Voici un exemple standard :

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res
       /android"
3    android:gravity="center"
4    android:orientation="vertical"
5    android:layout_width="wrap_content"
6    android:layout_height="wrap_content"
7    android:background="@drawable/background" >
8    <TextView
9      android:layout_width="fill_parent"
10     android:layout_height="wrap_content"
11     android:id="@+id/title"
12     android:textColor="#ff00ff00"
13     android:text="@string/title"
14     android:background="@drawable/black"
15     android:gravity="center" />
16   <Button
17     android:id="@+id/bouton"
18     android:layout_width="146dip"
19     android:layout_height="72dip"
20     android:text="@string/bouton"
21     android:background="@drawable/black"
22     android:gravity="center" />
23 </LinearLayout>
```

Vous remarquerez que j'ai utilisé des valeurs bien précises pour le bouton. En effet, il faut savoir que l'écran d'accueil est divisé en cellules. Une cellule est l'unité de base de longueur dans cette application, par exemple une icône fait une cellule de hauteur et une cellule de largeur. La plupart des écrans possèdent quatre cellules en hauteur et quatre cellules en largeur, ce qui donne $4 \times 4 = 16$ cellules en tout.

Pour déterminer la mesure que vous désirez en cellules, il suffit de faire le calcul $(74 \times N) - 2$ avec `N` le nombre de cellules voulues. Ainsi, j'ai voulu que mon bouton fasse une cellule de hauteur, ce qui donne $(74 \times 1) - 2 = 72$ dp.

Vous vous rappelez encore les dp ? C'est une unité qui est proportionnelle à la résolution de l'écran, contrairement à d'autres unités comme le pixel par exemple. Imaginez, sur un écran qui a 300 pixels en longueur, une ligne qui fait 150 pixels prendra la moitié de l'écran, mais sur un écran qui fait 1500 pixels de longueur elle n'en fera qu'un dixième ! En revanche, avec les dp (ou dip, c'est pareil), Android calculera automatiquement la valeur en pixels pour adapter la taille de la ligne à la résolution de l'écran.

Définir les propriétés

Maintenant, il faut préciser différents paramètres de l'AppWidget dans un fichier XML. Ce fichier XML représente un objet de type `AppWidgetProviderInfo`.

Tout d'abord, la racine est de type `<appwidget-provider>` et doit définir l'espace de nommage `android`, comme ceci :

```
1 <appwidget-provider xmlns:android="http://schemas.android.com/
      apk/res/android" />
```

Vous pouvez définir la hauteur minimale de l'AppWidget avec `android:minHeight` et sa largeur minimale avec `android:minWidth`. Les valeurs à indiquer sont en dp comme pour le layout.

Ensuite, on utilise `android:updatePeriodMillis` pour définir la fréquence de mise à jour voulue, en millisecondes. Ainsi, `android:updatePeriodMillis="60000"` fait une minute, `android:updatePeriodMillis="3600000"` fait une heure, etc. Puis on utilise `android:initialLayout` pour indiquer la référence au fichier XML qui indique le layout de l'AppWidget. Enfin, vous pouvez associer une activité qui permettra de configurer l'AppWidget avec `android:configure`.

Voici ce qu'on peut trouver dans un fichier du genre `res/xml/appwidget_info.xml` :

```
1 <appwidget-provider xmlns:android="http://schemas.android.com/
      apk/res/android"
2   android:minHeight="220dp"
3   android:minWidth="146dp"
4   android:updatePeriodMillis="3600000"
5   android:initialLayout="@layout/widget"
6   android:configure="sdz.chapitreQuatre.WidgetExample.
        AppWidgetConfigure" />
```

Le code

Le receiver

`AppWidgetProvider` est le composant de base qui permettra l'interaction avec l'AppWidget. Il permet de gérer tous les évènements autour de la vie de l'AppWidget.

`AppWidgetProvider` est une classe qui dérive de `BroadcastReceiver`, elle va donc recevoir les divers broadcast intents qui sont émis *et* qui sont destinés à l'AppWidget. On retrouve quatre évènements pris en compte : l'activation, la mise à jour, la désactivation et la suppression. Comme d'habitude, chaque période de la vie d'un AppWidget est représentée par une méthode de *callback*.

La mise à jour

La méthode la plus importante est celle relative à la mise à jour, vous devrez l'implémenter chaque fois.

Il s'agit de `public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds)` avec comme paramètres :

- Le `context` dans lequel le receiver s'exécute.
- `appWidgetManager` représente le gestionnaire des AppWidgets, il permet d'avoir des informations sur tous les AppWidgets disponibles sur le périphérique et de les mettre à jour.
- Les identifiants des AppWidgets à mettre à jour sont contenus dans `appWidgetIds`.

Cette méthode sera appelée à chaque expiration du délai `updatePeriodMillis`.

Ainsi, dans cette méthode, on va récupérer l'arbre de `RemoteViews` qui constitue l'interface graphique et on mettra à jour les vues qui ont besoin d'être mises à jour. Pour récupérer un `RemoteViews`, on utilisera le constructeur `RemoteViews(String packageName, int layoutId)` qui a besoin du nom du package du `context` dans `packageName` (on le récupère facilement avec la méthode `String getPackageName()` de `Context`) et l'identifiant du layout dans `layoutId`.

Vous pouvez ensuite manipuler n'importe quelle vue qui se trouve dans cette hiérarchie à l'aide de diverses méthodes de manipulation. Par exemple, pour changer le texte d'un `TextView`, on fera `void setTextViewText(int viewId, CharSequence text)` avec `viewId` l'identifiant du `TextView` et le nouveau `text`. Il n'existe bien entendu pas de méthodes pour toutes les méthodes que peuvent exécuter les différentes vues, c'est pourquoi `RemoteViews` propose des méthodes plus génériques qui permettent d'appeler des méthodes sur les vues et de leur passer des paramètres. Par exemple, un équivalent à :

```
1 remote.setTextViewText(R.id.textView, "Machin")
```

... pourrait être :

```
1 remote.setString(R.id.textView, "setText", "Machin")
```

On a en fait fait appel à la méthode `setText` de `TextView` en lui passant un `String`.

Maintenant que les modifications ont été faites, il faut les appliquer. En effet, elles ne sont pas effectives toutes seules, il vous faudra utiliser la méthode `void updateAppWidget (int appWidgetId, RemoteViews views)` avec `appWidgetId` l'identifiant du widget qui contient les vues et `views` la racine de type `RemoteViews`.

```
1 @Override
```

```
2  public void onUpdate(Context context, AppWidgetManager
       appWidgetManager, int[] appWidgetIds) {
3    // Pour chaque instance de notre AppWidget
4    for (int i = 0 ; i < appWidgetIds.length ; i++) {
5      // On crée la hiérarchie sous la forme d'un RemotViews
6      RemoteViews views = new RemoteViews(context.getPackageName
           (), R.layout.my_widget_layout);
7
8      // On récupère l'identifiant du widget actuel
9      int id = appWidgetIds[i];
10     // On met à jour toutes les vues du widget
11     appWidgetManager.updateAppWidget(id, views);
12   }
13 }
```

Les autres méthodes

Tout d'abord, comme `AppWidgetProvider` dérive de `BroadcastReceiver`, on pourra retrouver les méthodes de `BroadcastReceiver`, dont `public void onReceive(Context context, Intent intent)` qui est activé dès qu'on reçoit un broadcast intent.

La méthode `public void onEnabled(Context context)` n'est appelée que la première fois qu'un AppWidget est créé. Si l'utilisateur place deux fois un AppWidget sur l'écran d'accueil, alors cette méthode ne sera appelée que la première fois. Le broadcast intent associé est `APP_WIDGET_ENABLED`.

Ensuite, la méthode `public void onDeleted(Context context, int[] appWidgetIds)` est appelée à chaque fois qu'un AppWidget est supprimé. Il répond au broadcast intent `APP_WIDGET_DELETED`.

Et pour finir, quand la toute dernière instance de votre AppWidget est supprimée, le broadcast intent APP_WIDGET_DISABLED est envoyé afin de déclencher la méthode `public void onDisabled(Context context)`.

L'activité de configuration

C'est très simple, il suffit de créer une classe qui dérive de `PreferenceActivity` comme vous savez déjà le faire.

Déclarer l'AppWidget dans le Manifest

Le composant de base qui représente votre application est le `AppWidgetProvider`, c'est donc lui qu'il faut déclarer dans le Manifest. Comme `AppWidgetProvider` dérive de `BroadcastReceiver`, il faut déclarer un nœud de type `<receiver>`. Cependant, contrairement à un `BroadcastReceiver` classique où l'on pouvait ignorer les attributs

`android:icon` et `android:label`, ici il vaut mieux les déclarer. En effet, ils sont utilisés pour donner des informations sur l'écran de sélection des widgets :

```
<receiver
  android:name=".AppWidgetProviderExample"
  android:label="@string/nom_de_l_application"
  android:icon="@drawable/icone">
 ...
</receiver>
```

Il faut bien entendu rajouter des filtres à intents dans ce receiver, sinon il ne se lancera jamais. Le seul broadcast intent qui nous intéressera toujours est celui-ci : `android.appwidget.action.APPWIDGET_UPDATE`. Il est envoyé à chaque fois qu'il faut mettre à jour l'AppWidget :

```
<intent-filter>
  <action android:name="android.appwidget.action.
      APPWIDGET_UPDATE"/>
</intent-filter>
```

Ensuite, pour définir l'`AppWidgetProviderInfo`, il faut utiliser un élément de type `<meta-data>` avec les attributs `android:name` qui vaut `android.appwidget.provider` et `android:resource` qui est une référence au fichier XML contenant l'`AppWidgetProviderInfo` :

```
<meta-data android:name="android.appwidget.provider"
  android:resource="@xml/appwidget_info" />
```

Ce qui donne au complet :

```
<receiver
  android:name=".AppWidgetProviderExample"
  android:label="@string/nom_de_l_application"
  android:icon="@drawable/icone">
  <intent-filter>
    <action android:name="android.appwidget.action.
        APPWIDGET_UPDATE"/>
  </intent-filter>
  <meta-data android:name="android.appwidget.provider"
    android:resource="@xml/appwidget_info" />
</receiver>
```

Application : un AppWidget pour accéder aux tutoriels du Site du Zéro

On va créer un AppWidget qui ne sera pas lié à une application. Il permettra de choisir quel tutoriel du Site du Zéro l'utilisateur souhaite visualiser.

Résultat attendu

Mon AppWidget ressemble à la figure 21.1. Évidemment, vous pouvez modifier le design pour obtenir quelque chose de plus... esthétique. Là, c'est juste pour l'exemple.

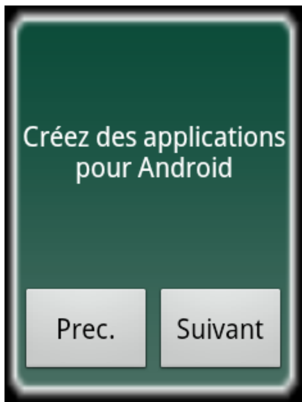


Figure 21.1 – Cet AppWidget permet d'accéder à des tutoriels du Site du Zéro

On peut cliquer sur le titre du tutoriel pour lancer le tutoriel dans un navigateur. Les deux boutons permettent de naviguer dans la liste des tutoriels disponibles.

Aspect technique

Pour permettre aux trois boutons (celui qui affiche le titre est aussi un bouton) de réagir aux clics, on va utiliser la méthode `void setOnClickPendingIntent(int viewId, PendingIntent pendingIntent)` de `RemoteViews` avec `viewId` l'identifiant du bouton et `pendingIntent` le `PendingIntent` qui contient l'`Intent` qui sera exécuté en cas de clic.

Détail important : pour ajouter plusieurs évènements de ce type, il faut différencier chaque `Intent` en leur ajoutant un champ Données différent. Par exemple, j'ai rajouté des données de cette manière à mes intents : `intent.setData(Uri.withAppendedPath(Uri.parse("WIDGET://widget/id/"), String.valueOf(Identifiant_de_cette_vue)))`. Ainsi, j'obtiens des données différentes pour chaque intent, même si ces données ne veulent rien dire.

Afin de faire en sorte qu'un intent lance la mise à jour de l'AppWidget, on lui mettra comme action `AppWidgetManager.ACTION_APPWIDGET_UPDATE` et comme extra les identifiants des widgets à mettre à jour ; l'identifiant de cet extra sera `AppWidgetManager.EXTRA_APPWIDGET_ID` :

```
1 | intent.setAction(AppWidgetManager.ACTION_APPWIDGET_UPDATE);
```

```
2 intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_IDS,
    appWidgetIds);
```

Comme `AppWidgetProvider` dérive de `BroadcastReceiver`, vous pouvez implémenter `void onReceive(Context context, Intent intent)` pour gérer chaque intent qui lance ce receiver.

Ma solution

Tout d'abord je déclare mon layout :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
3   android:layout_width="fill_parent"
4   android:layout_height="fill_parent"
5   android:orientation="vertical"
6   android:background="@drawable/background" >
7
8   <Button android:id="@+id/link"
9     android:layout_width="fill_parent"
10    android:layout_height="fill_parent"
11    android:layout_weight="30"
12    android:layout_marginLeft="10dp"
13    android:layout_marginRight="10dp"
14    android:layout_marginTop="10dp"
15    android:background="@android:color/transparent"
16    android:textColor="#FFFFFF" />
17
18  <LinearLayout android:layout_width="fill_parent"
19    android:layout_height="fill_parent"
20    android:layout_weight="70"
21    android:orientation="horizontal"
22    android:layout_marginBottom="10dp" >
23
24    <Button android:id="@+id/previous"
25      android:layout_width="fill_parent"
26      android:layout_height="wrap_content"
27      android:layout_marginLeft="10dp"
28      android:layout_weight="50"
29      android:text="Prec." />
30
31    <Button android:id="@+id/next"
32      android:layout_width="fill_parent"
33      android:layout_height="wrap_content"
34      android:layout_marginRight="10dp"
35      android:layout_weight="50"
36      android:text="Suivant" />
37  </LinearLayout>
38
```

```
39 </LinearLayout>
```

La seule chose réellement remarquable est que le fond du premier bouton est transparent grâce à l'attribut `android:background="@android:color/transparent"`.

Une fois mon interface graphique créée, je déclare mon `AppWidgetProviderInfo` :

```
1 <appwidget-provider xmlns:android="http://schemas.android.com/
      apk/res/android"
2   android:minWidth="144dip"
3   android:minHeight="144dip"
4   android:updatePeriodMillis="3600000"
5   android:initialLayout="@layout/widget" />
```

Je désire qu'il fasse au moins 2 cases en hauteur et 2 cases en largeur, et qu'il se rafraîchisse toutes les heures.

J'ai ensuite créé une classe très simple pour représenter les tutoriels :

```
1  package sdz.chapitrequatre.tutowidget;
2  import android.net.Uri;
3
4  public class Tuto {
5    private String intitule = null;
6    private Uri adresse = null;
7
8    public Tuto(String intitule, String adresse) {
9      this.intitule = intitule;
10     this.adresse = Uri.parse(adresse);
11   }
12
13   public String getIntitule() {
14     return intitule;
15   }
16
17   public void setIntitulé(String intitule) {
18     this.intitule = intitule;
19   }
20
21   public Uri getAdresse() {
22     return adresse;
23   }
24
25   public void setAdresse(Uri adresse) {
26     this.adresse = adresse;
27   }
28 }
```

Puis, le receiver associé à mon AppWidget :

```
1 package sdz.chapitrequatre.tutowidget;
2
3 import android.app.PendingIntent;
```

```
4   import android.appwidget.AppWidgetManager;
5   import android.appwidget.AppWidgetProvider;
6   import android.content.Context;
7   import android.content.Intent;
8   import android.net.Uri;
9   import android.widget.RemoteViews;
10
11  public class TutoWidget extends AppWidgetProvider {
12    // Les tutos que propose notre widget
13    private final static Tuto TUTO_ARRAY[] = {
14      new Tuto("Apprenez à créer votre site web avec HTML5 et
            CSS3", "http://www.siteduzero.com/tutoriel-3-13666-
            apprenez-a-creer-votre-site-web-avec-html5-et-css3.html"
            ),
15      new Tuto("Apprenez à programmer en C !", "http://www.
            siteduzero.com/tutoriel-3-14189-apprenez-a-programmer-en
            -c.html"),
16      new Tuto("Créez des applications pour Android", "http://www
            .siteduzero.com/tutoriel-3-554364-creez-des-applications
            -pour-android.html")
17    };
18
19    // Intitulé de l'extra qui contient la direction du défilé
20    private final static String EXTRA_DIRECTION = "extraDirection
          ";
21
22    // La valeur pour défiler vers la gauche
23    private final static String EXTRA_PREVIOUS = "previous";
24
25    // La valeur pour défiler vers la droite
26    private final static String EXTRA_NEXT = "next";
27
28    // Intitulé de l'extra qui contient l'indice actuel dans le
          tableau des tutos
29    private final static String EXTRA_INDICE = "extraIndice";
30
31    // Action qui indique qu'on essaie d'ouvrir un tuto sur
          internet
32    private final static String ACTION_OPEN_TUTO = "sdz.
          chapitreQuatre.tutowidget.action.OPEN_TUTO";
33
34    // Indice actuel dans le tableau des tutos
35    private int indice = 0;
36
37    @Override
38    public void onUpdate(Context context, AppWidgetManager
          appWidgetManager, int[] appWidgetIds) {
39      super.onUpdate(context, appWidgetManager, appWidgetIds);
40
```

```
41      // Petite astuce : permet de garder la longueur du tableau
           sans accéder plusieurs fois à l'objet, d'où optimisation
42      final int length = appWidgetIds.length;
43      for (int i = 0 ; i < length ; i++) {
44        // On récupère le RemoteViews qui correspond à l'
            AppWidget
45        RemoteViews views = new RemoteViews(context.
            getPackageName(), R.layout.widget);
46
47        // On met le bon texte dans le bouton
48        views.setTextViewText(R.id.link, TUTO_ARRAY[indice].
            getIntitule());
49
50        // La prochaine section est destinée au bouton qui permet
             de passer au tuto suivant
51        //
            ********************************************************
52        //*******************NEXT
            *********************************
53        //
            ********************************************************
54        Intent nextIntent = new Intent(context, TutoWidget.class)
            ;
55
56        // On veut que l'intent lance la mise à jour
57        nextIntent.setAction(AppWidgetManager.
            ACTION_APPWIDGET_UPDATE);
58
59        // On n'oublie pas les identifiants
60        nextIntent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_IDS,
             appWidgetIds);
61
62        // On rajoute la direction
63        nextIntent.putExtra(EXTRA_DIRECTION, EXTRA_NEXT);
64
65        // Ainsi que l'indice
66        nextIntent.putExtra(EXTRA_INDICE, indice);
67
68        // Les données inutiles mais qu'il faut rajouter
69        Uri data = Uri.withAppendedPath(Uri.parse("WIDGET://
            widget/id/"), String.valueOf(R.id.next));
70        nextIntent.setData(data);
71
72        // On insère l'intent dans un PendingIntent
73        PendingIntent nextPending = PendingIntent.getBroadcast(
            context, 0, nextIntent, PendingIntent.
            FLAG_UPDATE_CURRENT);
74
```

```
75          // Et on l'associe à l'activation du bouton
76          views.setOnClickPendingIntent(R.id.next, nextPending);
77
78          // La prochaine section est destinée au bouton qui permet
                de passer au tuto précédent
79          //
                *******************************************************
80          //*******************PREVIOUS
                *****************************
81          //
                *******************************************************
82
83          Intent previousIntent = new Intent(context, TutoWidget.
                class);
84
85          previousIntent.setAction(AppWidgetManager.
                ACTION_APPWIDGET_UPDATE);
86          previousIntent.putExtra(AppWidgetManager.
                EXTRA_APPWIDGET_IDS, appWidgetIds);
87          previousIntent.putExtra(EXTRA_DIRECTION, EXTRA_PREVIOUS);
88          previousIntent.putExtra(EXTRA_INDICE, indice);
89
90          data = Uri.withAppendedPath(Uri.parse("WIDGET://widget/id
                /"), String.valueOf(R.id.previous));
91          previousIntent.setData(data);
92
93          PendingIntent previousPending = PendingIntent.
                getBroadcast(context, 1, previousIntent, PendingIntent
                .FLAG_UPDATE_CURRENT);
94
95          views.setOnClickPendingIntent(R.id.previous,
                previousPending);
96
97
98          // La section suivante est destinée à l'ouverture d'un
                tuto dans le navigateur
99          //
                *******************************************************
100         //*******************LINK
                *********************************
101         //
                *******************************************************
102         // L'intent ouvre cette classe même...
103         Intent linkIntent = new Intent(context, TutoWidget.class)
                ;
104
```

```
105         // Action l'action ACTION_OPEN_TUTO
106         linkIntent.setAction(ACTION_OPEN_TUTO);
107         // Et l'adresse du site à visiter
108         linkIntent.setData(TUTO_ARRAY[indice].getAdresse());
109 
110         // On ajoute l'intent dans un PendingIntent
111         PendingIntent linkPending = PendingIntent.getBroadcast(
                context, 2, linkIntent, PendingIntent.
                FLAG_UPDATE_CURRENT);
112         views.setOnClickPendingIntent(R.id.link, linkPending);
113 
114         // Et il faut mettre à jour toutes les vues
115         appWidgetManager.updateAppWidget(appWidgetIds[i], views);
116       }
117     }
118 
119     @Override
120     public void onReceive(Context context, Intent intent) {
121       // Si l'action est celle d'ouverture du tutoriel
122       if(intent.getAction().equals(ACTION_OPEN_TUTO)) {
123         Intent link = new Intent(Intent.ACTION_VIEW);
124         link.setData(intent.getData());
125         link.addCategory(Intent.CATEGORY_DEFAULT);
126         // Comme on ne se trouve pas dans une activité, on
                demande à créer une nouvelle tâche
127         link.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
128         context.startActivity(link);
129       } else {
130         // Sinon, s'il s'agit d'une demande de mise à jour
131         // On récupère l'indice passé en extra, ou -1 s'il n'y a
                pas d'indice
132         int tmp = intent.getIntExtra(EXTRA_INDICE, -1);
133 
134         // S'il y avait bien un indice passé
135         if(tmp != -1) {
136           // On récupère la direction
137           String extra = intent.getStringExtra(EXTRA_DIRECTION);
138           // Et on calcule l'indice voulu par l'utilisateur
139           if (extra.equals(EXTRA_PREVIOUS)) {
140             indice = (tmp - 1) % TUTO_ARRAY.length;
141             if(indice < 0)
142               indice += TUTO_ARRAY.length;
143           } else if(extra.equals(EXTRA_NEXT))
144             indice = (tmp + 1) % TUTO_ARRAY.length;
145         }
146       }
147 
148       // On revient au traitement naturel du Receiver, qui va
              lancer onUpdate s'il y a demande de mise à jour
149       super.onReceive(context, intent);
```

```
    }

}
```

Enfin, on déclare le tout dans le Manifest :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sdz.chapitrequatre.tutowidget"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="7"
    android:targetSdkVersion="7" />

  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <receiver
      android:name=".TutoWidget"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
        <action android:name="sdz.chapitreQuatre.tutowidget.action.OPEN_TUTO" />
      </intent-filter>

      <meta-data
        android:name="android.appwidget.provider"
        android:resource="@xml/widget_provider_info" />
    </receiver>
  </application>

</manifest>
```

Vous pouvez copier les codes du projet grâce au code web suivant :

▷ Copier ce code
Code web : 923952

En résumé

- Un AppWidget est une extension de votre application. Afin que l'utilisateur ne soit pas désorienté, adoptez la même charte graphique que votre application.
- Les seules vues utilisables pour un widget sont les vues `RemoteViews`.

- La déclaration d'un AppWidget se fait dans un élément `appwidget-provider` à partir d'un fichier XML `AppWidgetProviderInfo`.
- La super classe de notre AppWidget sera un `AppWidgetProvider`. Il s'occupera de gérer tous les évènements sur le cycle de vie de notre AppWidget. Cette classe dérive de `BroadcastReceiver`, elle va donc recevoir les divers broadcast intents qui sont émis et qui sont destinés à l'AppWidget.
- Pour déclarer notre AppWidget dans le manifest, nous allons créer un élément `receiver` auquel nous ajoutons un élément `intent-filter`pour lancer notre AppWidget et un élément `meta-data` pour définir l'`AppWidgetProviderInfo`.

Cinquième partie

Exploiter les fonctionnalités d'Android

Chapitre 22

La connectivité réseau

Difficulté :

Maintenant que vous savez tout ce qu'il y a à savoir sur les différentes facettes des applications Android, voyons maintenant ce que nous offre notre terminal en matière de fonctionnalités. La première sur laquelle nous allons nous pencher est la connectivité réseau, en particulier l'accès à internet. On va ainsi voir comment surveiller la connexion au réseau ainsi que comment contrôler cet accès. Afin de se connecter à internet, le terminal peut utiliser deux interfaces. Soit le réseau mobile (3G, 4G, etc.), soit le WiFi.

Il y a de fortes chances pour que ce chapitre vous soit utile, puisque statistiquement la permission la plus demandée est celle qui permet de se connecter à internet.

Surveiller le réseau

Avant toute chose, nous devons nous assurer que l'appareil a bien accès à internet. Pour cela, nous avons besoin de demander la permission au système dans le Manifest :

```
1 <uses-permission android:name="android.permission.
     ACCESS_NETWORK_STATE" />
```

Il existe deux classes qui permettent d'obtenir des informations sur l'état du réseau. Si vous voulez des informations sur sa disponibilité en général, utilisez `ConnectivityManager`. En revanche, si vous souhaitez des informations sur l'état de l'une des interfaces réseau (en général le réseau mobile ou le WiFi), alors utilisez plutôt `NetworkInfo`.

On peut récupérer le gestionnaire de connexions dans un `Context` avec `ConnectivityManager Context.getSystemService(Context.CONNECTIVITY_SERVICE)`.

Ensuite, pour savoir quelle est l'interface active, on peut utiliser la méthode `NetworkInfo getActiveNetworkInfo()`. Si aucun réseau n'est disponible, cette méthode renverra `null`.

Vous pouvez également aller vérifier l'état de chaque interface avec `NetworkInfo getNetworkInfo(ConnectivityManager.TYPE_WIFI)` ou `NetworkInfo getNetworkInfo(ConnectivityManager.TYPE_MOBILE)` pour le réseau mobile.

Enfin, il est possible de demander à un `NetworkInfo` s'il est connecté à l'aide de la méthode `boolean isAvailable()` :

```
1 ConnectivityManager connectivityManager = (ConnectivityManager)
      getSystemService(CONNECTIVITY_SERVICE);
2 NetworkInfo networkInfo = connectivityManager.
     getActiveNetworkInfo();
3
4 if(networkInfo != null && networkInfo.isAvailable() &&
     networkInfo.isConnected()) {
5   boolean wifi = networkInfo.getType() == ConnectivityManager.
       TYPE_WIFI;
6   Log.d("NetworkState", "L'interface de connexion active est du
        Wifi : " + wifi);
7 }
```

De manière générale, on préférera utiliser internet si l'utilisateur est en WiFi parce que le réseau mobile est plus lent et est souvent payant. Il est conseillé de mettre en garde l'utilisateur avant de télécharger quelque chose en réseau mobile. Vous pouvez aussi envisager de bloquer les téléchargements quand seul le réseau mobile est disponible, comme c'est souvent fait.

Mais il est possible que l'état de la connexion change et qu'il faille réagir à ce changement. Dès qu'un changement surgit, le broadcast intent `ConnectivityManager.CONNECTIVITY_ACTION` est envoyé (sa valeur est étrangement `android.net.conn.CONNECTIVITY_CHANGE`). Vous pourrez donc l'écouter avec un receiver déclaré de cette

manière :

```
<receiver android:name=".ConnectionChangesReceiver" >
  <intent-filter >
    <action android:name="android.net.conn.CONNECTIVITY_CHANGE"
        />
  </intent-filter>
</receiver>
```

Vous trouverez ensuite dans les extras de l'intent plus d'informations. Par exemple `ConnectivityManager.EXTRA_NO_CONNECTIVITY` renvoie un booléen qui vaut `true` s'il n'y a pas de connexion à internet en cours. Vous pouvez aussi obtenir directement un `NetworkInfo` avec l'extra `ConnectivityManager.EXTRA_OTHER_NETWORK_INFO` afin d'avoir plus d'informations sur le changement.

Afficher des pages Web

Il pourrait vous prendre l'envie de montrer à votre utilisateur une page Web. Ou alors il se peut que vous vouliez faire une interface graphique à l'aide de HTML. Nous avons déjà vu une méthode pour mettre du HTML dans des `TextView`, mais ces méthodes ne sont pas valides pour des utilisations plus poussées du HTML, comme par exemple pour afficher des images ; alors pour afficher une page complète, n'imaginez même pas.

Ainsi, pour avoir une utilisation plus poussée de HTML, on va utiliser une nouvelle vue qui s'appelle `WebView`. En plus d'être une vue très puissante, `WebView` est commandé par WebKit, un moteur de rendu de page Web qui fournit des méthodes pratiques pour récupérer des pages sur internet, effectuer des recherches dans la page, etc.

Charger directement du HTML

Pour insérer des données HTML sous forme textuelle, vous pouvez utiliser `void loadData(String data, String mimeType, String encoding)` avec `data` les données HTML, `mimeType` le type MIME (en général `text/html`) et l'encodage défini dans `encoding`. Si vous ne savez pas quoi mettre pour `encoding`, mettez « UTF-8 », cela devrait aller la plupart du temps.

```
import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebView;

public class WebViewActivity extends Activity {
  private WebView mWebView = null;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_web_view);

```

```
13      mWebView = (WebView) findViewById(R.id.webview);
14      mWebView.loadData("<html><head><meta charset=\"utf-8\" /></head>" + "<body>Salut les Zéros !</body></html>", "text/html", "UTF-8");
15    }
16
17  }
```

On obtient alors la figure 22.1.

WebViewActivity

Salut les Zéros !

FIGURE 22.1 – Du HTML s'affiche

N'oubliez pas de préciser l'encodage dans l'en-tête, sinon vos accents ne passeront pas.

Charger une page sur internet

La première chose à faire est de demander la permission pour aller sur internet dans le Manifest :

```
1 <uses-permission android:name="android.permission.INTERNET" />
```

Puis vous pouvez charger le contenu avec `void loadUrl(String url)`. Ainsi, avec ce code :

```
1 public void onCreate(Bundle savedInstanceState) {
2   super.onCreate(savedInstanceState);
3   setContentView(R.layout.activity_web_view);
4
5   mWebView = (WebView) findViewById(R.id.webview);
6   mWebView.loadUrl("http://www.siteduzero.com");
7 }
```

... on obtient la figure 22.2 :

Effectuer des requêtes HTTP

Rappels sur le protocole HTTP

HTTP (Hypertext Transfer Protocol) est un protocole de communication, c'est-à-dire un ensemble de règles à suivre quand deux machines veulent communiquer. On l'utilise

FIGURE 22.2 – Le Site du Zéro est affiché à l'écran

surtout dans le cadre du *World Wide Web*, une des applications d'internet, celle qui vous permet de voir des sites en ligne. Vous remarquerez d'ailleurs que l'URI que vous utilisez pour accéder à un site sur internet a pour schéma `http:`, comme sur cette adresse : `http://www.siteduzero.com`.

Il fonctionne de cette manière : un client envoie une requête HTTP à un serveur qui va réagir et répondre en HTTP en fonction de cette entrée, comme le montre la figure 22.3.

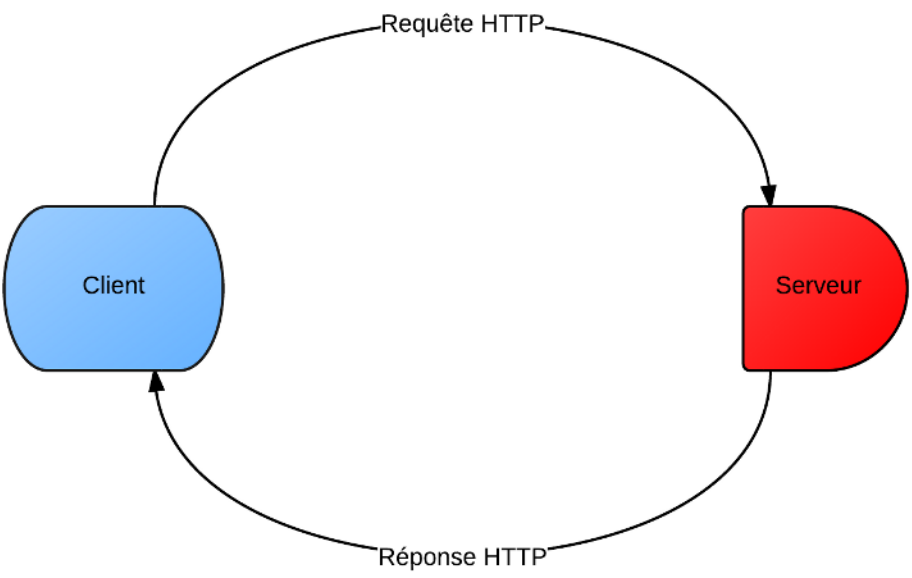


FIGURE 22.3 – La requête et la réponse utilisent le même protocole, mais leur contenu est déterminé par le client ou le serveur

Il existe plusieurs méthodes en HTTP. Ce sont des commandes, des ordres qui accompagnent les requêtes. Par exemple, si on veut récupérer une ressource on utilise la méthode `GET`. Quand vous tapez une adresse dans la barre de navigation de votre navigateur internet, il fera un `GET` pour récupérer le contenu de la page.

À l'opposé de `GET`, on trouve `POST` qui est utilisé pour envoyer des informations. Quand vous vous inscrivez sur un site, ce qui se fait souvent à l'aide d'un formulaire, l'envoi de ce dernier correspond en fait à un `POST` vers le serveur qui contient les diverses informations que vous avez envoyées. Mais comment ça fonctionne, concrètement ? C'est simple, dans votre requête `POST`, votre navigateur va ajouter comme données un ensemble de couples identifiant-clé de cette forme-ci : `identifiant1=clé1&identifiant2=clé2&identifiant3=clé3`. Ainsi, votre serveur sera capable de retrouver les identifiants avec les clés qui y sont associées.

Le HTTP sous Android

Il existe deux méthodes pour manipuler le protocole HTTP sous Android. La première est fournie par Apache et est celle que vous êtes censés utiliser avant l'API 9 (Gingerbread). En pratique, nous allons voir l'autre méthode même si elle n'est pas recommandée pour l'API 7, parce qu'elle est à privilégier pour la plupart de vos applications.

Pour commencer, la première chose à faire est de savoir sur quelle URL on va opérer avec un objet de type URL. La manière la plus simple d'en créer un est de le faire à l'aide d'une chaîne de caractères :

```
1 URL sdz = new URL("http://www.siteduzero.com")
```

Toutes vos requêtes HTTP devront se faire dans un thread différent du thread UI, puisqu'il s'agit de processus lents qui risqueraient d'affecter les performances de votre application.

On peut ensuite ouvrir une connexion vers cette URL avec la méthode URLConnection openConnection(). Elle renvoie un URLConnection, qui est une classe permettant de lire et d'écrire depuis une URL. Ici, nous allons voir en particulier la connexion à une URL avec le protocole HTTP, on va donc utiliser une classe qui dérive de URLConnection : HttpURLConnection.

```
1 URLConnection urlConnection = url.openConnection();
2 HttpURLConnection httpUrlConnection = (HttpURLConnection)
      connection;
```

Il est ensuite possible de récupérer le flux afin de lire des données en utilisant la méthode InputStream getInputStream(). Avant cela, vous souhaiterez peut être vérifier le code de réponse fourni par le serveur HTTP, car votre application ne réagira pas de la même manière si vous recevez une erreur ou si tout s'est déroulé correctement. Vous pouvez le faire avec la méthode int getResponseCode() :

```
1 if (httpConnection.getResponseCode() == HttpURLConnection.
      HTTP_OK) {
2   InputStream stream = httpConnection.getInputStream();
3   // Par exemple...
4   stream.read();
5 }
```

Enfin, si vous voulez effectuer des requêtes sortantes, c'est-à-dire vers un serveur, il faudra utiliser la méthode setDoOutput(true) sur votre HttpURLConnection afin d'autoriser les flux sortants. Ensuite, si vous connaissez la taille des paquets que vous allez transmettre, utilisez void setFixedLengthStreamingMode(int contentLength) pour optimiser la procédure, avec contentLength la taille des paquets. En revanche, si vous ne connaissez pas cette taille, alors utilisez setChunkedStreamingMode(0) qui va séparer votre requête en paquets d'une taille définie par le système :

```
HttpURLConnection connection = (HttpURLConnection) url.
    openConnection();
urlConnection.setDoOutput(true);
urlConnection.setChunkedStreamingMode(0);

OutputStream stream = new BufferedOutputStream(urlConnection.
    getOutputStream());
writeStream(stream );
```

Dans les versions les plus récentes d'Android, effectuer des requêtes HTTP dans le thread UI soulèvera une exception, vous serez donc obligés de créer un thread pour effectuer vos requêtes. De toute manière, même si vous êtes dans une ancienne version qui ne soulève pas d'exception, il vous faut quand même créer un nouveau thread, parce que c'est la bonne manière.

Pour finir, comme pour n'importe quel autre flux, n'oubliez pas de vous déconnecter avec `void disconnect()`.

Avant de vous laissez, je vais vous montrer une utilisation correcte de cette classe. Vous vous rappelez que je vous avais dit que normalement il ne fallait pas utiliser cette API pour faire ces requêtes ; c'est en fait parce qu'elle est boguée. L'un des bugs qui vous agacera le plus est que, vous aurez beau demander de fermer un flux, Android ne le fera pas. Pour passer outre, nous allons désactiver une fonctionnalité du système qui permet de contourner le problème :

```
System.setProperty("http.keepAlive", "false");
```

Voici par exemple une petite application qui envoie des données à une adresse et récupère ensuite la réponse :

```
System.setProperty("http.keepAlive", "false");
OutputStreamWriter writer = null;
BufferedReader reader = null;
URLConnection connexion = null;
try {
  // Encodage des paramètres de la requête
  String donnees = URLEncoder.encode("identifiant1", "UTF-8")+
      "="+URLEncoder.encode("valeur1", "UTF-8");
  donnees += "&"+URLEncoder.encode("identifiant2", "UTF-8")+ "=
      " + URLEncoder.encode("valeur2", "UTF-8");

  // On a envoyé les données à une adresse distante
  URL url = new URL(adresse);
  connexion = url.openConnection();
  connexion.setDoOutput(true);
  connexion.setChunkedStreamingMode(0);

  // On envoie la requête ici
  writer = new OutputStreamWriter(connexion.getOutputStream());

```

```
19      // On insère les données dans notre flux
20      writer.write(donnees);
21
22      // Et on s'assure que le flux est vidé
23      writer.flush();
24
25      // On lit la réponse ici
26      reader = new BufferedReader(new InputStreamReader(connexion.
            getInputStream()));
27      String ligne;
28
29      // Tant que « ligne » n'est pas null, c'est que le flux n'a
            pas terminé d'envoyer des informations
30      while ((ligne = reader.readLine()) != null) {
31        System.out.println(ligne);
32      }
33    } catch (Exception e) {
34      e.printStackTrace();
35    } finally {
36      try{writer.close();}catch(Exception e){}
37      try{reader.close();}catch(Exception e){}
38      try{connexion.disconnect();}catch(Exception e){}
39    }
```

En résumé

- Dans votre vie de programmeur Android, il est très probable que vous liez au moins une application à internet tellement c'est quelque chose de courant.
- On peut obtenir des informations sur l'état de la connectivité de l'appareil grâce à `ConnectivityManager`. C'est indispensable parce qu'il y a des chances que l'utilisateur passe du Wifi à un réseau mobile lorsqu'il exploite votre application. Dès que ça arrive, il faut couper tout téléchargement pour que votre pauvre utilisateur ne se retrouve pas avec une facture longue comme son bras !
- On peut très facilement afficher du code HTML avec une `WebView`.
- Sur le web, pour envoyer et recevoir des applications, on utilise le protocole HTTP. Il possède en autre la méthode `GET` pour récupérer du contenu sur internet et la méthode `POST` pour en envoyer.
- Quand on récupère du contenu sur internet, on passe toujours par un thread, car récupérer cela prend du temps et impacterait trop l'utilisateur si on l'employait dans le thread UI.
- On récupère et on poste du contenu facilement à l'aide de flux comme nous l'avons toujours fait pour écrire dans un fichier.

Chapitre 23

Apprenez à dessiner

Difficulté :

Je vous propose d'approfondir nos connaissances du dessin sous Android. Même si dessiner quand on programme peut sembler trivial à beaucoup d'entre vous, il faut que vous compreniez que c'est un élément qu'on retrouve dans énormément de domaines de l'informatique. Par exemple, quand on veut faire sa propre vue, on a besoin de la dessiner. De même, dessiner est une étape essentielle pour faire un jeu.

Enfin, ne vous emballez pas parce que je parle de jeu. En effet, un jeu est bien plus que des graphismes, il faut créer différents moteurs pour gérer le *gameplay*, il faut travailler sur l'aspect sonore, etc. De plus, la méthode présentée ici est assez peu adaptée au jeu. Mais elle va quand même nous permettre de faire des choses plutôt sympa.

La toile

Non, non, je ne parle pas d'internet ou d'un écran de cinéma, mais bien d'une vraie toile. Pas en lin ni en coton, mais une toile de pixels. C'est sur cette toile que s'effectuent nos dessins. Et vous l'avez déjà rencontrée, cette toile ! Mais oui, quand nous dessinions nos propres vues, nous avons vu un objet de type `Canvas` sur lequel dessiner !

Pour être tout à fait franc, ce n'était pas exactement la réalité. En effet, on ne dessine pas sur un `Canvas`, ce n'est pas un objet graphique, mais une interface qui va dessiner sur un objet graphique. Le dessin est en fait effectué sur un `Bitmap`. Ainsi, il ne suffit pas de créer un `Canvas` pour pouvoir dessiner, il faut lui attribuer un `Bitmap`.

La plupart du temps, vous n'aurez pas besoin de donner de `Bitmap` à un `Canvas` puisque les `Canvas` qu'on vous donnera auront déjà un `Bitmap` associé. Les seuls moments où vous devrez le faire manuellement sont les moments où vous créerez vous-mêmes un `Canvas`.

Ainsi, un `Canvas` est un objet qui réalise un dessin et un `Bitmap` est une surface sur laquelle dessiner. Pour raisonner par analogie, on peut se dire qu'un `Canvas` est un peintre et un `Bitmap` une toile. Cependant, que serait un peintre sans son fidèle pinceau ? Un pinceau est représenté par un objet `Paint` et permet de définir la couleur du trait, sa taille, etc. Alors quel est votre rôle à vous ? Eh bien, imaginez-vous en tant que client qui demande au peintre (`Canvas`) de dessiner ce que vous voulez, avec la couleur que vous voulez et sur la surface que vous voulez. C'est donc au `Canvas` que vous donnerez des ordres pour dessiner.

La toile

Il n'y a pas grand-chose à savoir sur les `Bitmap`. Tout d'abord, il n'y a pas de constructeur dans la classe `Bitmap`. Le moyen le plus simple de créer un `Bitmap` est de passer par la méthode statique `Bitmap createBitmap(int width, int height, Bitmap.Config config)` avec `width` la largeur de la surface, `height` sa hauteur et `config` un objet permettant de déterminer comment les pixels seront stockés dans le `Bitmap`.

En fait, le paramètre `config` permet de décrire quel espace de couleur sera utilisé. En effet, les couleurs peuvent être représentées d'au moins trois manières :

- Pour que chaque pixel ne puisse être qu'une couleur, utilisez `Bitmap.Config.RGB_565`.
- Pour que chaque pixel puisse être soit une couleur, soit transparent (c'est-à-dire qu'il n'affiche pas de couleur), utilisez `Bitmap.Config.ARGB_8888`.
- Enfin, si vous voulez que seul le canal qui représente des pixels transparents soit disponible, donc pour n'avoir que des pixels transparents, utilisez `Bitmap.Config.ALPHA_8`.

Par exemple :

```
1 | Bitmap b = Bitmap.createBitmap(128, 128, Config.ARGB_8888);
```

Il existe aussi une classe dédiée à la construction de `Bitmap` : `BitmapFactory`. Pour créer un `Bitmap` depuis un fichier d'image, on fait `BitmapFactory.decodeFile("Chemin`

vers le fichier"). Pour le faire depuis un fichier de ressource, on utilise la méthode statique decodeResource(Resources ressources, int id) avec le fichier qui permet l'accès aux ressources et l'identifiant de la ressource dans id. Par exemple :

```
1 Bitmap b = BitmapFactory.decodeResource(getResources(), R.
      drawable.ic_action_search);
```

N'oubliez pas qu'on peut récupérer un fichier de type Resources sur n'importe quel Context avec la méthode getResources().

Enfin, et surtout, vous pouvez récupérer un Bitmap avec BitmapFactory.decodeStream (InputStream).

À l'opposé, au moment où l'on n'a plus besoin de Bitmap, on utilise dessus la méthode void recycle(). En effet, ça semble une habitude mais Bitmap n'est aussi qu'une interface et recycle() permet de libérer toutes les références à certains objets de manière à ce qu'ils puissent être ramassés par le *garbage collector*.

Après cette opération, le Bitmap n'est plus valide, vous ne pourrez plus l'utiliser ou faire d'opération dessus.

Le pinceau

Pour être tout à fait exact, Paint représente à la fois le pinceau et la palette. On peut créer un objet simplement sans passer de paramètre, mais il est possible d'être plus précis en indiquant des fanions. Par exemple, pour avoir des dessins plus nets (mais peut-être plus gourmands en ressources), on ajoute les fanions Paint.ANTI_ALIAS_FLAG et Paint.DITHER_FLAG :

```
1 Paint p = new Paint(Paint.ANTI_ALIAS_FLAG | Paint.DITHER_FLAG);
```

La première chose est de déterminer ce qu'on veut dessiner : les contours d'une figure sans son intérieur, ou uniquement l'intérieur, ou bien même les contours *et* l'intérieur ? Afin d'assigner une valeur, on utilise void setStyle(Paint.Style style) :

```
1  Paint p = new Paint();
2
3  // Dessiner l'intérieur d'une figure
4  p.setStyle(Paint.Style.FILL);
5
6  // Dessiner ses contours
7  p.setStyle(Paint.Style.STROKE);
8
9  // Dessiner les deux
10 p.setStyle(Paint.Style.FILL_AND_STROKE);
```

On peut ensuite assigner une couleur avec `void setColor(int color)`. Comme vous pouvez le voir, cette méthode prend un entier, mais quelles valeurs peut-on lui donner ? Eh bien, pour vous aider dans cette tâche, Android fournit la classe `Color` qui va calculer pour vous la couleur en fonction de certains paramètres que vous passerez. Je pense particulièrement à `static int argb(int alpha, int red, int green, int blue)` qui dépend de la valeur de chaque composante (respectivement la transparence, le rouge, le vert et le bleu). On peut aussi penser à `static int parseColor(String colorString)` qui prend une chaîne de caractères comme on pourrait les trouver sur internet :

```
p.setColor(Color.parseColor("#12345678"));
```

Le peintre

Enfin, on va pouvoir peindre ! Ici, rien de formidable, il existe surtout des méthodes qui expriment la forme à représenter. Tout d'abord, n'oubliez pas de donner un `Bitmap` au `Canvas`, sinon il n'aura pas de surface sur laquelle dessiner :

```
Bitmap b = Bitmap.createBitmap(128, 128, Config.ARGB_8888);
Canvas c = new Canvas(b);
```

C'est tout ! Ensuite, pour dessiner une figure, il suffit d'appeler la méthode appropriée. Par exemple :

- `void drawColor(int color)` pour remplir la surface du `Bitmap` d'une couleur.
- `void drawRect(Rect r, Paint paint)` pour dessiner un rectangle.
- `void drawText(String text, float x, float y, Paint paint)` afin de dessiner... du texte. Eh oui, le texte se dessine aussi.

Vous trouverez plus de méthodes sur la page qui y est consacrée sur le site d'Android Developers.

Afficher notre toile

Il existe deux manières pour afficher nos œuvres d'art : sur n'importe quelle vue, ou sur une surface dédiée à cette tâche.

Sur une vue standard

Cette solution est la plus intéressante si votre surface de dessin n'a pas besoin d'être rapide ou fluide. C'est le cas quand on veut faire une banale vue personnalisée, mais pas quand on veut faire un jeu.

Il n'y a pas grand-chose à dire ici que vous ne sachiez déjà. Les dessins seront à effectuer dans la méthode de *callback* `void onDraw (Canvas canvas)` qui vous fournit le `Canvas` sur lequel dessiner. Ce `Canvas` contient déjà un `Bitmap` qui représente le dessin de la vue.

Cette méthode onDraw(Canvas) sera appelée à chaque fois que la vue juge que c'est nécessaire. Si vous voulez indiquer manuellement à la vue qu'elle doit se redessiner *le plus vite possible*, on peut le faire en utilisant la méthode void invalidate().

Un appel à la méthode invalidate() n'est pas nécessairement instantané, il se peut qu'elle prenne un peu de temps puisque cet appel doit se faire dans le thread UI et passera par conséquent après toutes les actions en cours. Il est d'ailleurs possible d'invalider une vue depuis un autre thread avec la méthode void postInvalidate().

Sur une surface dédiée à ce travail

Cette solution est déjà plus intéressante dès qu'il s'agit de faire un jeu, parce qu'elle permet de dessiner dans des threads différents du thread UI. Ainsi, au lieu d'avoir à attendre qu'Android déclare à notre vue qu'elle peut se redessiner, on aura notre propre thread dédié à cette tâche, donc sans encombrer le thread UI. Mais ce n'est pas tout ! En plus d'être plus rapide, cette surface peut être prise en charge par OpenGL si vous voulez effectuer des opérations graphiques encore plus compliquées.

Techniquement, la classe sur laquelle nous allons dessiner s'appelle SurfaceView. Cependant, nous n'allons pas la manipuler directement, nous allons passer par une couche d'abstraction représentée par la classe SurfaceHo
lder. Afin de récupérer un SurfaceHolder depuis un SurfaceView, il suffit d'appeler SurfaceHolder getHolder(). De plus, pour gérer correctement le cycle de vie de notre SurfaceView, on aura besoin d'implémenter l'interface SurfaceHolder.Callback, qui permet au SurfaceView de recevoir des informations sur les différentes phases et modifications qu'elle expérimente. Pour associer un SurfaceView à un SurfaceHolder.Call
back, on utilise la méthode void addCallback(SurfaceHolder.Callback callback) sur le SurfaceHolder associé au SurfaceView. Cette opération doit être effectuée dès la création du SurfaceView afin de pouvoir prendre en compte son commencement.

N'ayez toujours qu'un thread au maximum qui manipule une SurfaceView, sinon gare aux soucis !

```
1  import android.content.Context;
2  import android.util.AttributeSet;
3  import android.view.SurfaceHolder;
4  import android.view.SurfaceView;
5
6  public class ExampleSurfaceView extends SurfaceView implements
       SurfaceHolder.Callback {
7    private SurfaceHolder mHolder = null;
8
9    /**
10   * Utilisé pour construire la vue en Java
11   * @param context le contexte qui héberge la vue
12   */
13   public ExampleSurfaceView(Context context) {
```

```
14        super(context);
15        init();
16      }
17
18      /**
19      * Utilisé pour construire la vue depuis XML sans style
20      * @param context le contexte qui héberge la vue
21      * @param attrs les attributs définis en XML
22      */
23      public ExampleSurfaceView(Context context, AttributeSet attrs
            ) {
24        super(context, attrs);
25        init();
26      }
27
28      /**
29      * Utilisé pour construire la vue depuis XML avec un style
30      * @param context le contexte qui héberge la vue
31      * @param attrs les attributs définis en XML
32      * @param defStyle référence au style associé
33      */
34      public ExampleSurfaceView(Context context, AttributeSet attrs
            , int defStyle) {
35        super(context, attrs, defStyle);
36        init();
37      }
38
39      public void init() {
40        mHolder = getHolder();
41        mHolder.addCallback(this);
42      }
43
44    }
```

Ainsi, il nous faudra implémenter trois méthodes de *callback* qui réagiront à trois évènements différents :

- `void surfaceChanged(SurfaceHolder holder, int format, int width, int height)` sera enclenché à chaque fois que la surface est modifiée, c'est donc ici qu'on mettra à jour notre image. Mis à part certains paramètres que vous connaissez déjà tels que la largeur `width`, la hauteur `height` et le format `PixelFormat`, on trouve un nouvel objet de type `SurfaceHolder`. Un `SurfaceHolder` est une interface qui représente la surface sur laquelle dessiner. Mais ce n'est pas avec lui qu'on dessine, c'est bien avec un `Canvas`, de façon à ne pas manipuler directement le `SurfaceView`.
- `void surfaceCreated(SurfaceHolder holder)` sera quant à elle déclenchée uniquement à la création de la surface. On peut donc commencer à dessiner ici.
- Enfin, `void surfaceDestroyed(SurfaceHolder holder)` est déclenchée dès que la surface est détruite, de façon à ce que vous sachiez quand arrêter votre thread. Après que cette méthode a été appelée, la surface n'est plus disponible du tout.

Passons maintenant au dessin en tant que tel. Comme d'habitude, il faudra dessiner à l'aide d'un `Canvas`, sachant qu'il a déjà un `Bitmap` attribué. Comme notre dessin est dynamique, il faut d'abord bloquer le `Canvas`, c'est-à-dire immobiliser l'image actuelle pour pouvoir dessiner dessus. Pour bloquer le `Canvas`, il suffit d'utiliser la méthode `Canvas lockCanvas()`. Puis, une fois votre dessin terminé, vous pouvez le remettre en route avec `void unlockCanvasAndPost(Canvas canvas)`. C'est indispensable, sinon votre téléphone restera bloqué.

La surface sur laquelle se fait le dessin sera supprimée à chaque fois que l'activité se met en pause et sera recréée dès que l'activité reprendra, il faut donc interrompre le dessin à ces moments-là.

Pour économiser un peu notre processeur, on va instaurer une pause dans la boucle principale. En effet, si on ne fait pas de boucle, le thread va dessiner sans cesse le plus vite, alors que l'oeil humain ne sera pas capable de voir la majorité des images qui seront dessinées. C'est pourquoi nous allons rajouter un morceau de code qui impose au thread de ne plus calculer pendant 20 millisecondes. De cette manière, on affichera 50 images par seconde en moyenne, l'illusion sera parfaite pour l'utilisateur et la batterie de vos utilisateurs vous remercie déjà :

```
1 try {
2     Thread.sleep(20);
3 } catch (InterruptedException e) {}
```

Voici un exemple d'implémentation de `SurfaceView` :

```
1  package sdz.chapitreQuatre.surfaceexample;
2
3  import android.content.Context;
4  import android.graphics.Canvas;
5  import android.util.AttributeSet;
6  import android.view.SurfaceHolder;
7  import android.view.SurfaceView;
8
9  public class ExampleSurfaceView extends SurfaceView implements
       SurfaceHolder.Callback {
10     // Le holder
11     SurfaceHolder mSurfaceHolder;
12     // Le thread dans lequel le dessin se fera
13     DrawingThread mThread;
14
15     public ExampleSurfaceView (Context context) {
16         super(context);
17         mSurfaceHolder = getHolder();
18         mSurfaceHolder.addCallback(this);
19
20         mThread = new DrawingThread();
21     }
22
23     @Override
```

```
24      protected void onDraw(Canvas pCanvas) {
25          // Dessinez ici !
26      }
27
28      @Override
29      public void surfaceChanged(SurfaceHolder holder, int format
            , int width, int height) {
30          // Que faire quand le surface change ? (L'utilisateur
                tourne son téléphone par exemple)
31      }
32
33      @Override
34      public void surfaceCreated(SurfaceHolder holder) {
35          mThread.keepDrawing = true;
36          mThread.start();
37      }
38
39      @Override
40      public void surfaceDestroyed(SurfaceHolder holder) {
41          mThread.keepDrawing = false;
42
43          boolean joined = false;
44          while (!joined) {
45              try {
46                  mThread.join();
47                  joined = true;
48              } catch (InterruptedException e) {}
49          }
50      }
51
52      private class DrawingThread extends Thread {
53          // Utilisé pour arrêter le dessin quand il le faut
54          boolean keepDrawing = true;
55
56          @Override
57          public void run() {
58
59              while (keepDrawing) {
60                  Canvas canvas = null;
61
62                  try {
63                      // On récupère le canvas pour dessiner
                            dessus
64                      canvas = mSurfaceHolder.lockCanvas();
65                      // On s'assure qu'aucun autre thread n'accè
                            de au holder
66                      synchronized (mSurfaceHolder) {
67                          // Et on dessine
68                          onDraw(canvas);
69                      }
```

```
70                  } finally {
71                      // Notre dessin fini, on relâche le Canvas
                            pour que le dessin s'affiche
72                      if (canvas != null)
73                          mSurfaceHolder.unlockCanvasAndPost(
                                canvas);
74                  }
75
76                  // Pour dessiner à 50 fps
77                  try {
78                      Thread.sleep(20);
79                  } catch (InterruptedException e) {}
80              }
81          }
82      }
83  }
```

En résumé

- On a besoin de plusieurs éléments pour dessiner : un `Canvas`, un `Bitmap` et un `Paint`.
- Le `Bitmap` est l'objet qui contiendra le dessin, on peut le comparer à la toile d'un tableau.
- Un `Paint` est tout simplement un objet qui représente un pinceau, on peut lui attribuer une couleur ou une épaisseur par exemple.
- Pour faire le lien entre une toile et un pinceau, on a besoin d'un peintre ! Ce peintre est un `Canvas`. Il contient un `Bitmap` et dessine dessus avec un `Paint`. Il est quand même assez rare qu'on fournisse un `Bitmap` à un `Canvas`, puisqu'en général la vue qui affichera le dessin nous fournira un `Canvas` qui possède déjà un `Bitmap` tout configuré.
- En tant que programmeur, vous êtes un client qui ordonne au peintre d'effectuer un dessin, ce qui fait que vous n'avez pas à vous préoccuper de manipuler le pinceau ou la toile, le peintre le fera pour vous.
- Pour afficher un dessin, on peut soit le faire avec une vue comme on l'a vu dans la seconde partie, soit créer carrément une surface dédiée au dessin. Cette solution est à privilégier quand on veut créer un jeu par exemple. Le plus gros point faible est qu'on doit utiliser des threads.

Chapitre 24

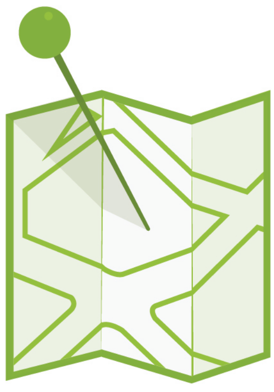
La localisation et les cartes

Difficulté :

Nous sommes nombreux à avoir déjà utilisé Google Maps. Que ce soit pour trouver le vendeur de pizzas le plus proche, tracer l'itinéraire entre chez soi et le supermarché ou, encore mieux, regarder sa propre maison avec les images satellite.

Avec les progrès de la miniaturisation, la plupart — voire la quasi-totalité — des terminaux sont équipés de puces GPS. La géolocalisation est ainsi devenue un élément du quotidien qu'on retrouve dans énormément d'applications. On peut penser aux applications de navigation aidée par GPS, mais aussi aux applications sportives qui suivent nos efforts et élaborent des statistiques, ou encore aux applications pour noter les restaurants et les situer. On trouve ainsi deux API qui sont liées au concept de localisation :

- Une API qui permet de localiser l'appareil.
- Une API qui permet d'afficher des cartes.



La localisation

Préambule

On trouve tous les outils de localisation dans le package `android.location`.

Le GPS est la solution la plus efficace pour localiser un appareil, cependant il s'agit aussi de la plus coûteuse en batterie. Une autre solution courante est de se localiser à l'aide des points d'accès WiFi à proximité et de la distance mesurée avec les antennes relais du réseau mobile les plus proches (par triangulation).

Tout d'abord, vous devrez demander la permission dans le Manifest pour utiliser les fonctionnalités de localisation. Si vous voulez utiliser la géolocalisation (par GPS, donc), utilisez `ACCESS_FINE_LOCATION` ; pour une localisation plus imprécise par WiFi et antennes relais, utilisez `ACCESS_COARSE_LOCATION`. Enfin, si vous voulez utiliser les deux types de localisation, vous pouvez déclarer uniquement `ACCESS_FINE_LOCATION`, qui comprend toujours `ACCESS_COARSE_LOCATION` :

```
1 <uses-permission android:name="android.permission.
      ACCESS_FINE_LOCATION" />
2 <uses-permission android:name="android.permission.
      ACCESS_COARSE_LOCATION" />
```

Ensuite, on va faire appel à un nouveau service système pour accéder à ces fonctionnalités : `LocationManager`, que l'on récupère de cette manière :

```
1 LocationManager locationManager = (LocationManager)
      getSystemService(Context.LOCATION_SERVICE);
```

Les fournisseurs de position

Vous aurez ensuite besoin d'un fournisseur de position qui sera dans la capacité de déterminer la position actuelle. On a par un exemple un fournisseur pour le GPS et un autre pour les antennes relais. Ces fournisseurs dériveront de la classe abstraite `LocationProvider`. Il existe plusieurs méthodes pour récupérer les fournisseurs de position disponibles sur l'appareil. Pour récupérer le nom de tous les fournisseurs, il suffit de faire `List<String> getAllProviders()`. Le problème de cette méthode est qu'elle va récupérer tous les fournisseurs qui existent, même si l'application n'a pas le droit de les utiliser ou qu'ils sont désactivés par l'utilisateur.

Pour ne récupérer que le nom des fournisseurs qui sont réellement utilisables, on utilisera `List<String> getProviders(boolean enabledOnly)`. Enfin, on peut obtenir un `LocationProvider` à partir de son nom avec `LocationProvider getProvider(String name)` :

```
ArrayList<LocationProvider> providers = new ArrayList<
    LocationProvider>();
ArrayList<String> names = locationManager.getProviders(true);

for(String name : names)
  providers.add(locationManager.getProvider(name));
```

Les noms des fournisseurs sont contenus dans des constantes, comme `LocationManager.GPS_PROVIDER` pour le GPS et `LocationManager.NETWORK_PROVIDER` pour la triangulation.

Cependant, il se peut que vous ayez à sélectionner un fournisseur en fonction de critères bien précis. Pour cela, il vous faudra créer un objet de type `Criteria`. Par exemple, pour configurer tous les critères, on fera :

```
Criteria critere = new Criteria();

// Pour indiquer la précision voulue
// On peut mettre ACCURACY_FINE pour une haute précision ou
    ACCURACY_COARSE pour une moins bonne précision
critere.setAccuracy(Criteria.ACCURACY_FINE);

// Est-ce que le fournisseur doit être capable de donner une
    altitude ?
critere.setAltitudeRequired(true);

// Est-ce que le fournisseur doit être capable de donner une
    direction ?
critere.setBearingRequired(true);

// Est-ce que le fournisseur peut être payant ?
critere.setCostAllowed(false);

// Pour indiquer la consommation d'énergie demandée
// Criteria.POWER_HIGH pour une haute consommation, Criteria.
    POWER_MEDIUM pour une consommation moyenne et Criteria.
    POWER_LOW pour une basse consommation
critere.setPowerRequirement(Criteria.POWER_HIGH);

// Est-ce que le fournisseur doit être capable de donner une
    vitesse ?
critere.setSpeedRequired(true);
```

Pour obtenir tous les fournisseurs qui correspondent à ces critères, on utilise `List<String> getProviders(Criteria criteria, boolean enabledOnly)` et, pour obtenir le fournisseur qui correspond le plus, on utilise `String getBestProvider(Criteria criteria, boolean enabledOnly)`.

Obtenir des notifications du fournisseur

Pour obtenir la dernière position connue de l'appareil, utilisez ce code :

```
Location getLastKnownLocation(String provider)
```

La dernière position connue n'est pas forcément la position *actuelle* de l'appareil. En effet, il faut demander à mettre à jour la position pour que celle-ci soit renouvelée dans le fournisseur. Si vous voulez faire en sorte que le fournisseur se mette à jour automatiquement à une certaine période ou tous les `x` mètres, on peut utiliser la méthode `void requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener)` avec :

- `provider` le fournisseur de position.
- `minTime` la période entre deux mises à jour en millisecondes. Il faut mettre une valeur supérieure à 0, sinon le fournisseur ne sera pas mis à jour périodiquement. D'ailleurs, ne mettez pas de valeur en dessous de 60 000 ms pour préserver la batterie.
- `minDistance` la période entre deux mises à jour en mètres. Tout comme pour `minTime`, il faut mettre une valeur supérieure à 0, sinon ce critère ne sera pas pris en compte. De plus, on privilégie quand même `minTime` parce qu'il consomme moins de batterie.
- `listener` est l'écouteur qui sera lancé dès que le fournisseur sera activé.

Ainsi, il faut que vous utilisiez l'interface `LocationListener`, dont la méthode `void onLocationChanged(Location location)` sera déclenchée à chaque mise à jour. Cette méthode de *callback* contient un objet de type `Location` duquel on peut extraire des informations sur l'emplacement donné. Par exemple, on peut récupérer la latitude avec `double getLatitude()` et la longitude avec `double getLongitude()` :

```
locationManager.requestLocationUpdates(LocationManager.
    GPS_PROVIDER, 60000, 150, new LocationListener() {

  @Override
  public void onStatusChanged(String provider, int status,
      Bundle extras) {

  }

  @Override
  public void onProviderEnabled(String provider) {

  }

  @Override
  public void onProviderDisabled(String provider) {

  }

  @Override
  public void onLocationChanged(Location location) {
    Log.d("GPS", "Latitude " + location.getLatitude() + " et
        longitude " + location.getLongitude());
```

```
21     }
22   });
```

Cependant, ce code ne fonctionnera que si votre application est en cours de fonctionnement. Mais si vous souhaitez recevoir des notifications même quand l'application ne fonctionne pas ? On peut utiliser à la place `void requestLocationUpdates(String provider, long minTime, float minDistance, PendingIntent intent)` où le `PendingIntent` contenu dans `intent` sera lancé à chaque mise à jour du fournisseur. L'emplacement sera contenu dans un extra dont la clé est `KEY_LOCATION_CHANGED`.

```
1  Intent intent = new Intent(this, GPSUpdateReceiver.class);
2
3  PendingIntent pending = PendingIntent.getBroadcast(this, 0,
       intent, PendingIntent.FLAG_UPDATE_CURRENT);
4
5  locationManager.requestLocationUpdates(provider, 60000, 150,
       pending);
```

On le recevra ensuite dans :

```
1  public class GPSUpdateReceiver extends BroadcastReceiver {
2    @Override
3    public void onReceive(Context context, Intent intent) {
4      Location location = (Location)intent.getParcelableExtra(
           LocationManager.KEY_LOCATION_CHANGED);
5    }
6  }
```

Enfin, vous pouvez désactiver les notifications avec `removeUpdates` en lui donnant le `LocationListener` ou le `PendingIntent` concerné. Si vous ne le faites pas, votre application continuera à recevoir des notifications après que tous les composants de l'application auront été fermés.

Les alertes de proximité

Dernière fonctionnalité que nous allons voir, le fait d'être informés quand on s'approche d'un endroit ou qu'on s'en éloigne. Cet endroit peut être symbolisé par un cercle dont on va préciser le centre et le rayon. Ainsi, si on entre dans ce cercle ou qu'on sort de ce cercle, l'alerte est lancée.

Le prototype de la méthode qui peut créer une alerte de proximité est `void addProximityAlert(double latitude, double longitude, float radius, long expiration, PendingIntent intent)` avec :

- La `latitude` et la `longitude` du centre du cercle.
- Le rayon d'effet est précisé dans `radius` en mètres.
- `expiration` permet de déclarer combien de temps cette alerte est valable. Tout nombre en dessous de 0 signifie qu'il n'y a pas d'expiration possible.
- Enfin, comme vous vous en doutez, on donne aussi le `PendingIntent` qui sera lancé quand cette alerte est déclenchée, avec `intent`.

Cette fois, l'intent contiendra un booléen en extra, dont la clé est `KEY_PROXIMITY_ENTERING` et la valeur sera `true` si on entre dans la zone et `false` si on en sort.

```
Intent intent = new Intent(this, AlertReceiver.class);

PendingIntent pending = PendingIntent.getBroadcast(this, 0,
    intent, PendingIntent.FLAG_UPDATE_CURRENT);

// On ajoute une alerte de proximité si on s'approche ou s'é
    loigne du bâtiment de Simple IT
locationManager.addProximityAlert(48.872808, 2.33517, 150, -1,
    pending);
```

On le recevra ensuite dans :

```
public class AlertReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    // Vaudra true par défaut si on ne trouve pas l'extra boolé
        en dont la clé est LocationManager.
        KEY_PROXIMITY_ENTERING
    bool entrer = booleanValue(intent.getBooleanExtra(
        LocationManager.KEY_PROXIMITY_ENTERING, true));
  }
}
```

Enfin, il faut désactiver une alerte de proximité avec `void removeProximityAlert (PendingIntent intent)`.

Afficher des cartes

C'est bien de pouvoir récupérer l'emplacement de l'appareil à l'aide du GPS, mais il faut avouer que si on ne peut pas l'afficher sur une carte c'est sacrément moins sympa ! Pour cela, on va passer par l'API Google Maps.

Contrairement aux API que nous avons vues pour l'instant, l'API pour Google Maps n'est pas intégrée à Android, mais appartient à une extension appelée « Google APIs », comme nous l'avons vu au cours des premiers chapitres. Ainsi, quand vous allez créer un projet, au lieu de sélectionner `Android 2.1 (API 7)`, on va sélectionner `Google APIs (Google Inc.) (API 7)`. Et si vous ne trouvez pas `Google APIs (Google Inc.) (API 7)`, c'est qu'il vous faudra le télécharger, auquel cas je vous renvoie au chapitre 2 de la première partie qui traite de ce sujet. Il faut ensuite l'ajouter dans le Manifest. Pour cela, on doit ajouter la ligne suivante dans le nœud `application` :

```
<uses-library android:name="com.google.android.maps" />
```

Cette opération rendra votre application invisible sur le Play Store si l'appareil n'a pas Google Maps. De plus, n'oubliez pas d'ajouter une permission pour accéder à internet, les cartes ne se téléchargent pas par magie :

```
<uses-permission android:name="android.permission.INTERNET" />
```

Obtenir une clé pour utiliser Google Maps

Pour pouvoir utiliser Google Maps, il vous faudra demander l'autorisation pour accéder aux services sur internet. Pour cela, vous allez demander une clé.

▷ Demander une clé
Code web : 898915

Comme vous pouvez le voir, pour obtenir la clé, vous aurez besoin de l'empreinte MD5 du certificat que vous utilisez pour signer votre application. Si vous ne comprenez pas un traître mot de ce que je viens de dire, c'est que vous n'avez pas lu l'annexe sur la publication d'applications, ce que je vous invite à faire, en particulier la partie sur la signature (page 541).

Ensuite, la première chose à faire est de repérer où se trouve votre certificat. Si vous travaillez en mode debug, alors Eclipse va générer un certificat pour vous :

– Sous Windows, le certificat par défaut se trouve dans le répertoire consacré à votre compte utilisateur dans `\.android\debug.keystore`. Si vous avez Windows Vista, 7 ou 8, alors ce répertoire utilisateur se trouve par défaut dans `C:\Users\nom_d_utilisateur`. Pour Windows XP, il se trouve dans `C:\DocumentsandSettings\nom_d_utilisateur`.
– Pour Mac OS et Linux, il se trouve dans `~/.android/`.

Puis il vous suffira de lancer la commande suivante dans un terminal :

```
keytool -list -keystore "Emplacement du certificat" -storepass
   android -keypass android
```

Deux choses auxquelles vous devez faire attention :

– Si cette commande ne marche pas, c'est que vous avez mal configuré votre `PATH` au moment de l'installation de Java. Ce n'est pas grave, il vous suffit d'aller à l'emplacement où se trouve l'outil *keytool* et d'effectuer la commande. Chez moi, il s'agit de `C:\Program Files (x86)\Java\jre7\bin`.
– Et si comme moi vous utilisez Java 7 au lieu de Java 6, vous devrez rajouter `-v` à la commande, ce qui donne `keytool -list -keystore "Emplacement du certificat" -storepass android -keypass android -v`

Ainsi, ce que j'ai fait pour obtenir ma clé MD5, c'est :

– Aller dans `C:\Program Files (x86)\Java\jre7\bin` ;
– Taper `keytool -list -keystore "C:\Users\Apollidore\.android\debug.keystore" -storepass android -keypass android -v` ;
– Repérer la ligne où il était écrit « MD5 », comme à la figure 24.1.

Insérez ensuite le code MD5 sur le site pour obtenir une clé que vous pourrez utiliser dans l'application, mais nous verrons comment procéder plus tard.

```
C:\Program Files (x86)\Java\jre7\bin>keytool -list -keystore "C:\Users\DakuTenshi\.android\debug.keystore" -v -storepass android -keypass android

Type de fichier de clÚs : JKS
Fournisseur de fichier de clÚs : SUN

Votre fichier de clÚs d'accÞs contient 1 entrÚe

Nom d'alias : androiddebugkey
Date de crÚation : 4 juil. 2012
Type d'entrÚeá: PrivateKeyEntry
Longueur de cha¯ne du certificat : 1
Certificat[1]:
PropriÚtaire : CN=Android Debug, O=Android, C=US
Emetteur : CN=Android Debug, O=Android, C=US
NumÚro de sÚrie : 5adb67e7
Valide du : Wed Jul 04 13:19:24 CEST 2012 au : Fri Jun 27 13:19:24 CEST 2042
Empreintes du certificat :
         MD5:
         SHA1 :
         SHA256 :
         Nom de l'algorithme de signature : SHA256withRSA
         Version : 3

Extensionsá:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [

]
]
```

FIGURE 24.1 – Comme vous pouvez le voir, on évite de montrer à tout le monde les codes obtenus, sinon on pourrait se faire passer pour vous

L'activité qui contiendra la carte

Tout d'abord, pour simplifier l'utilisation des cartes, chaque activité qui contiendra une carte sera de type `MapActivity`. Vous aurez besoin d'implémenter au moins deux méthodes : `onCreate(Bundle)` et `protected boolean isRouteDisplayed()`. Cette méthode permet de savoir si d'une manière ou d'une autre la vue qui affichera la carte permettra de visualiser des informations de type itinéraire ou parcours. Renvoyez `true` si c'est le cas, sinon renvoyez `false`. Enfin, vous devez aussi implémenter la méthode `protected boolean isLocationDisplayed()` qui renverra `true` si votre application affiche l'emplacement actuel de l'utilisateur.

Vous devez implémenter ces deux méthodes pour utiliser légalement cette API.

```java
import android.os.Bundle;

import com.google.android.maps.MapActivity;

public class MainActivity extends MapActivity {

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  @Override
  protected boolean isRouteDisplayed() {
    return false;
  }
```

```
17
18    @Override
19    protected boolean isLocationDisplayed() {
20      return true;
21    }
22  }
```

La carte en elle-même

Pour gérer l'affichage de la carte, on va utiliser une `MapView` qui se trouve elle aussi dans le package `com.google.android.maps`. Au moment de la déclaration en XML, on va penser surtout à deux attributs :

- `android:clickable`, si vous voulez que l'utilisateur puisse cliquer sur la carte ou se déplacer dans la carte ;
- `android:apiKey`, pour préciser la clé que vous avez récupérée sur le site.

Par exemple :

```
1 <com.google.android.maps.MapView android:id="+id/mapView"
2   android:layout_width="fill_parent"
3   android:layout_height="fill_parent"
4   android:clickable="true"
5   android:apiKey="votre clé" />
```

Ne mettez pas plus d'une `MapActivity` et une `MapView` par application, ce n'est pas très bien supporté pour l'instant, elles pourraient entrer en conflit.

Comme vous le savez probablement, il existe trois modes d'affichage sur une carte Google Maps :

- Si vous voulez afficher la vue satellitaire, utilisez `mapView.setSatellite(true)`.
- Si vous voulez afficher les routes, les noms des rues et tout ce qu'on peut attendre d'une carte routière, utilisez `mapView.setTraffic(true)`.
- Et si vous voulez afficher la possibilité de passer en mode Street View (vous savez, ce mode qui prend le point de vue d'un piéton au milieu d'une rue), alors utilisez `mapView.setStreetView(true)`. Ce mode n'est pas compatible avec le mode trafic.

Le contrôleur

Votre carte affiche ce que vous voulez… enfin presque. Si vous voulez faire un zoom ou déplacer la région actuellement visualisée, il vous faudra utiliser un `MapController`. Pour récupérer le `MapController` associé à une `MapView`, il suffit de faire `MapController getController()`.

Le zoom

Il existe 21 niveaux de zoom, et chacun d'eux représente un doublement dans le nombre de pixels qu'affiche une même zone. Ainsi en zoom 1, on peut voir toute la planète (plusieurs fois), alors qu'en zoom 21 on peut voir le chien du voisin. Pour contrôler le zoom, utilisez sur le contrôleur la méthode `int setZoom(int zoomLevel)` qui retourne le nouveau niveau de zoom. Vous pouvez zoomer et dézoomer d'un niveau avec respectivement `boolean zoomIn()` et `boolean zoomOut()`.

Se déplacer dans la carte

Pour modifier l'emplacement qu'affiche le centre de la carte, il faut utiliser la méthode `void setCenter(GeoPoint point)` (ou `public void animateTo(GeoPoint point)` si vous voulez une animation). Comme vous pouvez le voir, ces deux méthodes prennent des objets de type `GeoPoint`. Très simplement, un `GeoPoint` est utilisé pour représenter un emplacement sur la planète, en lui donnant une longitude et une latitude. Ainsi, le `GeoPoint` qui représente le bâtiment où se situe Simple IT sera créé de cette manière :

```
// Comme les unités sont en microdegrés, il faut multiplier par
    1E6
int latitude = 48.872808 * 1E6;
int longitude = 2.33517 * 1E6;

GeoPoint simpleIt = new GeoPoint(latitude.intValue(), longitude
    .intValue());
```

Ce qui est pratique, c'est que ce calque permet d'effectuer une action dès que le GPS détecte la position de l'utilisateur avec la méthode `boolean runOnFirstFix(Runnable runnable)`, par exemple pour zoomer sur la position de l'utilisateur à ce moment-là :

```
overlay.runOnFirstFix(new Runnable() {
  @Override
  public void run() {
    mMapView.getController().animateTo(overlay.getMyLocation())
        ;
  }
});
```

Utiliser les calques pour afficher des informations complémentaires

Parfois, afficher une carte n'est pas suffisant, on veut en plus y ajouter des informations. Et si je veux afficher Zozor, la mascotte du Site du Zéro, sur ma carte à l'emplacement où se trouve Simple IT ? Et si en plus je voulais détecter les clics des utilisateurs sur un emplacement de la carte ? Il est possible de rajouter plusieurs couches sur lesquelles dessiner et qui sauront réagir aux évènements communs. Ces couches sont des calques, qui sont des objets de type `Overlay`.

Ajouter des calques

Pour récupérer la liste des calques que contient la carte, on fait `List<Overlay> getOverlays()`. Il suffit d'ajouter des `Overlay` à cette liste pour qu'ils soient dessinés sur la carte. Vous pouvez très bien accumuler les calques sur une carte, de manière à dessiner des choses les unes sur les autres. Chaque calque ajouté se superpose aux précédents, il se place au-dessus. Ainsi, les dessins de ce nouveau calque se placeront au-dessus des précédents et les évènements, par exemple les touchers, seront gérés de la manière suivante : le calque qui se trouve au sommet recevra en premier l'évènement. Si ce calque n'est pas capable de gérer cet évènement, ce dernier est alors transmis aux calques qui se trouvent en dessous. Néanmoins, si le calque est effectivement capable de gérer cet évènement, alors il s'en charge et l'évènement ne sera plus propagé.

Enfin, pour indiquer qu'on a rajouté un `Overlay`, on utilise la méthode `void postInvalidate()` pour faire se redessiner la `MapView` :

```
1 List<Overlay> overlays = mapView.getOverlays();
2 OverlayExample o = new OverlayExample();
3 overlays.add(o);
4 mapView.postInvalidate();
```

Dessiner sur un calque

Cependant, si vous voulez ajouter un point d'intérêt — on dit aussi un « POI » —, c'est-à-dire un endroit remarquable sur la carte, je vous recommande plutôt d'utiliser la classe `ItemizedOverlay`.

On implémente en général au moins deux méthodes. La première est `void draw(Canvas canvas, MapView mapView, boolean shadow)` qui décrit le dessin à effectuer. On dessine sur le `canvas` et on projette le dessin du `Canvas` sur la vue `mapView`. En ce qui concerne `shadow`, c'est plus compliqué. En fait, cette méthode `draw` est appelée deux fois : une première fois où `shadow` vaut `false` pour dessiner normalement et une seconde où `shadow` vaut `true` pour rajouter des ombres à votre dessin.

On a un problème d'interface entre le `Canvas` et la `MapView` : les coordonnées sur le `Canvas` sont en `Point` et les coordonnées sur la `MapView` sont en `GeoPoint`, il nous faut donc un moyen pour convertir nos `Point` en `GeoPoint`. Pour cela, on utilise une `Projection` avec la méthode `Projection getProjection()` sur la `MapView`. Pour obtenir un `GeoPoint` depuis un `Point`, on peut faire `GeoPoint fromPixels(int x, int y)` et, pour obtenir un `Point` depuis un `GeoPoint`, on peut faire `Point toPixels(GeoPoint in, Point out)`. Par exemple :

```
1 Point point = new Point(10, 10);
2 // Obtenir un GeoPoint
3 GeoPoint geo = projection.fromPixels(point.x, point.y);
4
5 Point nouveauPoint = null;
6 // Obtenir un Point. On ignore le retour pour passer l'objet en
      paramètre pour des raisons d'optimisation
```

```
7 projection.toPixels(geo, nouveauPoint);
```

Ainsi, pour dessiner un point rouge à l'emplacement du bâtiment de Simple IT :

```
1  import android.graphics.Canvas;
2  import android.graphics.Paint;
3  import android.graphics.Point;
4  import android.graphics.RectF;
5
6  import com.google.android.maps.GeoPoint;
7  import com.google.android.maps.MapView;
8  import com.google.android.maps.Overlay;
9  import com.google.android.maps.Projection;
10
11 public class PositionOverlay extends Overlay {
12   private int mRadius = 5;
13
14   // Coordonnées du bâtiment de Simple IT
15   private Double mLatitutde = 48.872808*1E6;
16   private Double mLongitude = 2.33517*1E6;
17
18   public PositionOverlay() {
19
20   }
21
22   @Override
23   public void draw(Canvas canvas, MapView mapView, boolean
        shadow) {
24     Projection projection = mapView.getProjection();
25
26     GeoPoint geo = new GeoPoint(mLatitutde.intValue(),
         mLongitude.intValue());
27
28     if (!shadow) {
29       Point point = new Point();
30
31       // Convertir les points géographiques en points pour le
           Canvas
32       projection.toPixels(geo, point);
33
34       //  Créer le pinceau
35       Paint paint = new Paint();
36       paint.setARGB(255, 255, 0, 0);
37
38       // Création du cercle
39       RectF cercle = new RectF(point.x - mRadius, point.y -
           mRadius, point.x + mRadius, point.y + mRadius);
40
41       // Dessine le cercle
42       canvas.drawOval(cercle, paint);
43     }
```

```
44      }
45
46  }
```

Gérer les évènements sur un calque

La méthode de *callback* qui sera appelée quand l'utilisateur appuiera sur le calque s'appelle `boolean onTap(GeoPoint p, MapView mapView)` avec `p` l'endroit où l'utilisateur a appuyé et `mapView` la carte sur laquelle il a appuyé. Il vous est demandé de renvoyer `true` si l'évènement a été géré (auquel cas il ne sera plus transmis aux couches qui se trouvent en dessous).

```
1  @Override
2  public boolean onTap(GeoPoint point, MapView mapView) {
3    if (/** Test **/) {
4      // Faire quelque chose
5      return true;
6    }
7    return false;
8  }
```

Quelques calques spécifiques

Maintenant que vous pouvez créer tous les calques que vous voulez, nous allons en voir quelques-uns qui permettent de nous faciliter grandement la vie dès qu'il s'agit de faire quelques tâches standards.

Afficher la position actuelle

Les calques de type `MyLocationOverlay` permettent d'afficher votre position actuelle ainsi que votre orientation à l'aide d'un capteur qui est disponible dans la plupart des appareils de nos jours. Pour activer l'affichage de la position actuelle, il suffit d'utiliser `boolean enableMyLocation()` et, pour afficher l'orientation, il suffit d'utiliser `boolean enableCompass()`. Afin d'économiser la batterie, il est conseillé de désactiver ces deux fonctionnalités quand l'activité passe en pause, puis de les réactiver après :

```
1  public void onResume() {
2    super.onResume();
3    location.enableCompass();
4    location.enableMyLocation();
5  }
6
7  public void onPause() {
8    super.onPause();
9    location.disableCompass();
10   location.disableMyLocation();
11 }
```

Ajouter des marqueurs

Pour marquer l'endroit où se trouvait le bâtiment de Simple IT, nous avons ajouté un rond rouge sur la carte, cependant il existe un type d'objets qui permet de faire ce genre de tâches très facilement. Il s'agit d'`OverlayItem`. Vous aurez aussi besoin d'`ItemizedOverlay` qui est une liste d'`OverlayItem` à gérer et c'est elle qui fera les dessins, la gestion des évènements, etc.

Nous allons donc créer une classe qui dérive d'`ItemizedOverlay<OverlayItem>`. Nous aurons ensuite à nous occuper du constructeur. Dans celui-ci, il faudra passer un `Drawable` qui représentera le marqueur visuel de nos points d'intérêt. De plus, dans le constructeur, il vous faudra construire les différents `OverlayItem` et déclarer quand vous aurez fini avec la méthode `void populate()` :

```
public class ZozorOverlay extends ItemizedOverlay<OverlayItem>
    {
  private List<OverlayItem> mItems = new ArrayList<OverlayItem
      >();

  public ZozorOverlay(Drawable defaultMarker) {
    super(defaultMarker);

    Double latitude = 48.872808*1E6;
    Double longitude = 2.33517*1E6;
    mItems.add(new OverlayItem(new GeoPoint(latitude.intValue()
        , longitude.intValue()), "Simple IT", "Maison du Site
        du Zéro"));
    populate();
  }

  // ...
}
```

Vous remarquerez qu'un `OverlayItem` prend trois paramètres :

- Le premier est le `GeoPoint` où se trouve l'objet sur la carte.
- Le deuxième est le titre à attribuer au point d'intérêt.
- Le dernier est un court texte descriptif.

Enfin, il existe deux autres méthodes que vous devez implémenter :

- `int size()`, qui retourne le nombre de points d'intérêt dans votre liste.
- `Item createItem(int i)`, pour retourner le i-ième élément de votre liste.

En résumé

- Il existe plusieurs manières de détecter la position d'un appareil. La plus efficace est la méthode qui exploite la puce GPS, mais c'est aussi la plus consommatrice en énergie. La seconde méthode se base plutôt sur les réseaux Wifi et les capteurs internes du téléphone. Elle est beaucoup moins précise mais peut être utilisée en

dernier recours si l'utilisateur a désactivé l'utilisation de la puce GPS.

- Il est possible de récupérer la position de l'utilisateur avec le `LocationManager`. Il est aussi possible de créer des évènements qui permettent de réagir au statut de l'utilisateur avec `LocationListener`.
- Il est possible d'afficher des cartes fournies par Google Maps avec l'API Google Maps. Ainsi, on gérera une carte avec une `MapActivity` et on l'affichera dans une `MapView`.
- Le plus intéressant est de pouvoir ajouter des calques, afin de fournir des informations basées sur la géographie à l'utilisateur. On peut par exemple proposer à un utilisateur d'afficher les restaurants qui se trouvent à proximité. Un calque est un objet de type `Overlay`.

Chapitre 25

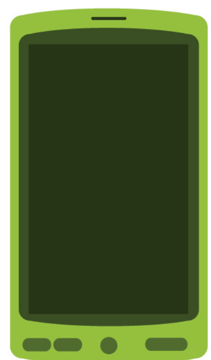
La téléphonie

Difficulté :

Il y a de grandes chances pour que votre appareil sous Android soit un téléphone. Et comme tous les téléphones, il est capable d'appeler ou d'envoyer des messages. Et comme nous sommes sous Android, il est possible de supplanter ces fonctionnalités natives pour les gérer nous-mêmes.

Encore une fois, tout le monde n'aura pas besoin de ce dont on va parler. Mais il peut très bien arriver que vous ayez envie qu'appuyer sur un bouton appelle un numéro d'urgence, ou le numéro d'un médecin, ou quoi que ce soit.

De plus, il faut savoir que le SMS — vous savez les petits messages courts qui font 160 caractères au maximum — est le moyen le plus courant pour communiquer entre deux appareils mobiles, il est donc très courant qu'un utilisateur ait envie d'en envoyer un à un instant `t`. Même s'ils sont beaucoup moins utilisés, les MMS — comme un SMS, mais avec un média (son, vidéo ou image) qui l'accompagne — sont monnaie courante.



Téléphoner

La première chose qu'on va faire, c'est s'assurer que l'appareil sur lequel fonctionnera votre application peut téléphoner, sinon notre application de téléphonie n'aura absolument aucun sens. Pour cela, on va indiquer dans notre Manifest que l'application ne peut marcher sans la téléphonie, en lui ajoutant la ligne suivante :

```
<uses-feature android:name="android.hardware.telephony"
  android:required="true" />
```

De cette manière, les utilisateurs ne pourront pas télécharger votre application sur le Play Store s'ils se trouvent sur leur tablette par exemple.

Maintenant, d'un point de vue technique, nous allons utiliser l'API téléphonique qui est incarnée par la classe `TelephonyManager`. Les méthodes fournies dans cette classe permettent d'obtenir des informations sur le réseau et d'accéder à des informations sur l'abonné. Vous l'aurez remarqué, il s'agit encore une fois d'un `Manager` ; ainsi, pour l'obtenir, on doit en demander l'accès au `Context` :

```
TelephonyManager manager = Context.getSystemService(Context.
    TELEPHONY_SERVICE);
```

Obtenir des informations

Ensuite, si nous voulons obtenir des informations sur l'appareil, il faut demander la permission :

```
<uses-permission android:name="android.permission.
    READ_PHONE_STATE" />
```

Informations statiques

Tout d'abord, on peut déterminer le type du téléphone avec `int getPhoneType()`. Cette méthode peut retourner trois valeurs :

- `TelephonyManager.PHONE_TYPE_NONE` si l'appareil n'est pas un téléphone ou ne peut pas téléphoner.
- `TelephonyManager.PHONE_TYPE_GSM` si le téléphone exploite la norme de téléphonie mobile GSM. En France, ce sera le cas tout le temps.
- `TelephonyManager.PHONE_TYPE_CDMA` si le téléphone exploite la norme de téléphonie mobile CDMA. C'est une technologie vieillissante, mais encore très en vogue en Amérique du Nord.

Pour obtenir un identifiant unique de l'appareil, vous pouvez utiliser `String getDeviceId()`, et il est (parfois) possible d'obtenir le numéro de téléphone de l'utilisateur avec `String getLineNumber()`.

Informations dynamiques

Les informations précédentes étaient statiques, il y avait très peu de risques qu'elles évoluent pendant la durée de l'exécution de l'application. Cependant, il existe des données liées au réseau qui risquent de changer de manière régulière. Pour observer ces changements, on va passer par une interface dédiée : `PhoneStateListener`. On peut ensuite indiquer quels changements d'état on veut écouter avec le `TelephonyManager` en utilisant la méthode `void listen (PhoneStateListener listener, int events)` avec `events` des flags pour indiquer quels évènements on veut écouter. On note par exemple la présence des flags suivants :

- `PhoneStateListener.LISTEN_CALL_STATE` pour savoir que l'appareil déclenche ou reçoit un appel.
- `PhoneStateListener.LISTEN_DATA_CONNECTION_STATE` pour l'état de la connexion avec internet.
- `PhoneStateListener.LISTEN_DATA_ACTIVITY` pour l'état de l'échange des données avec internet.
- `PhoneStateListener.LISTEN_CELL_LOCATION` pour être notifié des déplacements de l'appareil.

```
1 TelephonyManager manager = (TelephonyManager) getSystemService(
      Context.TELEPHONY_SERVICE);
2 // Pour écouter les trois évènements
3 manager.listen(new PhoneStateListener(), PhoneStateListener.
      LISTEN_CALL_STATE | PhoneStateListener.
      LISTEN_DATA_CONNECTION_STATE | PhoneStateListener.
      LISTEN_DATA_ACTIVITY | PhoneStateListener.
      LISTEN_CELL_LOCATION);
```

Bien entendu, si vous voulez que l'appareil puisse écouter les déplacements, il vous faudra demander la permission avec `ACCESS_COARSE_LOCATION`, comme pour la localisation expliquée au chapitre précédent.

Ensuite, à chaque évènement correspond une méthode de *callback* à définir dans votre implémentation de `PhoneStateListener` :

```
1  protected String TAG = "TelephonyExample";
2
3  PhoneStateListener stateListener = new PhoneStateListener() {
4
5    // Appelée quand est déclenché l'évènement LISTEN_CALL_STATE
6    @Override
7    public void onCallStateChanged (int state, String
         incomingNumber) {
8      switch (state) {
9      case TelephonyManager.CALL_STATE_IDLE :
10       Log.d(TAG, "Pas d'appel en cours");
11       break;
12     case TelephonyManager.CALL_STATE_OFFHOOK :
```

```
13          Log.d(TAG, "Il y a une communication téléphonique en
              cours");
14          break;
15        case TelephonyManager.CALL_STATE_RINGING :
16          Log.d(TAG, "Le téléphone sonne, l'appelant est " +
              incomingNumber);
17          break;
18        default :
19          Log.d(TAG, "Etat inconnu");
20        }
21      }
22
23      // Appelée quand est déclenché l'évènement
          LISTEN_DATA_CONNECTION_STATE
24      @Override
25      public void onDataConnectionStateChanged (int state) {
26        switch (state) {
27        case TelephonyManager.DATA_CONNECTED :
28          Log.d(TAG, "L'appareil est connecté.");
29          break;
30        case TelephonyManager.DATA_CONNECTING :
31          Log.d(TAG, "L'appareil est en train de se connecter.");
32          break;
33        case TelephonyManager.DATA_DISCONNECTED :
34          Log.d(TAG, "L'appareil est déconnecté.");
35          break;
36        case TelephonyManager.DATA_SUSPENDED :
37          Log.d(TAG, "L'appareil est suspendu de manière temporaire
              .");
38          break;
39        }
40      }
41
42      // Appelée quand est déclenché l'évènement
          LISTEN_DATA_ACTIVITY
43      @Override
44      public void onDataActivity (int direction) {
45        switch (direction) {
46        case TelephonyManager.DATA_ACTIVITY_IN :
47          Log.d(TAG, "L'appareil est en train de télécharger des
              données.");
48          break;
49        case TelephonyManager.DATA_ACTIVITY_OUT :
50          Log.d(TAG, "L'appareil est en train d'envoyer des données
              .");
51          break;
52        case TelephonyManager.DATA_ACTIVITY_INOUT :
53          Log.d(TAG, "L'appareil est en train de télécharger ET d'
              envoyer des données.");
54          break;
```

```
55      case TelephonyManager.DATA_ACTIVITY_NONE :
56        Log.d(TAG, "L'appareil n'envoie pas de données et n'en té
            lécharge pas.");
57        break;
58      }
59    }
60
61    // Appelée quand est déclenché l'évènement
        LISTEN_SERVICE_STATE
62    @Override
63    public void onServiceStateChanged(ServiceState serviceState)
        {
64      // Est-ce que l'itinérance est activée ?
65      Log.d(TAG, "L'itinérance est activée : " + serviceState.
          getRoaming());
66      switch (serviceState.getState()) {
67      case ServiceState.STATE_IN_SERVICE :
68        Log.d(TAG, "Conditions normales d'appel");
69        // Pour obtenir un identifiant de l'opérateur
70        Log.d(TAG, "L'opérateur est " + serviceState.
            getOperatorAlphaLong());
71        break;
72      case ServiceState.STATE_EMERGENCY_ONLY :
73        Log.d(TAG, "Seuls les appels d'urgence sont autorisés.");
74        break;
75      case ServiceState.STATE_OUT_OF_SERVICE :
76        Log.d(TAG, "Ce téléphone n'est pas lié à un opérateur
            actuellement.");
77        break;
78      case ServiceState.STATE_POWER_OFF :
79        Log.d(TAG, "Le téléphone est en mode avion");
80        break;
81      default :
82        Log.d(TAG, "Etat inconnu");
83      }
84    }
85  };
```

Vous pouvez copier ce code grâce au code web suivant :

▷ Copier ce code
Code web : 679334

Téléphoner

Pour téléphoner, c'est très simple. En fait, vous savez déjà le faire. Vous l'avez peut-être même déjà fait. Il vous suffit de lancer un `Intent` qui a pour action `Intent.ACTION_CALL` et pour données `tel:numéro_de_téléphone` :

```
Intent appel = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:0102030405"));
startActivity(appel);
```

Cette action va lancer l'activité du combiné téléphonique pour que l'utilisateur puisse initier l'appel par lui-même. Ainsi, il n'y a pas besoin d'autorisation puisqu'au final l'utilisateur doit amorcer l'appel manuellement. Cependant, il se peut que vous souhaitiez que votre application lance l'appel directement, auquel cas vous devrez demander une permission particulière :

```
<uses-permission android:name="android.permission.CALL_PHONE" />
```

Envoyer et recevoir des SMS et MMS

L'envoi

Tout comme pour passer des appels, il existe deux manières de faire : soit avec l'application liée aux SMS, soit directement par l'application.

Prise en charge par une autre application

Pour transmettre un SMS à une application qui sera en charge de l'envoyer, il suffit d'utiliser un `Intent`. Il utilisera comme action `Intent.ACTION_SENDTO`, aura pour données un `smsto:numéro_de_téléphone` pour indiquer à qui sera envoyé le SMS, et enfin aura un extra de titre `sms_body` qui indiquera le contenu du message :

```
Intent sms = new Intent(Intent.ACTION_SENDTO, Uri.parse("smsto:0102030405");
sms.putExtra("sms_body", "Salut les Zéros !");
startActivity(sms);
```

Pour faire de même avec un MMS, c'est déjà plus compliqué. Déjà, les MMS ne fonctionnent pas avec `SENDTO` mais avec `SEND` tout court. De plus, le numéro de téléphone de destination devra être défini dans un extra qui s'appellera `address`. Enfin, le média associé au MMS sera ajouté à l'intent dans les données et dans un extra de nom `Intent.EXTRA_STREAM` à l'aide d'une URI qui pointe vers lui :

```
Uri image = Uri.fromFile("/sdcard/images/zozor.jpg");
Intent mms = new Intent(Intent.ACTION_SEND, image);
mms.putExtra(Intent.EXTRA_STREAM, image);
mms.setType("image/jpeg");

mms.putExtra("sms_body", "Salut les Zéros (mais avec une image !)");
mms.putExtra("address", "0102030405");

startActivity(mms);
```

Prise en charge directe

Tout d'abord, on a besoin de demander la permission :

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

Pour envoyer directement un SMS sans passer par une application externe, on utilise la classe `SmsManager`.

Ah ! J'imagine qu'on peut la récupérer en faisant `Context.getSystemService(Context.SMS_SERVICE)`, j'ai compris le truc maintenant !

Pour une fois, même si le nom de la classe se termine par « Manager », on va instancier un objet avec la méthode `static SmsManager SmsManager.getDefault()`. Pour envoyer un message, il suffit d'utiliser la méthode `void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)` :

- Il vous faut écrire le numéro du destinataire dans `destinationAddress`.
- Vous pouvez spécifier un centre de service d'adressage qui va gérer l'envoi du SMS dans `scAddress`. Si vous n'avez aucune idée de quoi mettre, un `null` sera suffisant.
- `text` contient le texte à envoyer.
- Il vous est possible d'insérer un `PendingIntent` dans `sentIntent` si vous souhaitez avoir des nouvelles de la transmission du message. Ce `PendingIntent` sera transmis à tout le système de façon à ce que vous puissiez le récupérer si vous le voulez. Le `PendingIntent` contiendra le code de résultat `Activity.RESULT_OK` si tout s'est bien passé. Enfin, vous pouvez aussi très bien mettre `null`.
- Encore une fois, vous pouvez mettre un `PendingIntent` dans `deliveryIntent` si vous souhaitez avoir des informations complémentaires sur la transmission.

On peut ainsi envisager un exemple simple :

```
SmsManager manager = SmsManager.getDefault();
manager.sendTextMessage("0102030405", null, "Salut les Zéros !
    ", null, null);
```

La taille maximum d'un SMS est de 160 caractères ! Vous pouvez cependant couper un message trop long avec `ArrayList<String> divideMessage(String text)`, puis vous pouvez envoyer les messages de façon à ce qu'ils soient liés les uns aux autres avec `void sendMultipartTextMessage(String destinationAddress, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntents, ArrayList <PendingIntent> deliveryIntents)`, les paramètres étant analogues à ceux de la méthode précédente.

Malheureusement, envoyer des MMS directement n'est pas aussi simple, c'est même tellement complexe que je ne l'aborderai pas !

Recevoir des SMS

On va faire ici quelque chose d'un peu étrange. Disons le carrément, on va s'enfoncer dans la quatrième dimension. En fait, recevoir des SMS n'est pas réellement prévu de manière officielle dans le SDK. C'est à vous de voir si vous voulez le faire.

La première chose à faire est de demander la permission dans le Manifest :

```
<uses-permission android:name="android.permission.RECEIVE_SMS"
    />
```

Ensuite, dès que le système reçoit un nouveau SMS, un broadcast intent est émis avec comme action `android.provider.Telephony.SMS_RECEIVED`. C'est donc à vous de développer un broadcast receiver qui gérera la réception du message. L'`Intent` qui enclenchera le `Receiver` contiendra un tableau d'objets qui s'appelle « pdus » dans les extras. Les PDU (Protocol Data Unit) sont les données qui sont transmises et qui représentent le message, ou les messages s'il a été divisé en plusieurs. Vous pourrez ensuite, à partir de ce tableau d'objets, créer un tableau de `SmsMessage` avec la méthode `static SmsMessage SmsMessage.createFromPdu(byte[] pdu)` :

```
// On récupère tous les extras
Bundle bundle = intent.getExtras();
if(bundle != null) {
  // Et on récupère le tableau d'objets qui s'appelle « pdus »
  Object[] pdus = (Object[]) bundle.get("pdus");

  // On crée un tableau de SmsMessage pour chaque message
  SmsMessage[] msg = new SmsMessage[pdus.length];

  // Puis, pour chaque tableau, on crée un message qu'on insère
      dans le tableau
  for(Object pdu : pdus)
    msg[i] = SmsMessage.createFromPdu((byte[]) pdu);
}
```

Vous pouvez ensuite récupérer des informations sur le message avec diverses méthodes : le contenu du message avec `String getMessageBody()`, le numéro de l'expéditeur avec `String getOriginatingAddress()` et le moment de l'envoi avec `long getTimestampMillis()`.

En résumé

- On peut obtenir beaucoup d'informations différentes sur le téléphone et le réseau de téléphonie auquel il est relié à l'aide de `TelephonyManager`. Il est même possible de surveiller l'état d'un téléphone avec `PhoneStateListener` de manière à pouvoir réagir rapidement à ses changements d'état.
- Envoyer des SMS se fait aussi assez facilement grâce à `SmsMessage` ; en revanche, envoyer des MMS est beaucoup plus complexe et demande un travail important.

Chapitre 26

Le multimédia

Difficulté :

Il y a une époque pas si lointaine où, quand on voulait écouter de la musique en faisant son jogging, il fallait avoir un lecteur dédié, un *walkman*. Et si on voulait regarder un film dans le train, il fallait un lecteur DVD portable. Heureusement, avec les progrès de la miniaturisation, il est maintenant possible de le faire n'importe où et n'importe quand, avec n'importe quel *smartphone*. Clairement, les appareils mobiles doivent désormais remplir de nouvelles fonctions, et il faut des applications pour assumer ces fonctions.

C'est pourquoi nous verrons ici comment lire des musiques ou des vidéos, qu'elles soient sur un support ou en *streaming*. Mais nous allons aussi voir comment effectuer des enregistrements audio et vidéo.

Le lecteur multimédia

Où trouver des fichiers multimédia ?

Il existe trois emplacements à partir desquels vous pourrez lire des fichiers multimédia :

1. Vous pouvez tout d'abord les insérer en tant que ressources dans votre projet, auquel cas il faut les mettre dans le répertoire `res/raw`. Vous pouvez aussi les insérer dans le répertoire `assets/` afin d'y accéder avec une URI de type `file://android_asset/nom_du_fichier.format_du_fichier`. Il s'agit de la solution la plus simple, mais aussi de la moins souple.
2. Vous pouvez stocker les fichiers sur l'appareil, par exemple sur le répertoire local de l'application en interne, auquel cas ils ne seront disponibles que pour cette application, ou alors sur un support externe (genre carte SD), auquel cas ils seront disponibles pour toutes les applications de l'appareil.
3. Il est aussi possible de lire des fichiers en *streaming* sur internet.

Formats des fichiers qui peuvent être lus

Tout d'abord, pour le streaming, on accepte le RTSP (Real Time Streaming Protocole), RTP (Real-Time Transport Protocole) et le streaming via HTTP.

Ensuite, je vais vous présenter tous les formats que connaît Android de base. En effet, il se peut que le constructeur de votre téléphone ait rajouté des capacités que je ne peux connaître. Ainsi, Android pourra toujours lire tous les fichiers présentés ci-dessous. Vous devriez comprendre toutes les colonnes de ce tableau, à l'exception peut-être de la colonne « **Encodeur** » : elle vous indique si oui ou non Android est capable de convertir un fichier vers ce format.

Audio

Format	Encodeur	Extension
AAC LC	oui	3GPP (.3gp), MPEG-4 (.mp4, .m4a)
HE-AACv1 (AAC+)		3GPP (.3gp), MPEG-4 (.mp4, .m4a)
HE-AACv2 (enhanced AAC+)		3GPP (.3gp), MPEG-4 (.mp4, .m4a)
AMR-NB	oui	3GPP (.3gp)
AMR-WB	oui	3GPP (.3gp)
MP3		MP3 (.mp3)
MIDI		Type 0 and 1 (.mid, .xmf, .mxmf), RTTTL/RTX (.rtttl, .rtx), OTA (.ota), iMelody (.imy)
Vorbis		Ogg (.ogg), Matroska (.mkv, Android 4.0+)
PCM/WAVE		WAVE (.wav)

Vidéo

Format	Encodeur	Extension
H.263	oui	3GPP (.3gp), MPEG-4 (.mp4)
H.264 AVC		3GPP (.3gp), MPEG-4 (.mp4)
MPEG-4 SP		3GPP (.3gp)

Le lecteur multimédia

Permissions

La première chose qu'on va faire, c'est penser aux permissions qu'il faut demander. Il n'y a pas de permission en particulier pour la lecture ou l'enregistrement ; en revanche, certaines fonctionnalités nécessitent quand même une autorisation. Par exemple, pour le streaming, il faut demander l'autorisation d'accéder à internet :

```
<uses-permission android:name="android.permission.INTERNET" />
```

De même, il est possible que vous vouliez faire en sorte que l'appareil ne se mette jamais en veille de façon à ce que l'utilisateur puisse continuer à regarder une vidéo qui dure longtemps sans être interrompu :

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

La lecture

La lecture de fichiers multimédia se fait avec la classe `MediaPlayer`. Sa vie peut être représentée par une machine à état, c'est-à-dire qu'elle traverse différents états et que la transition entre chaque état est symbolisée par des appels à des méthodes.

Comme pour une activité ?

Mais oui, exactement, vous avez tout compris !

Je pourrais très bien expliquer toutes les étapes et toutes les transitions, mais je doute que cela puisse vous être réellement utile, je ne ferais que vous embrouiller, je vais donc simplifier le processus. On va ainsi ne considérer que cinq états : **initialisé** quand on crée le lecteur, **préparé** quand on lui attribue un média, **démarré** tant que le média est joué, **en pause** quand la lecture est mise en pause ou **arrêté** quand elle est arrêtée, et enfin **terminé** quand la lecture est terminée.

Tout d'abord, pour créer un `MediaPlayer`, il existe un constructeur par défaut qui ne prend pas de paramètre. Un lecteur ainsi créé se trouve dans l'état **initialisé**. Vous pouvez ensuite lui indiquer un fichier à lire avec `void setDataSource(String path)` ou `void setDataSource(Context context, Uri uri)`. Il nous faut ensuite passer de

l'état **initialisé** à **préparé** (c'est-à-dire que le lecteur aura commencé à lire le fichier dans sa mémoire pour pouvoir commencer la lecture). Pour cela, on utilise une méthode qui s'appelle simplement `void prepare()`.

Cette méthode est synchrone, elle risque donc de bloquer le thread dans lequel elle se trouve. Ainsi, si vous appelez cette méthode dans le thread UI, vous risquez de le bloquer. En général, si vous essayez de lire dans un fichier cela devrait passer, mais pour un flux streaming il ne faut jamais faire cela. De manière générale, il faut appeler `prepare()` dans un thread différent du thread UI. Vous pouvez aussi appeler la méthode `void prepareAsync()`, qui est asynchrone et qui le fait de manière automatique pour vous.

Il est aussi possible de créer un lecteur multimédia directement préparé avec une méthode de type `create` :

```
1 // public static MediaPlayer create (Context context, int resid
    )
2 MediaPlayer media = MediaPlayer.create(getContext(), R.raw.file
    );
3 // public static MediaPlayer create (Context context, Uri uri)
4 media = MediaPlayer.create(getContext(), Uri.parse("file://
    android_asset/fichier.mp4");
5 media = MediaPlayer.create(getContext(), Uri.parse("file://
    sdcard/music/fichier.mp3");
6 media = MediaPlayer.create(getContext(), Uri.parse("http://www.
    site_trop_cool.com/musique.mp3");
7 media = MediaPlayer.create(getContext(), Uri.parse("rtsp://www.
    site_trop_cool.com/streaming.mov");
```

Maintenant que notre lecteur est en mode **préparé**, on veut passer en mode **démarré** qui symbolise la lecture du média ! Pour passer en mode **démarré**, on utilise la méthode `void start()`.

On peut ensuite passer à deux états différents :

- L'état **en pause**, en utilisant la méthode `void pause()`. On peut revenir à tout moment à l'état **démarré** avec la méthode `void resume()`.
- L'état **arrêté**, qui est enclenché en utilisant la méthode `void stop()`. À partir de cet état, on ne peut pas revenir directement à **démarré**. En effet, il faudra repasser à l'état **préparé**, puis indiquer qu'on veut retourner au début du média (avec la méthode `void seekTo(int msec)` qui permet de se balader dans le média).

```
1 player.stop();
2 player.prepare();
3 // On retourne au début du média, 0 est la première
    milliseconde
4 player.seekTo(0);
```

Enfin, une fois la lecture terminée, on passe à l'état **terminé**. À partir de là, on peut recommencer la lecture depuis le début avec `void start()`.

Enfin, n'oubliez pas de libérer la mémoire de votre lecteur multimédia avec la méthode `void release()`, on pourrait ainsi voir dans l'activité qui contient votre lecteur :

```
@Override
protected void onDestroy() {
  if(player != null) {
    player.release();
    player = null;
  }
}
```

Le volume et l'avancement

Pour changer le volume du lecteur, il suffit d'utiliser la méthode `void setVolume(float leftVolume, float rightVolume)` avec `leftVolume` un entier entre 0.0f (pour silencieux) et 1.0f (pour le volume maximum) du côté gauche, et `rightVolume` idem pour le côté droit. De base, si vous appuyez sur les boutons pour changer le volume, seul le volume de la sonnerie sera modifié. Si vous voulez que ce soit le volume du lecteur qui change et non celui de la sonnerie, indiquez-le avec `void setVolumeControlStream (AudioManager.STREAM_MUSIC)`.

Si vous voulez que l'écran ne s'éteigne pas quand vous lisez un média, utilisez `void setScreenOnWhilePlaying(boolean screenOn)`.

Enfin, si vous voulez que la lecture se fasse en boucle, c'est-à-dire qu'une fois arrivé à **terminé** on passe à **démarré**, utilisez `void setLooping(boolean looping)`.

La lecture de vidéos

Maintenant qu'on sait lire des fichiers audio, on va faire en sorte de pouvoir regarder des vidéos. Eh oui, parce qu'en plus du son, on aura besoin de la vidéo. Pour cela, on aura besoin d'une vue qui s'appelle `VideoView`. Elle ne prend pas d'attributs particuliers en XML :

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
  <VideoView android:id="@+id/videoView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
</LinearLayout>
```

Puis on va attribuer à ce `VideoView` un `MediaController`.

Mais qu'est-ce qu'un `MediaController` ? Nous n'en avons pas encore parlé ! Il s'agit en fait d'un layout qui permet de contrôler un média, aussi bien un son qu'une vidéo. Contrairement aux vues standards, on n'implémente pas un `MediaController` en XML

mais dans le code. Tout d'abord, on va le construire avec `public MediaController (Context context)`, puis on l'attribue au `VideoView` avec `void setMediaController (MediaController controller)` :

```
VideoView video = (VideoView) findViewById(R.id.videoView);
video.setMediaController(new MediaController(getContext()));
video.setVideoURI(Uri.parse("file://sdcard/video/example.avi"))
    ;
video.start();
```

Enregistrement

On aura besoin d'une permission pour enregistrer :

```
<uses-permission android:name="android.permission.RECORD_AUDIO"
    />
```

Il existe deux manières d'enregistrer.

Enregistrement sonore standard

Vous aurez besoin d'utiliser un `MediaRecorder` pour tous les enregistrements, dont les vidéos — mais nous le verrons plus tard. Ensuite c'est très simple, il suffit d'utiliser les méthodes suivantes :

- On indique quel est le matériel qui va enregistrer le son avec `void setAudioSource(int audio_source)`. Pour le micro, on lui donnera comme valeur `MediaRecorder.AudioSource.MIC`.
- Ensuite, vous pouvez choisir le format de sortie avec `void setOutputFormat(int output_format)`. De manière générale, on va mettre la valeur `MediaRecorder.OutputFormat.DEFAULT`, mais la valeur `MediaRecorder.OutputFormat.THREE_GPP` est aussi acceptable.
- Nous allons ensuite déclarer quelle méthode d'encodage audio nous voulons grâce à `void setAudioEncoder(int audio_encoder)`, qui prendra la plupart du temps `MediaRecorder.AudioEncoder.DEFAULT`.
- La prochaine chose à faire est de définir où sera enregistré le fichier avec `void setOutputFile(String path)`.
- Puis, comme pour le lecteur multimédia, on passe l'enregistreur en état **préparé** avec `void prepare()`.
- Enfin, on commence l'enregistrement avec `void start()`.

Pas facile à retenir, tout ça ! L'avantage ici, c'est que tout est automatique, alors vous n'avez « que » ces étapes à respecter.

```
MediaRecorder recorder = new MediaRecorder();
recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
recorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);
recorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
```

```
5 recorder.setOutputFile(PATH_NAME);
6 recorder.prepare();
7 recorder.start();
```

Une fois que vous avez décidé de finir l'enregistrement, il vous suffit d'appeler la méthode `void stop()`, puis de libérer la mémoire :

```
1 recorder.stop();
2 recorder.release();
3 recorder = null;
```

Enregistrer du son au format brut

L'avantage du son au format brut, c'est qu'il n'est pas traité et permet par conséquent certains traitements que la méthode précédente ne permettait pas. De cette manière, le son est de bien meilleure qualité. On va ici gérer un flux sonore, et non des fichiers. C'est très pratique dès qu'il faut effectuer des analyses du signal en temps réel.

Nous allons utiliser ici un *buffer*, c'est-à-dire un emplacement mémoire temporaire qui fait l'intermédiaire entre deux matériels ou processus différents. Ici, le buffer récupérera les données du flux sonore pour que nous puissions les utiliser dans notre code.

La classe à utiliser cette fois est `AudioRecord`, et on peut en construire une instance avec `public AudioRecord(int audioSource, int sampleRateInHz, int channelConfig, int audioFormat, int bufferSizeInBytes)` où :

- `audioSource` est la source d'enregistrement ; souvent on utilisera le micro `MediaRecorder.AudioSource.MIC`.
- Le taux d'échantillonnage est à indiquer dans `sampleRateInHz`, même si dans la pratique on ne met que 44100.
- Il faut mettre dans `channelConfig` la configuration des canaux audio ; s'il s'agit de mono, on utilise `AudioFormat.CHANNEL_IN_MONO` ; s'il s'agit de stéréo, on utilise `AudioFormat.CHANNEL_IN_STEREO`.
- On peut préciser le format avec `audioFormat`, mais en pratique on mettra toujours `AudioFormat.ENCODING_PCM_16BIT`.
- Enfin,on va mettre la taille totale du buffer dans `bufferSizeInBytes`. Si vous n'y comprenez rien, ce n'est pas grave, la méthode `static int AudioRecord.getMinBufferSize(int sampleRateInHz, int channelConfig, int audioFormat)` vous fournira une bonne valeur à utiliser.

Une utilisation typique pourrait être :

```
1 int sampleRateInHz = 44100;
2 int channelconfig = AudioFormat.CHANNEL_IN_STEREO;
3 int audioFormat = AudioFormat.ENCODING_PCM_16BIT;
4 int bufferSize = AudioRecord.getMinBufferSize(sampleRateInHz,
     channelconfig, audioFormat)
```

```
5 AudioRecord recorder = new AudioRecord(MediaRecorder.
      AudioSource.MIC, sampleRateInHz, channelconfig, audioFormat,
       bufferSize);
```

Chaque lecture que nous ferons dans `AudioRecord` prendra la taille du buffer, il nous faudra donc avoir un tableau qui fait la taille de ce buffer pour récupérer les données :

```
1 short[] buffer = new short[bufferSize];
```

Puis vous pouvez lire le flux en temps réel avec `int read(short[] audioData, int offsetInShorts, int sizeInShorts)` :

```
1 while(recorder.getRecordingState() == AudioRecord.
      RECORDSTATE_RECORDING) {
2   // Retourne le nombre de « shorts » lus, parce qu'il peut y
        en avoir moins que la taille du tableau
3   int nombreDeShorts = audioRecord.read(buffer, 0, bufferSize);
4 }
```

Enfin, il ne faut pas oublier de fermer le flux et de libérer la mémoire :

```
1 recorder.stop();
2 recorder.release();
3 recorder = null;
```

Prendre des photos

Demander à une autre application de le faire

La première chose que nous allons voir, c'est la solution de facilité : comment demander à une autre application de prendre des photos pour nous, puis ensuite les récupérer. On va bien entendu utiliser un intent, et son action sera `MediaStore.ACTION_IMAGE_CAPTURE`. Vous vous rappelez comment on lance une activité en lui demandant un résultat, j'espère ! Avec `void startActivityForResult(Intent intent, int requestCode)` où `requestCode` est un code qui permet d'identifier le retour. Le résultat sera ensuite disponible dans `void onActivityResult(int requestCode, int resultCode, Intent data)` avec `requestCode` qui vaut comme le `requestCode` que vous avez passé précédemment. On va ensuite préciser qu'on veut que l'image soit en extra dans le retour :

```
1 // L'endroit où sera enregistrée la photo
2 // Remarquez que mFichier est un attribut de ma classe
3 mFichier = new File(Environment.getExternalStorageDirectory(),
      "photo.jpg");
4 // On récupère ensuite l'URI associée au fichier
5 Uri fileUri = Uri.fromFile(mFichier);
6
7 // Maintenant, on crée l'intent
8 Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
```

```
9  // Et on déclare qu'on veut que l'image soit enregistrée là où
       pointe l'URI
10 intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);
11
12 // Enfin, on lance l'intent pour que l'application de photo se
       lance
13 startActivityForResult(intent, PHOTO_RESULT);
```

Il faut ensuite récupérer la photo dès que l'utilisateur revient dans l'application. On a ici un problème, parce que toutes les applications ne renverront pas le même résultat. Certaines renverront une image comme nous le voulons ; d'autres, juste une miniature… Nous allons donc voir ici comment gérer ces deux cas :

```
1  @Override
2  protected void onActivityResult(int requestCode, int resultCode
       , Intent data) {
3    // Si on revient de l'activité qu'on avait lancée avec le
         code PHOTO_RESULT
4    if (requestCode == PHOTO_RESULT && resultCode == RESULT_OK) {
5      // Si l'image est une miniature
6      if (data != null) {
7        if (data.hasExtra("data"))
8          Bitmap thumbnail = data.getParcelableExtra("data");
9      } else {
10       // On sait ici que le fichier pointé par mFichier est
            accessible, on peut donc faire ce qu'on veut avec, par
             exemple en faire un Bitmap
11       Bitmap image = BitmapFactory.decodeFile(mFichier);
12     }
13   }
14 }
```

Tout gérer nous-mêmes

La technique précédente peut dépanner par moments, mais ce n'est pas non plus la solution à tout. Il se peut qu'on veuille avoir le contrôle total sur notre caméra ! Pour cela, on aura besoin de la permission de l'utilisateur d'utiliser sa caméra :

```
1 <uses-permission android:name="android.permission.CAMERA" />
```

Vous pouvez ensuite manipuler très simplement la caméra avec la classe `Camera`. Pour récupérer une instance de cette classe, on utilise la méthode `static Camera Camera.open()`.

Il est ensuite possible de modifier les paramètres de l'appareil avec `void setParameters (Camera.Parameters params)`. Cependant, avant toute chose, il faut s'assurer que l'appareil peut supporter les paramètres qu'on va lui donner. En effet, chaque appareil aura un objectif photographique différent et par conséquent des caractéristiques différentes, alors il faudra faire en sorte de gérer le plus de cas possible. On va donc

récupérer les paramètres avec `Camera.Parameters getParameters()`, puis on pourra vérifier les modes supportés par l'appareil avec différentes méthodes, par exemple :

```
Camera camera = Camera.open();
Camera.Parameters params = camera.getParameters();

// Pour connaître les modes de flash supportés
List<String> flashs = params.getSupportedFlashModes();

// Pour connaître les tailles d'image supportées
List<Camera.Size> tailles = getSupportedPictureSizes();
```

Vous trouverez plus d'informations sur les modes supportés sur la page de `Camera.Parameters`. Une fois que vous connaissez les modes compatibles, vous pouvez manipuler la caméra à volonté :

```
camera.setFlashMode(Camera.Parameters.FLASH_MODE_AUTO);
camera.setPictureSize(1028, 768);
```

Ensuite, il existe deux méthodes pour prendre une photo :

```
void takePicture(Camera.ShutterCallback shutter, Camera.
    PictureCallback raw, Camera.PictureCallback jpeg);

void takePicture(Camera.ShutterCallback shutter, Camera.
    PictureCallback raw, Camera.PictureCallback postview, Camera
    .PictureCallback jpeg);
```

À noter que la seconde méthode, celle avec `postview`, ne sera accessible que si vous avez activé la prévisualisation.

On rencontre ici deux types de classes appelées en *callback* :

- `Camera.ShutterCallback` est utilisée pour indiquer le moment exact où la photo est prise. Elle ne contient qu'une méthode, `void onShutter()`.
- `Camera.PictureCallback` est utilisée une fois que l'image est prête. Elle contient la méthode `void onPictureTaken(byte[] data, Camera camera)` avec l'image contenue dans `data` et la `camera` avec laquelle la photo a été prise.

Ainsi, `shutter` est lancé dès que l'image est prise, mais avant qu'elle soit prête. `raw` correspond à l'instant où l'image est prête mais pas encore traitée pour correspondre aux paramètres que vous avez entrés. Encore après sera appelé `postview`, quand l'image sera redimensionnée comme vous l'avez demandé (ce n'est pas supporté par tous les appareils). Enfin, `jpeg` sera appelé dès que l'image finale sera prête. Vous pouvez passer `null` à tous les *callbacks* si vous n'en avez rien à faire :

```
private void takePicture(Camera camera) {
  // Jouera un son au moment où on prend une photo
  Camera.ShutterCallback shutterCallback = new Camera.
      ShutterCallback() {
```

```
    public void onShutter() {
      MediaPlayer media = MediaPlayer.create(getBaseContext(),
          R.raw.sonnerie);
      media.start();
      // Une fois la lecture terminée
      media.setOnCompletionListener(new MediaPlayer.
          OnCompletionListener() {
        public  void onCompletion(MediaPlayer mp) {
          // On libère le lecteur multimédia
          mp.release();
        }
      });
    }
  };

  // Sera lancée une fois l'image traitée, on enregistre l'
      image sur le support externe
  Camera.PictureCallback jpegCallback = new Camera.
      PictureCallback() {
    public void onPictureTaken(byte[] data, Camera camera) {
      FileOutputStream stream = null;
      try {
        String path = Environment.getExternalStorageDirectory()
            + "\\photo.jpg";
        stream = new FileOutputStream(path);
        stream.write(data);
      } catch (Exception e) {

      } finally {
        try { stream.close();} catch (Exception e) {}
      }
    }
  };

  camera.takePicture(shutterCallback, null, jpegCallback);
}
```

Enfin, on va voir comment permettre à l'utilisateur de prévisualiser ce qu'il va prendre en photo. Pour cela, on a besoin d'une vue particulière : `SurfaceView`. Il n'y a pas d'attributs particuliers à connaître pour la déclaration XML :

```
<SurfaceView
  android:id="@+id/surface_view"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" />
```

On aura ensuite besoin de récupérer le `SurfaceHolder` associé à notre `SurfaceView`, et ce avec la méthode `SurfaceHolder getHolder()`. On a ensuite besoin de lui attribuer un type, ce qui donne :

```
1 SurfaceView surface = (SurfaceView)findViewById(R.id.
      surfaceView);
2 SurfaceHolder holder = surface.getHolder();
3 holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
```

Ne vous inquiétez pas, c'est bientôt fini ! On n'a plus qu'à implémenter des méthodes de *callback* de manière à pouvoir gérer correctement le cycle de vie de la caméra et de la surface de prévisualisation. Pour cela, on utilise l'interface `SurfaceHolder.Callback` qui contient trois méthodes de *callback* qu'il est possible d'implémenter :

- `void surfaceChanged(SurfaceHolder holder, int format, int width, int height)` est lancée quand le `SurfaceView` change de dimensions.
- `void surfaceCreated(SurfaceHolder holder)` est appelée dès que la surface est créée. C'est dedans qu'on va associer la caméra au `SurfaceView`.
- À l'opposé, au moment de la destruction de la surface, la méthode `void surfaceDestroyed(SurfaceHolder holder)` sera exécutée. Elle permettra de dissocier la caméra et la surface.

Voici maintenant un exemple d'implémentation de cette synergie :

```
1  // Notre classe implémente SurfaceHolder.Callback
2  public class CameraActivity extends Activity implements
       SurfaceHolder.Callback {
3    private Camera mCamera = null;
4
5    @Override
6    public void onCreate(Bundle savedInstanceState) {
7      super.onCreate(savedInstanceState);
8      setContentView(R.layout.activity_main);
9
10     SurfaceView surface = (SurfaceView)findViewById(R.id.
           menu_settings);
11
12     SurfaceHolder holder = surface.getHolder();
13     holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
14
15     // On déclare que la classe actuelle gérera les callbacks
16     holder.addCallback(this);
17   }
18
19   // Se déclenche quand la surface est créée
20   public void surfaceCreated(SurfaceHolder holder) {
21     try {
22       mCamera.setPreviewDisplay(holder);
23       mCamera.startPreview();
24     } catch (IOException e) {
25       e.printStackTrace();
26     }
27   }
28
29   // Se déclenche quand la surface est détruite
```

```
30    public void surfaceDestroyed(SurfaceHolder holder) {
31      mCamera.stopPreview();
32    }
33
34    // Se déclenche quand la surface change de dimensions ou de
        format
35    public void surfaceChanged(SurfaceHolder holder, int format,
        int width, int height) {
36    }
37
38    @Override
39    protected void onResume() {
40      super.onResume();
41      mCamera = Camera.open();
42    }
43
44    @Override
45    protected void onPause() {
46      super.onPause();
47      mCamera.release();
48    }
49  }
```

Enfin, pour libérer la caméra, on utilise la méthode `void release()`.

Enregistrer des vidéos

Demander à une autre application de le faire à notre place

Encore une fois, il est tout à fait possible de demander à une autre application de prendre une vidéo pour nous, puis de la récupérer afin de la traiter. Cette fois, l'action à spécifier est `MediaStore.ACTION_VIDEO_CAPTURE`. Pour préciser dans quel emplacement stocker la vidéo, il faut utiliser l'extra `MediaStore.EXTRA_OUTPUT` :

```
1   private static final int VIDEO = 0;
2
3   @Override
4   public void onCreate(Bundle savedInstanceState) {
5     super.onCreate(savedInstanceState);
6     setContentView(R.layout.activity_main);
7
8     Uri emplacement = Uri.parse(new File(Environment.
        getExternalStorageDirectory() + "\\video\\nouvelle.3gp"));
9
10    Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
11    intent.putExtra(MediaStore.EXTRA_OUTPUT, emplacement);
12
13    startActivityForResult(intent, VIDEO);
14  }
```

```
@Override
protected void onActivityResult(int requestCode, int resultCode
    , Intent data) {
  if (requestCode == VIDEO) {
    if(resultCode == RESULT_OK) {
      Uri emplacement = data.getData();
    }
  }
}
```

Tout faire nous-mêmes

Tout d'abord, on a besoin de trois autorisations : une pour utiliser la caméra, une pour enregistrer le son et une pour enregistrer la vidéo :

```
<uses-permission android:name="android.permission.RECORD_AUDIO"
    />
<uses-permission android:name="android.permission.RECORD_VIDEO"
    />
<uses-permission android:name="android.permission.CAMERA" />
```

Au final, maintenant qu'on sait enregistrer du son, enregistrer de la vidéo n'est pas beaucoup plus complexe. En effet, on va encore utiliser `MediaRecorder`. Cependant, il faut avant débloquer la caméra pour qu'elle puisse être utilisée avec le `MediaRecorder`. Il suffit pour cela d'appeler sur votre caméra la méthode `void unlock()`. Vous pouvez maintenant associer votre `MediaRecorder` et votre `Camera` avec la méthode `void setCamera(Camera camera)`. Puis, comme pour l'enregistrement audio, il faut définir les sources :

```
camera.unlock();
mediaRecorder.setCamera(camera);
// Cette fois, on choisit un micro qui se trouve le plus proche
    possible de l'axe de la caméra
mediaRecorder.setAudioSource(MediaRecorder.AudioSource.
    CAMCORDER);
mediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
```

Cependant, quand on enregistre une vidéo, il est préférable de montrer à l'utilisateur ce qu'il est en train de filmer de manière à ce qu'il ne filme pas à l'aveugle. Comme nous l'avons déjà fait pour la prise de photographies, il est possible de donner un `SurfaceView` au `MediaRecorder`. La méthode à utiliser pour cela est `void setPreviewDisplay(SurfaceView surface)`. Encore une fois, vous pouvez implémenter les méthodes de *callback* contenues dans `SurfaceHolder.Callback`.

Enfin, comme pour l'enregistrement audio, on doit définir l'emplacement où enregistrer le fichier, préparer le lecteur, puis lancer l'enregistrement.

```
mediaRecorder.setOutputFile(PATH_NAME);
mediaRecorder.prepare();
```

```
mediaRecorder.start();
```

Toujours appeler setPreviewDisplay *avant* prepare, sinon vous aurez une erreur.

Enfin, il faut libérer la mémoire une fois la lecture terminée :

```
mediaRecorder.stop();
mediaRecorder.release();
mediaRecorder = null;
```

En résumé

- Android est capable de lire nativement beaucoup de formats de fichier différents, ce qui en fait un lecteur multimédia mobile idéal.
- Pour lire des fichiers multimédia, on peut utiliser un objet `MediaPlayer`. Il s'agit d'un objet qui se comporte comme une machine à états, il est donc assez délicat et lourd à manipuler ; cependant, il permet de lire des fichiers efficacement dès qu'on a appris à le maîtriser.
- Pour afficher des vidéos, on devra passer par une `VideoView`, qu'il est possible de lier à un `MediaPlayer` auquel on donnera des fichiers vidéo qu'il pourra lire nativement.
- L'enregistrement sonore est plus délicat, il faut réfléchir à l'avance à ce qu'on va faire en fonction de ce qu'on désire faire. Par exemple, `MediaRecorder` est en général utilisé, mais si on veut quelque chose de moins lourd, sur lequel on peut effectuer des traitements en temps réel, on utilisera plutôt `AudioRecord`.
- Il est possible de prendre une photo avec `Camera`. Il est possible de personnaliser à l'extrême son utilisation pour celui qui désire contrôler tous les aspects de la prise d'images.
- Pour prendre des vidéos, on utilisera aussi un `MediaRecorder`, mais on fera en sorte d'afficher une prévisualisation du résultat grâce à un `SurfaceView`.

Chapitre 27

Les capteurs

Difficulté :

La majorité des appareils modernes sont bien plus que de simples outils pour communiquer ou naviguer sur internet. Ils ont des capacités sensorielles, matérialisées par leurs capteurs. Ces capteurs nous fournissent des informations brutes avec une grande précision, qu'il est possible d'interpréter pour comprendre les transitions d'état que vit le terminal. On trouve par exemple des accéléromètres, des gyroscopes, des capteurs de champ magnétique, etc. Tous ces capteurs nous permettent d'explorer de nouvelles voies, d'offrir de nouvelles possibilités aux utilisateurs.

On va donc voir dans ce chapitre comment surveiller ces capteurs et comment les manipuler. On verra ainsi les informations que donnent les capteurs et comment en déduire ce que fait faire l'utilisateur à l'appareil.

Les différents capteurs

On peut répartir les capteurs en trois catégories :

- Les capteurs de mouvements : en mesurant les forces d'accélération et de rotation sur les trois axes, ces capteurs sont capables de déterminer dans quelle direction se dirige l'appareil. On y trouve l'accéléromètre, les capteurs de gravité, les gyroscopes et les capteurs de vecteurs de rotation.
- Les capteurs de position : évidemment, ils déterminent la position de l'appareil. On trouve ainsi les capteurs d'orientation et le magnétomètre.
- Les capteurs environnementaux : ce sont trois capteurs (baromètre, photomètre et thermomètre) qui mesurent la pression atmosphérique, l'illumination et la température ambiante.

D'un point de vue technique, on trouve deux types de capteurs. Certains sont des composants matériels, c'est-à-dire qu'il y a un composant physique présent sur le terminal. Ils fournissent des données en prenant des mesures. Certains autres capteurs sont uniquement présents d'une manière logicielle. Ils se basent sur des données fournies par des capteurs physiques pour calculer des données nouvelles.

Il n'est pas rare qu'un terminal n'ait pas tous les capteurs, mais seulement une sélection. Par exemple, la grande majorité des appareils ont un accéléromètre ou un magnétomètre, mais peu ont un thermomètre. De plus, il arrive qu'un terminal ait plusieurs exemplaires d'un capteur, mais calibrés d'une manière différente de façon à avoir des résultats différents.

Ces différents capteurs sont représentés par une valeur dans la classe `Sensor`. Regardez le tableau suivant.

Les lignes en italique correspondent aux valeurs qui existent dans l'API 7 mais qui ne sont pas utilisables avant l'API 9.

Opérations génériques

Demander la présence d'un capteur

Il se peut que votre application n'ait aucun sens sans un certain capteur. Si c'est un jeu qui exploite la détection de mouvements par exemple, vous feriez mieux d'interdire aux gens qui n'ont pas un accéléromètre de pouvoir télécharger votre application sur le Play Store. Pour indiquer qu'on ne veut pas qu'un utilisateur sans accéléromètre puisse télécharger votre application, il vous faudra ajouter une ligne de type `<uses-feature>` dans votre Manifest :

```
1 <uses-feature android:name="android.hardware.sensor.
     accelerometer"
2   android:required="true" />
```

Nom du capteur	**Valeur système**	**Type**	**Description**	**Utilisation typique**
Accéléromètre	`TYPE_ACCELEROMETER`	Matériel	Mesure la force d'accélération appliquée au terminal sur les trois axes (m/s^2).	Détecter les mouvements.
Tous les capteurs	`TYPE_ALL`	Matériel et logiciel	Représente tous les capteurs qui existent.	
Gyroscope	*`TYPE_GYROSCOPE`*	*Matériel*	*Mesure le taux de rotation sur chacun des trois axes en radian par seconde.*	*Détecter l'orientation de l'appareil.*
Photomètre	`TYPE_LIGHT`	Matériel	Mesure le niveau de lumière ambiante en lux.	Détecter la luminosité pour adapter celle de l'écran de l'appareil.
Magnétomètre	`TYPE_MAGNETIC_FIELD`	Matériel	Mesure le champ géomagnétique sur les trois axes en microtesla.	Créer un compas.
Orientation	`TYPE_ORIENTATION`	Logiciel	Mesure le degré de rotation que l'appareil effectue sur les trois axes.	Déterminer la position de l'appareil.
Baromètre	*`TYPE_PRESSURE`*	*Matériel*	*Mesure la pression ambiante en hectopascal ou millibar.*	*Surveiller les changements de pression de l'air ambiant.*
Capteur de proximité	`TYPE_PROXIMITY`	Matériel	Mesure la proximité d'un objet en centimètres.	Détecter si l'utilisateur porte le téléphone à son oreille pendant un appel.
Thermomètre	`TYPE_TEMPERATURE`	Matériel	Mesure la température de l'appareil en degrés Celsius.	Surveiller la température.

N'oubliez pas que `android:required="true"` sert à préciser que la présence de l'accéléromètre est absolument indispensable. S'il est possible d'utiliser votre application sans l'accéléromètre mais qu'il est fortement recommandé d'en posséder un, alors il vous suffit de mettre à la place `android:required="false"`.

Identifier les capteurs

La classe qui permet d'accéder aux capteurs est `SensorManager`. Pour en obtenir une instance, il suffit de faire :

```
1 SensorManager sensorManager = (SensorManager)getSystemService(
      Context.SENSOR_SERVICE);
```

Comme je l'ai déjà dit, les capteurs sont représentés par la classe `Sensor`. Si vous voulez connaître la liste de tous les capteurs existants sur l'appareil, il vous faudra utiliser la méthode `List<Sensor> getSensorList(int type)` avec `type` qui vaut `Sensor.TYPE_ALL`. De même, pour connaître tous les capteurs qui correspondent à une catégorie de capteurs, utilisez l'une des valeurs vues précédemment dans cette même méthode. Par exemple, pour connaître la liste de tous les magnétomètres :

```
1 ArrayList<Sensor> liste = sensorManager.getSensorList(Sensor.
      TYPE_MAGNETIC_FIELD);
```

Il est aussi possible d'obtenir une instance d'un capteur. Il suffit d'utiliser la méthode `Sensor getDefaultSensor(int type)` avec `type` un identifiant présenté dans le tableau précédent. Comme je vous l'ai déjà dit, il peut y avoir plusieurs capteurs qui ont le même objectif dans un appareil, c'est pourquoi cette méthode ne donnera que l'appareil par défaut, celui qui correspondra aux besoins les plus génériques.

Si le capteur demandé n'existe pas dans l'appareil, la méthode `getDefaultSensor` renverra `null`.

```
1 Sensor accelerometre = sensorManager.getDefaultSensor(Sensor.
      TYPE_ACCELETOMETER);
2 if(accelerometre != null)
3   // Il y a au moins un accéléromètre
4 else
5   // Il n'y en a pas
```

On peut ensuite récupérer des informations sur le capteur, comme sa consommation électrique avec `float getPower()` et sa portée avec `float getMaximumRange()`.

Vérifiez toujours qu'un capteur existe, même s'il est très populaire. Par exemple, il est peu probable qu'un accéléromètre soit absent, mais c'est possible !

Détection des changements des capteurs

L'interface `SensorEventListener` permet de détecter deux types de changement dans les capteurs :

- Un changement de précision du capteur avec la méthode de *callback* `void onAccuracyChanged(Sensor sensor, int accuracy)` avec `sensor` le capteur dont la précision a changé et `accuracy` la nouvelle précision. `accuracy` peut valoir `SensorManager.SENSOR_STATUS_ACCURACY_LOW` pour une faible précision, `SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM` pour une précision moyenne, `SensorManager.SENSOR_STATUS_ACCURACY_HIGH` pour une précision maximale et `SensorManager.SENSOR_STATUS_ACCURACY_UNRELIABLE` s'il ne faut pas faire confiance à ce capteur.
- Le capteur a calculé une nouvelle valeur, auquel cas se lancera la méthode de *callback* `void onSensorChanged(SensorEvent event)`. Un `SensorEvent` indique à chaque fois quatre informations contenues dans quatre attributs : l'attribut `accuracy` indique la précision de cette mesure (il peut avoir les mêmes valeurs que précédemment), l'attribut `sensor` contient une référence au capteur qui a fait la mesure, l'attribut `timestamp` est l'instant en nanosecondes où la valeur a été prise, et enfin les valeurs sont contenues dans l'attribut `values`.

`values` est un tableau d'entiers. Si dans le tableau précédent j'ai dit que les valeurs correspondaient aux trois axes, alors le tableau a trois valeurs : `values[0]` est la valeur sur l'axe x, `values[1]` la valeur sur l'axe y et `values[2]` la valeur sur l'axe z. Si le calcul ne se fait pas sur trois axes, alors il n'y aura que `values[0]` qui contiendra la valeur. Attention, cette méthode sera appelée très souvent, il est donc de votre devoir de ne pas effectuer d'opérations bloquantes à l'intérieur. Si vous effectuez des opérations longues à résoudre, alors il se peut que la méthode soit à nouveau lancée alors que l'ancienne exécution n'avait pas fini ses calculs, ce qui va encombrer le processeur au fur et à mesure.

```
1 final SensorEventListener mSensorEventListener = new
      SensorEventListener() {
2   public void onAccuracyChanged(Sensor sensor, int accuracy) {
3     // Que faire en cas de changement de précision ?
4   }
5
6   public void onSensorChanged(SensorEvent sensorEvent) {
7     // Que faire en cas d'évènements sur le capteur ?
8   }
9 };
```

Une fois notre interface écrite, il faut déclarer au capteur que nous sommes à son écoute. Pour cela, on va utiliser la méthode `boolean registerListener(SensorEventListener listener, Sensor sensor, int rate)` de `SensorManager`, avec le `listener`, le capteur dans `sensor` et la fréquence de mise à jour dans `rate`. Il est possible de donner à `rate` les valeurs suivantes, de la fréquence la moins élevée à la plus élevée :

- `SensorManager.SENSOR_DELAY_NORMAL` (0,2 seconde entre chaque prise) ;
- `SensorManager.SENSOR_DELAY_UI` (0,06 seconde entre chaque mise à jour, délai assez lent qui convient aux interfaces graphiques) ;
- `SensorManager.SENSOR_DELAY_GAME` (0,02 seconde entre chaque prise, convient aux jeux) ;
- `SensorManager.SENSOR_DELAY_FASTEST` (0 seconde entre les prises).

Le délai que vous indiquez n'est qu'une indication, il ne s'agit pas d'un délai très précis. Il se peut que la prise se fasse avant ou après le moment choisi. De manière générale, la meilleure pratique est d'avoir la valeur la plus lente possible, puisque c'est elle qui permet d'économiser le plus le processeur et donc la batterie.

Enfin, on peut désactiver l'écoute d'un capteur avec `void unregisterListener(SensorEventListener listener, Sensor sensor)`. N'oubliez pas de désactiver vos capteurs pendant que l'activité n'est pas au premier plan (donc il faut le désactiver pendant `onPause()` et le réactiver pendant `onResume()`), car le système ne le fera pas pour vous. De manière générale, désactivez les capteurs dès que vous ne les utilisez plus.

```
1  private SensorManager mSensorManager = null;
2  private Sensor mAccelerometer = null;
3
4  final SensorEventListener mSensorEventListener = new
       SensorEventListener() {
5    public void onAccuracyChanged(Sensor sensor, int accuracy) {
6      // Que faire en cas de changement de précision ?
7    }
8
9    public void onSensorChanged(SensorEvent sensorEvent) {
10     // Que faire en cas d'évènements sur le capteur ?
11   }
12 };
13
14 @Override
15 public final void onCreate(Bundle savedInstanceState) {
16   super.onCreate(savedInstanceState);
17   setContentView(R.layout.activity_main);
18
19   mSensorManager = (SensorManager) getSystemService(Context.
         SENSOR_SERVICE);
20   mAccelerometer = mSensorManager.getDefaultSensor(Sensor.
         TYPE_ACCELEROMETER);
21 }
22
23 @Override
24 protected void onResume() {
25   super.onResume();
26   mSensorManager.registerListener(mSensorEventListener,
         mAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
27 }
28
29 @Override
```

```
30 protected void onPause() {
31   super.onPause();
32   mSensorManager.unregisterListener(mSensorEventListener,
         mAccelerometer);
33 }
```

Les capteurs de mouvements

On va ici étudier les capteurs qui permettent de garder un œil sur les mouvements du terminal. Pour l'API 7, on trouve surtout l'accéléromètre, mais les versions suivantes supportent aussi le gyroscope, ainsi que trois capteurs logiciels (gravitationnel, d'accélération linéaire et de vecteurs de rotation). De nos jours, on trouve presque tout le temps un accéléromètre et un gyroscope. Pour les capteurs logiciels, c'est plus complexe puisqu'ils se basent souvent sur plusieurs capteurs pour déduire et calculer des données, ils nécessitent donc la présence de plusieurs capteurs différents.

On utilise les capteurs de mouvements pour détecter les... mouvements. Je pense en particulier aux inclinaisons, aux secousses, aux rotations ou aux balancements. Typiquement, les capteurs de mouvements ne sont pas utilisés pour détecter la position de l'utilisateur, mais si on les utilise conjointement avec d'autres capteurs, comme par exemple le magnétomètre, ils permettent de mieux évaluer ses déplacements.

Tous ces capteurs retournent un tableau de `float` de taille 3, chaque élément correspondant à un axe différent (voir figure 27.1). La première valeur, `values[0]`, se trouve sur l'axe x, il s'agit de l'axe de l'horizon, quand vous bougez votre téléphone de gauche à droite. Ainsi, la valeur est positive et augmente quand vous déplacez le téléphone vers la droite, alors qu'elle est négative et continue à diminuer plus vous le déplacez vers la gauche.

La deuxième valeur, `values[1]`, correspond à l'axe y, c'est-à-dire l'axe vertical, quand vous déplacez votre téléphone de haut en bas. La valeur est positive quand vous déplacez le téléphone vers le haut et négative quand vous le déplacez vers le bas.

Enfin, la troisième valeur, `values[2]`, correspond à l'axe z, il s'agit de l'axe sur lequel vous pouvez éloigner ou rapprocher le téléphone de vous. Quand vous le rapprochez de vous, la valeur est positive et, quand vous l'éloignez, la valeur est négative.

Vous l'aurez compris, toutes ces valeurs respectent un schéma identique : un 0 signifie pas de mouvement, une valeur positive un déplacement dans le sens de l'axe et une valeur négative un déplacement dans le sens inverse de celui de l'axe.

Enfin, cela va peut-être vous sembler logique, mais l'accéléromètre ne mesure pas du tout la vitesse, juste le changement de vitesse. Si vous voulez obtenir la vitesse depuis les données de l'accéléromètre, il vous faudra intégrer l'accélération sur le temps (que l'on peut obtenir avec l'attribut `timestamp`) pour obtenir la vitesse. Et pour obtenir une distance, il vous faudra intégrer la vitesse sur le temps.

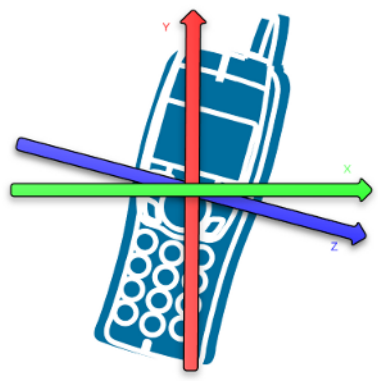


Figure 27.1 – Les différents axes

Les capteurs de position

On trouve trois (bon, en fait deux et un autre moins puissant) capteurs qui permettent de déterminer la position du terminal : le magnétomètre, le capteur d'orientation et le capteur de proximité (c'est le moins puissant, il est uniquement utilisé pour détecter quand l'utilisateur a le visage collé au téléphone, afin d'afficher le menu uniquement quand l'utilisateur n'a pas le téléphone contre la joue). Le magnétomètre et le capteur de proximité sont matériels, alors que le capteur d'orientation est une combinaison logicielle de l'accéléromètre et du magnétomètre.

Le capteur d'orientation et le magnétomètre renvoient un tableau de taille 3, alors que pour le capteur de proximité c'est plus compliqué. Parfois il renvoie une valeur en centimètres, parfois juste une valeur qui veut dire « proche » et une autre qui veut dire « loin » ; dans ces cas-là, un objet est considéré comme éloigné s'il se trouve à plus de 5 cm.

Cependant, le magnétomètre n'est pas utilisé que pour déterminer la position de l'appareil. Si on l'utilise conjointement avec l'accéléromètre, il est possible de détecter l'inclinaison de l'appareil. Et pour cela, la seule chose dont nous avons besoin, c'est de faire de gros calculs trigonométriques. Enfin... Android va (heureusement !) les faire pour nous.

Dans tous les cas, nous allons utiliser deux capteurs, il nous faudra donc déclarer les deux dans deux listeners différents. Une fois les données récupérées, il est possible de calculer ce qu'on appelle la méthode `Rotation` avec la méthode statique `static boolean SensorManager.getRotationMatrix(float[] R, float[] I, float[] gravity, float[] geomagnetic)` avec `R` le tableau de taille 9 dans lequel seront stockés les résultats, `I` un tableau d'inclinaison qui peut bien valoir `null`, `gravity` les données de l'accéléromètre et `geomagnetic` les données du magnétomètre.

La matrice rendue est une matrice de rotation. Depuis de celle-ci, vous pouvez obtenir l'orientation de l'appareil avec `static float[] SensorManager.getOrientation (float[] R, float[] values)` avec `R` la matrice de rotation et `values` le tableau de

taille 3 qui contiendra la valeur de la rotation pour chaque axe :

```
SensorManager sensorManager = (SensorManager)getSystemService(
    Context.SENSOR_SERVICE);
Sensor accelerometre = sm.getDefaultSensor(Sensor.
    TYPE_ACCELEROMETER);
Sensor magnetometre = sm.getDefaultSensor(Sensor.
    TYPE_MAGNETIC_FIELD);

sensorManager.registerListener(accelerometreListener,
    accelerometre, SensorManager.SENSOR_DELAY_UI);
sensorManager.registerListener(magnetometreListener,
    magnetometre, SensorManager.SENSOR_DELAY_UI);

// ...

float[] values = new float[3];
float[] R = new float[9];
SensorManager.getRotationMatrix(R, null, accelerometreValues,
    magnetometreValues);
SensorManager.getOrientation(R, values);

Log.d("Sensors", "Rotation sur l'axe z : " + values[0]);
Log.d("Sensors", "Rotation sur l'axe x : " + values[1]);
Log.d("Sensors", "Rotation sur l'axe y : " + values[2]);
```

Je vais vous expliquer maintenant à quoi correspondent ces chiffres, cependant avant toute chose, vous devez imaginez votre téléphone portable posé sur une table, le haut qui pointe vers vous, l'écran vers le sol. Ainsi, l'axe z pointe de bas en haut, l'axe x de droite à gauche et l'axe y de « loin devant vous » à « loin derrière vous » (en gros c'est l'axe qui vous traverse) :

- La rotation sur l'axe z est le mouvement que vous faites pour ouvrir ou fermer une bouteille de jus d'orange posée verticalement sur une table. C'est donc comme si vous englobiez le téléphone dans votre main et que vous le faisiez tourner sur l'écran. Il s'agit de l'angle entre le nord magnétique et l'angle de l'axe y. Il vaut 0 quand l'axe y pointe vers le nord magnétique, 180 si l'axe y pointe vers le sud, 90 quand il pointe vers l'est et 270 quand il pointe vers l'ouest.
- La rotation autour de l'axe x est le mouvement que vous faites quand vous ouvrez la bouteille de jus d'orange posée sur une table mais que le bouchon pointe vers votre droite. Enfin, ce n'est pas malin parce que vous allez tout renverser par terre. Elle vaut -90 quand vous tournez vers vous (ouvrir la bouteille) et 90 quand vous tournez dans l'autre sens (fermer la bouteille).
- La rotation sur l'axe y est le mouvement que vous faites quand vous ouvrez une bouteille de jus d'orange couchée sur la table alors que le bouchon pointe vers vous. Elle vaut 180 quand vous tournez vers la gauche (fermer la bouteille) et -180 quand vous tournez vers la droite (ouvrir la bouteille).

Enfin, il vous est possible de changer le système de coordonnées pour qu'il corresponde à vos besoins. C'est utile si votre application est censée être utilisée en mode paysage

plutôt qu'en mode portrait par exemple. On va utiliser la méthode `static boolean SensorManager.remapCoordinateSystem(float[] inR, int X, int Y, float[] outR)` avec `inR` la matrice de rotation à transformer (celle qu'on obtient avec `getRotation Matrix`), `X` désigne la nouvelle orientation de l'axe x, `Y` la nouvelle orientation de l'axe y et `outR` la nouvelle matrice de rotation (ne mettez pas `inR` dedans). Vous pouvez mettre dans `X` et `Y` des valeurs telles que `SensorManager.AXIS_X` qui représente l'axe x et `SensorManager.AXIS_MINUS_X` son orientation inverse. Vous trouverez de même les valeurs `SensorManager.AXIS_Y`, `SensorManager.AXIS_MINUS_Y`, `SensorManager.AXIS_Z` et `SensorManager.AXIS_MINUS_Z`.

Les capteurs environnementaux

Pour être franc, il n'y a pas tellement à dire sur les capteurs environnementaux. On en trouve trois dans l'API 7 : le baromètre, le photomètre et le thermomètre. Ce sont tous des capteurs matériels, mais il est bien possible qu'ils ne soient pas présents dans un appareil. Tout dépend du type d'appareil, pour être exact. Sur un téléphone et sur une tablette, on en trouve rarement (à l'exception du photomètre qui permet de détecter automatiquement la meilleure luminosité pour l'écran), mais sur une station météo ils sont souvent présents. Il faut donc redoubler de prudence quand vous essayez de les utiliser, vérifiez à l'avance leur présence.

Tous ces capteurs rendent des valeurs uniques, pas de tableaux à plusieurs dimensions.

En résumé

- La majorité des appareils sous Android utilisent des capteurs pour créer un lien supplémentaire entre l'utilisateur et son appareil. On rencontre trois types de capteurs : les capteurs de mouvements, les capteurs de position et les capteurs environnementaux.
- La présence d'un capteur dépend énormément des appareils, il faut donc faire attention à bien déclarer son utilisation dans le Manifest et à vérifier sa présence au moment de l'utilisation.
- Les capteurs de mouvements permettent de détecter les déplacements que l'utilisateur fait faire à son appareil. Il arrive qu'ils soient influencés par des facteurs extérieurs comme la gravité, il faut donc réfléchir à des solutions basées sur des calculs physiques quand on désire avoir une valeur précise d'une mesure.
- Les capteurs de position sont capables de repérer la position de l'appareil par rapport à un référentiel. Par exemple, le capteur de proximité peut donner la distance entre l'utilisateur et l'appareil. En pratique, le magnétomètre est surtout utilisé conjointement avec des capteurs de mouvements pour détecter d'autres types de mouvements, comme les rotations.
- Les capteurs environnementaux sont vraiment beaucoup plus rares sur un téléphone ou une tablette, mais il existe d'autres terminaux spécifiques, comme des stations météorologiques, qui sont bardés de ce type de capteurs.

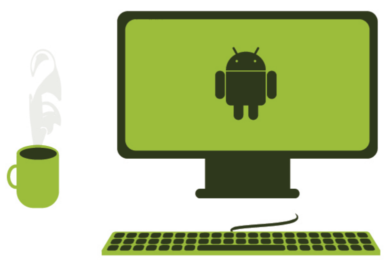
Chapitre 28

TP : un labyrinthe

Difficulté :

Nous voici arrivés au dernier TP de ce cours ! Et comme beaucoup de personnes m'ont demandé comment faire un jeu, je vais vous indiquer ici quelques pistes de réflexion en créant un jeu relativement simple : un labyrinthe. Et en dépit de l'apparente simplicité de ce jeu, vous verrez qu'il faut penser à beaucoup de choses pour que le jeu reste amusant et cohérent.

Nous nous baserons ici uniquement sur les API que nous connaissons déjà. Ainsi, ce TP n'aborde pas Open GL par exemple, dont la maîtrise va bien au-delà de l'objectif de ce cours ! Mais vous verrez qu'avec un brin d'astuce il est déjà possible de faire beaucoup avec ce que nous avons à portée de main.



Objectifs

Vous l'aurez compris, nous allons faire un labyrinthe. Le principe du jeu est très simple : le joueur utilise l'accéléromètre de son téléphone pour diriger une boule. Ainsi, quand il penche l'appareil vers le bas, la boule se déplace vers le bas. Quand il penche l'appareil vers le haut, la boule se dirige vers le haut, de même pour la gauche et la droite. L'objectif est de pouvoir placer la boule à un emplacement particulier qui symbolisera la sortie. Cependant, le parcours sera semé d'embûches ! Il faudra en effet faire en sorte de zigzaguer entre des trous situés dans le sol, placés par les immondes Zörglubienotchs qui n'ont qu'un seul objectif : détruire le monde (Ha ! Ha ! Ha ! Ha !).

Le scénario est optionnel.

La figure 28.1 est un aperçu du résultat final que j'obtiens.

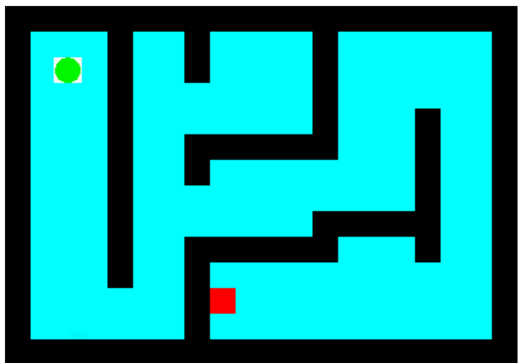


FIGURE 28.1 – Le labyrinthe

On peut y voir les différents éléments qui composent le jeu :

- La boule verte, le seul élément qui bouge quand vous bougez votre téléphone.
- Une case blanche, qui indique le départ du labyrinthe.
- Une case rouge, qui indique l'objectif à atteindre pour détruire le roi des Zörglubienotchs.
- Plein de cases noires : ce sont les pièges posés par les Zörglubienotchs et qui détruisent votre boule.

Quand l'utilisateur perd, une boîte de dialogue le signale et le jeu se met en pause. Quand l'utilisateur gagne, une autre boîte de dialogue le signale et le jeu se met en pause, c'est aussi simple que cela !

Avant de vous laisser vous aventurer seuls, laissez-moi vous donner quelques indications qui pourraient vous être précieuses.

Spécifications techniques

Organisation du code

De manière générale, quand on développe un jeu, on doit penser à trois moteurs qui permettront de gérer les différentes composantes qui constituent le jeu :

- Le moteur graphique qui s'occupera de dessiner.
- Le moteur physique qui s'occupera de gérer les positions, déplacements et interactions entre les éléments.
- Le moteur multimédia qui joue les animations et les sons au bon moment.

Nous n'utiliserons que deux de ces moteurs : le moteur graphique et le moteur physique. Cette organisation implique une chose : il y aura deux représentations pour chaque élément. Par exemple, une représentation graphique de la boule — celle que connaîtra le moteur graphique — et une représentation physique — celle que connaîtra le moteur physique. On peut ainsi dire que la boule sera divisée en deux parties distinctes, qu'il faudra lier pour avoir un ensemble cohérent.

La toute première chose à laquelle il faut penser, c'est qu'on va donner du matériel à ces moteurs. Le moteur graphique ne peut dessiner s'il n'a rien à dessiner, le moteur physique ne peut calculer de déplacements s'il n'y a pas quelque chose qui bouge ! On va ainsi définir des modèles qui vont contenir les différentes informations sur les constituants.

Les modèles

Comme je viens de le dire, un modèle sera une classe Java qui contiendra des informations sur les constituants du jeu. Ces informations dépendront bien entendu de l'objet représenté. Réfléchissons maintenant à ce qui constitue notre jeu. Nous avons déjà une boule. Ensuite, nous avons des trous dans lesquels peut tomber la boule, une case de départ et une case d'arrivée. Ces trois types d'objets ne bougent pas, et se dessinent toujours un peu de la même manière ! On peut alors décréter qu'ils sont assez similaires quand même. Voyons maintenant ce que doivent contenir les modèles.

La boule

Il s'agit du cœur du jeu, de l'élément le plus compliqué à gérer. Tout d'abord, il va se déplacer, il nous faut donc connaître sa position. Le `Canvas` du `SurfaceView` se comporte comme n'importe quel autre `Canvas` que nous avons vu, c'est-à-dire qu'il possède un axe x qui va de gauche à droite (le rebord gauche vaut 0 et le rebord droit vaut la taille de l'écran en largeur). Il possède aussi un axe y qui va de haut en bas (le plafond du téléphone vaut 0 et le plancher vaut la taille de l'écran en hauteur). Vous aurez donc besoin de deux attributs pour situer votre boule sur le `Canvas` : un pour l'axe x, un pour l'axe y.

En plus de la position, il faut penser à la vitesse. Eh oui, plus la boule roule, plus elle

accélère ! Comme notre boule se déplace sur deux axes (x et y), on aura besoin de deux indicateurs de vitesse : un pour l'axe x, et un pour l'axe y. Alors, accélérer, c'est bien, mais si notre boule dépasse la vitesse du son, c'est moins pratique pour jouer quand même. Il nous faudra alors aussi un attribut qui indiquera la vitesse à ne pas dépasser.

Pour le dessin, nous aurons aussi besoin d'indiquer la taille de la boule ainsi que sa couleur. De cette manière, on a pensé à tout, on obtient alors cette classe :

```
1  public class Boule {
2    // Je garde le rayon dans une constante au cas où j'aurais
       besoin d'y accéder depuis une autre classe
3    public static final int RAYON = 10;
4
5    // Ma boule sera verte
6    private int mCouleur = Color.GREEN;
7
8    // Je n'initialise pas ma position puisque je l'ignore au dé
       marrage
9    private float mX;
10   private float mY;
11
12   // La vitesse est nulle au début du jeu
13   private float mSpeedX = 0;
14   private float mSpeedY = 0;
15
16   // Après quelques tests, pour moi, la vitesse maximale
       optimale est 20
17   private static final float MAX_SPEED = 20.0f;
```

Les blocs

Même s'ils ont un comportement physique similaire, les blocs ont tous un dessin et un objectif différent. Il nous faut ainsi un moyen de les différencier, en dépit du fait qu'ils soient tous des objets de la classe `Bloc`. Alors comment faire ? Il existe deux solutions :

- Soit on crée des classes qui dérivent de `Bloc` pour chaque type de bloc, auquel cas on pourra tester si un objet appartient à une classe particulière avec l'instruction `instanceof`. Par exemple, `bloc instanceof sdz.chapitreCinq.labyrinthe.Trou`.
- Ou alors on ajoute un attribut `type` à la classe `Bloc`, qui contiendra le type de notre bloc. Tous les types possibles seront alors décrits dans une énumération.

J'ai privilégié la seconde méthode, tout simplement parce qu'elle impliquait d'utiliser les énumérations, ce qui en fait un exemple pédagogiquement plus intéressant.

C'est quoi une énumération ?

Avec la programmation orientée objet, on utilise plus rarement les énumérations, et pourtant elles sont pratiques ! Une énumération, c'est une façon de décrire une liste de constantes. Il existe trois types de blocs (trou, départ, arrivée), on aura donc trois types de constantes dans notre énumération :

```
enum  Type { TROU, DEPART, ARRIVEE };
```

Comme vous pouvez le voir, on n'a pas besoin d'ajouter une valeur à nos constantes ; en effet, leur nom fera office de valeur.

Autre chose : comme il faut placer les blocs, nous avons encore une fois besoin des coordonnées du bloc. De plus, il est nécessaire de définir la taille d'un bloc. De ce fait, on obtient :

```
public class Bloc {
  enum  Type { TROU, DEBUT, FIN };

  private float SIZE = Boule.RAYON * 2;

  private float mX;
  private float mY;

  private Type mType = null;
```

Comme vous pouvez le voir, j'ai fait en sorte qu'un bloc ait deux fois la taille de la boule.

Le moteur graphique

Très simple à comprendre, il sera en charge de dessiner les composants de notre scène de jeu. Ce à quoi il faut faire attention ici, c'est que certains éléments se déplacent (je pense en particulier à la boule). Il faut ainsi faire en sorte que le dessin corresponde toujours à la position exacte de l'élément : il ne faut pas que la boule se trouve à un emplacement et que le dessin affiche toujours son ancien emplacement. Regardez la figure 28.2.

FIGURE 28.2 – À gauche le dessin de la boule, à droite sa représentation physique

Maintenant, regardez la figure 28.3.

À gauche, les deux représentations se superposent : la boule ne bouge pas, alors, au moment de dessiner la boule, il suffit de la dessiner au même endroit que précédemment. Cependant, à l'instant suivant (à droite), le joueur penche l'appareil, et la boule se met à se déplacer. On peut voir que la représentation graphique est restée au même endroit alors que la représentation physique a bougé, et donc ce que le joueur voit n'est pas ce que le jeu sait de l'emplacement de la boule. C'est ce que je veux dire par « il faut faire

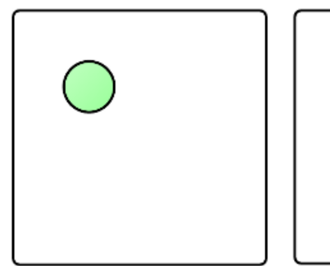
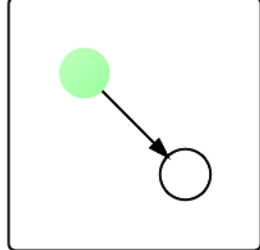
 

FIGURE 28.3 – Représentation des deux moteurs au temps T à gauche, T+1 à droite

en sorte que le dessin corresponde toujours à la position exacte de l'élément ». Ainsi, à chaque fois que vous voulez dessiner la boule, il faudra le faire avec sa position exacte.

Pour effectuer les dessins, on va utiliser un `SurfaceView`, puisqu'il s'agit de la manière la plus facile de dessiner avec de bonnes performances. Ensuite, chaque élément devra être dessiné sur le `Canvas` du `SurfaceView`. Par exemple, chez moi, la boule est un disque de rayon 10 et de couleur verte.

Pour vous faciliter la vie, je vous propose de récupérer tout simplement le *framework* que nous avons écrit dans le chapitre sur le dessin, puisqu'il convient parfaitement à ce projet. Il ne vous reste plus ensuite qu'à dessiner dans la méthode de *callback* `void onDraw(Canvas canvas)`.

Pour adapter le dessin à tous les périphériques, vos éléments doivent être proportionnels à la taille de l'écran. Je pense au moins aux différents blocs qui doivent rentrer dans tous les écrans, même les plus petits.

Le moteur physique

Plus délicat à gérer que le moteur graphique, le moteur physique gère la position, les déplacements et l'interaction entre les différents éléments de votre jeu. De plus, dans notre cas particulier, il faudra aussi manipuler l'accéléromètre ! Vous savez déjà le faire normalement, alors pas de soucis ! Cependant, qu'allons-nous faire des données fournies par le capteur ? Eh bien, nous n'avons besoin que de deux données : les deux axes. J'ai choisi de faire en sorte que la position de base soit le téléphone posé à plat sur une table. Quand l'utilisateur penche le téléphone vers lui, la boule « tombe », comme si elle était attirée par la gravité. Si l'utilisateur penche l'appareil dans l'autre sens quand la boule « tombe », alors elle remonte une pente, elle a du mal à « monter » et elle se met à rouler dans le sens de la pente, comme le ferait une vraie boule. De ce fait, j'ai conservé les données sur deux axes seulement : x et y.

Ces données servent à modifier la vitesse de la boule. Si la boule roule dans le sens de la pente, elle prend de la vitesse et donc sa vitesse augmente avec la valeur du capteur. Si la vitesse dépasse la vitesse maximale, alors on impose la vitesse maximale comme vitesse de la boule. Enfin, si la vitesse est négative... cela veut tout simplement dire

que la boule se dirige vers la gauche ou le haut, c'est normal !

```
SensorEventListener mSensorEventListener = new
    SensorEventListener() {
  @Override
  public void onSensorChanged(SensorEvent event) {
    // La valeur sur l'axe x
    float x = event.values[0];
    // La valeur sur l'axe y
    float y = event.values[1];

    // On accélère ou décélère en fonction des valeurs données
    boule.xSpeed = boule.xSpeed + x;
    // On vérifie qu'on ne dépasse pas la vitesse maximale
    if(boule.xSpeed > Boule.MAX_SPEED)
      boule.xSpeed = Boule.MAX_SPEED;
    if(boule.xSpeed < Boule.MAX_SPEED)
      boule.xSpeed = -Boule.MAX_SPEED;

    boule.ySpeed = boule.ySpeed + y;
    if(boule.ySpeed > Boule.MAX_SPEED)
      boule.ySpeed = Boule.MAX_SPEED;
    if(boule.ySpeed < Boule.MAX_SPEED)
      boule.ySpeed = -Boule.MAX_SPEED;

    // Puis on modifie les coordonnées en fonction de la
        vitesse
    boule.x += xSpeed;
    boule.y += ySpeed;
  }

  @Override
  public void onAccuracyChanged(Sensor sensor, int accuracy) {

  }
}
```

Maintenant que notre boule bouge, que faire quand elle rencontre un bloc ? Comment détecter cette rencontre ? Le plus simple est encore d'utiliser des objets de type `RectF`, tout simplement parce qu'ils possèdent une méthode qui permet de détecter si deux `RectF` entrent en collision. Cette méthode est `boolean intersect(RectF r)` : le `boolean` retourné vaudra `true` si les deux rectangles entrent bien en collision et `r` sera remplacé par le rectangle formé par la collision.

Je le répète, le rectangle passé en attribut sera modifié par cette méthode, il vous faut donc faire une copie du rectangle dont vous souhaitez vérifier la collision, sinon il sera modifié. Pour copier un `RectF`, utilisez le constructeur `public RectF(RectF r)`.

Ainsi, on va rajouter un rectangle à nos blocs et à notre boule. C'est très simple, il vous suffit de deux données : les coordonnées du point en haut à gauche (sur l'axe x et l'axe y), puis la taille du rectangle. Avec ces données, on peut très bien construire un rectangle, voyez vous-mêmes :

```
1 public RectF (float left, float top, float right, float bottom)
```

En fait, l'attribut `left` correspond à la coordonnée sur l'axe x du côté gauche du rectangle, `top` à la coordonnée sur l'axe y du plafond, `right` à la coordonnée sur l'axe y du côté droit et `bottom` à la coordonnée sur l'axe y du plancher. De ce fait, avec les données que je vous ai demandées, il suffit de faire :

```
1 public RectF (float coordonnee_x, float coordonnee_y, float
      coordonnee_x + taille_du_rectangle, float coordonnee_y +
      taille_du_rectangle)
```

Mais comment faire pour la boule ? C'est un disque, pas un rectangle !

Cela peut sembler bizarre, mais on n'a nullement besoin d'une représentation exacte de la boule, on peut accompagner sa représentation d'un rectangle, tout simplement parce que la majorité des collisions ne peuvent pas se faire en diagonale, uniquement sur les rebords extrêmes de la boule, comme schématisé à la figure 28.4.

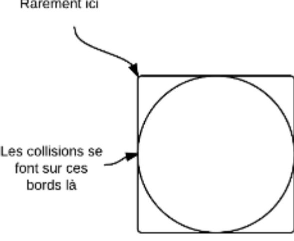


Figure 28.4 – Emplacement des collisions

Bien sûr, les collisions qui se feront sur les diagonales ne seront pas précises, mais franchement elles sont tellement rares et ce serait tellement complexe de les gérer qu'on va simplement les laisser tomber. De ce fait, il faut ajouter un `RectF` dans les attributs de la boule et, à chaque fois qu'elle bouge, il faut mettre à jour les coordonnées du rectangle pour qu'il englobe bien la boule et puisse ainsi détecter les collisions.

Le labyrinthe

C'est très simple, pour cette version simplifiée, le labyrinthe sera tout simplement une liste de blocs qui est générée au lancement de l'application. Chez moi, j'ai utilisé le labyrinthe suivant :

```
List<Bloc> Blocs = new ArrayList<Bloc>();
Blocs.add(new Bloc(Type.TROU, 0, 0));
Blocs.add(new Bloc(Type.TROU, 0, 1));
Blocs.add(new Bloc(Type.TROU, 0, 2));
Blocs.add(new Bloc(Type.TROU, 0, 3));
Blocs.add(new Bloc(Type.TROU, 0, 4));
Blocs.add(new Bloc(Type.TROU, 0, 5));
Blocs.add(new Bloc(Type.TROU, 0, 6));
Blocs.add(new Bloc(Type.TROU, 0, 7));
Blocs.add(new Bloc(Type.TROU, 0, 8));
Blocs.add(new Bloc(Type.TROU, 0, 9));
Blocs.add(new Bloc(Type.TROU, 0, 10));
Blocs.add(new Bloc(Type.TROU, 0, 11));
Blocs.add(new Bloc(Type.TROU, 0, 12));
Blocs.add(new Bloc(Type.TROU, 0, 13));

Blocs.add(new Bloc(Type.TROU, 1, 0));
Blocs.add(new Bloc(Type.TROU, 1, 13));

Blocs.add(new Bloc(Type.TROU, 2, 0));
Blocs.add(new Bloc(Type.TROU, 2, 13));

Blocs.add(new Bloc(Type.TROU, 3, 0));
Blocs.add(new Bloc(Type.TROU, 3, 13));

Blocs.add(new Bloc(Type.TROU, 4, 0));
Blocs.add(new Bloc(Type.TROU, 4, 1));
Blocs.add(new Bloc(Type.TROU, 4, 2));
Blocs.add(new Bloc(Type.TROU, 4, 3));
Blocs.add(new Bloc(Type.TROU, 4, 4));
Blocs.add(new Bloc(Type.TROU, 4, 5));
Blocs.add(new Bloc(Type.TROU, 4, 6));
Blocs.add(new Bloc(Type.TROU, 4, 7));
Blocs.add(new Bloc(Type.TROU, 4, 8));
Blocs.add(new Bloc(Type.TROU, 4, 9));
Blocs.add(new Bloc(Type.TROU, 4, 10));
Blocs.add(new Bloc(Type.TROU, 4, 13));

Blocs.add(new Bloc(Type.TROU, 5, 0));
Blocs.add(new Bloc(Type.TROU, 5, 13));

Blocs.add(new Bloc(Type.TROU, 6, 0));
Blocs.add(new Bloc(Type.TROU, 6, 13));

Blocs.add(new Bloc(Type.TROU, 7, 0));
Blocs.add(new Bloc(Type.TROU, 7, 1));
Blocs.add(new Bloc(Type.TROU, 7, 2));
Blocs.add(new Bloc(Type.TROU, 7, 5));
Blocs.add(new Bloc(Type.TROU, 7, 6));
Blocs.add(new Bloc(Type.TROU, 7, 9));
```

```
51  Blocs.add(new Bloc(Type.TROU, 7, 10));
52  Blocs.add(new Bloc(Type.TROU, 7, 11));
53  Blocs.add(new Bloc(Type.TROU, 7, 12));
54  Blocs.add(new Bloc(Type.TROU, 7, 13));
55
56  Blocs.add(new Bloc(Type.TROU, 8, 0));
57  Blocs.add(new Bloc(Type.TROU, 8, 5));
58  Blocs.add(new Bloc(Type.TROU, 8, 9));
59  Blocs.add(new Bloc(Type.TROU, 8, 13));
60
61  Blocs.add(new Bloc(Type.TROU, 9, 0));
62  Blocs.add(new Bloc(Type.TROU, 9, 5));
63  Blocs.add(new Bloc(Type.TROU, 9, 9));
64  Blocs.add(new Bloc(Type.TROU, 9, 13));
65
66  Blocs.add(new Bloc(Type.TROU, 10, 0));
67  Blocs.add(new Bloc(Type.TROU, 10, 5));
68  Blocs.add(new Bloc(Type.TROU, 10, 9));
69  Blocs.add(new Bloc(Type.TROU, 10, 13));
70
71  Blocs.add(new Bloc(Type.TROU, 11, 0));
72  Blocs.add(new Bloc(Type.TROU, 11, 5));
73  Blocs.add(new Bloc(Type.TROU, 11, 9));
74  Blocs.add(new Bloc(Type.TROU, 11, 13));
75
76  Blocs.add(new Bloc(Type.TROU, 12, 0));
77  Blocs.add(new Bloc(Type.TROU, 12, 1));
78  Blocs.add(new Bloc(Type.TROU, 12, 2));
79  Blocs.add(new Bloc(Type.TROU, 12, 3));
80  Blocs.add(new Bloc(Type.TROU, 12, 4));
81  Blocs.add(new Bloc(Type.TROU, 12, 5));
82  Blocs.add(new Bloc(Type.TROU, 12, 8));
83  Blocs.add(new Bloc(Type.TROU, 12, 9));
84  Blocs.add(new Bloc(Type.TROU, 12, 13));
85
86  Blocs.add(new Bloc(Type.TROU, 13, 0));
87  Blocs.add(new Bloc(Type.TROU, 13, 8));
88  Blocs.add(new Bloc(Type.TROU, 13, 13));
89
90  Blocs.add(new Bloc(Type.TROU, 14, 0));
91  Blocs.add(new Bloc(Type.TROU, 14, 8));
92  Blocs.add(new Bloc(Type.TROU, 14, 13));
93
94  Blocs.add(new Bloc(Type.TROU, 15, 0));
95  Blocs.add(new Bloc(Type.TROU, 15, 8));
96  Blocs.add(new Bloc(Type.TROU, 15, 13));
97
98  Blocs.add(new Bloc(Type.TROU, 16, 0));
99  Blocs.add(new Bloc(Type.TROU, 16, 4));
100 Blocs.add(new Bloc(Type.TROU, 16, 5));
```

```
101 Blocs.add(new Bloc(Type.TROU, 16, 6));
102 Blocs.add(new Bloc(Type.TROU, 16, 7));
103 Blocs.add(new Bloc(Type.TROU, 16, 8));
104 Blocs.add(new Bloc(Type.TROU, 16, 9));
105 Blocs.add(new Bloc(Type.TROU, 16, 13));
106
107 Blocs.add(new Bloc(Type.TROU, 17, 0));
108 Blocs.add(new Bloc(Type.TROU, 17, 13));
109
110 Blocs.add(new Bloc(Type.TROU, 18, 0));
111 Blocs.add(new Bloc(Type.TROU, 18, 13));
112
113 Blocs.add(new Bloc(Type.TROU, 19, 0));
114 Blocs.add(new Bloc(Type.TROU, 19, 1));
115 Blocs.add(new Bloc(Type.TROU, 19, 2));
116 Blocs.add(new Bloc(Type.TROU, 19, 3));
117 Blocs.add(new Bloc(Type.TROU, 19, 4));
118 Blocs.add(new Bloc(Type.TROU, 19, 5));
119 Blocs.add(new Bloc(Type.TROU, 19, 6));
120 Blocs.add(new Bloc(Type.TROU, 19, 7));
121 Blocs.add(new Bloc(Type.TROU, 19, 8));
122 Blocs.add(new Bloc(Type.TROU, 19, 9));
123 Blocs.add(new Bloc(Type.TROU, 19, 10));
124 Blocs.add(new Bloc(Type.TROU, 19, 11));
125 Blocs.add(new Bloc(Type.TROU, 19, 12));
126 Blocs.add(new Bloc(Type.TROU, 19, 13));
127
128 Blocs.add(new Bloc(Type.DEPART, 2, 2));
129
130 Blocs.add(new Bloc(Type.ARRIVEE, 8, 11));
```

Comme vous pouvez le voir, ma méthode pour construire un bloc est simple, j'ai besoin de :

- Son type (`TROU`, `DEPART` ou `ARRIVEE`).
- Sa position sur l'axe x (attention, sa position en blocs et pas en pixels. Par exemple, si je mets 5, je parle du cinquième bloc, pas du cinquième pixel).
- Sa position sur l'axe y (en blocs aussi).

Ma solution

Le Manifest

La première chose à faire est de modifier le Manifest. Vous verrez deux choses particulières :

- L'appareil est bloqué en mode paysage (`<activity android:configChanges="orientation" android:screenOrientation="landscape" >`).
- L'application n'est pas montrée aux utilisateurs qui n'ont pas d'accéléromètre

(<uses-feature android:name="android.hardware.sensor.accelerometer" android:required="true" />).

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sdz.chapitreCinq"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="7"
    android:targetSdkVersion="7" />

  <uses-feature
    android:name="android.hardware.sensor.accelerometer"
    android:required="true" />

  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name" >
    <activity
      android:name="sdz.chapitreCinq.LabyrintheActivity"
      android:configChanges="orientation"
      android:label="@string/app_name"
      android:screenOrientation="landscape" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

Les modèles

Nous allons tout d'abord voir les différents modèles qui permettent de décrire les composants de notre jeu.

Les blocs

```
import android.graphics.RectF;

public class Bloc {
    enum  Type { TROU, DEPART, ARRIVEE };

```

```
6       private float SIZE = Boule.RAYON * 2;
7
8       private Type mType = null;
9       private RectF mRectangle = null;
10
11      public Type getType() {
12          return mType;
13      }
14
15      public RectF getRectangle() {
16          return mRectangle;
17      }
18
19      public Bloc(Type pType, int pX, int pY) {
20          this.mType = pType;
21          this.mRectangle = new RectF(pX * SIZE, pY * SIZE, (pX +
                1) * SIZE, (pY + 1) * SIZE);
22      }
23  }
```

Rien de spécial ici, je vous ai déjà parlé de tout auparavant. Remarquez le calcul qui permet de placer un bloc en fonction de sa position en tant que bloc et non en pixels.

La boule

```
1   import android.graphics.Color;
2   import android.graphics.RectF;
3
4   public class Boule {
5       // Rayon de la boule
6       public static final int RAYON = 10;
7
8       // Couleur de la boule
9       private int mCouleur = Color.GREEN;
10      public int getCouleur() {
11          return mCouleur;
12      }
13
14      // Vitesse maximale autorisée pour la boule
15      private static final float MAX_SPEED = 20.0f;
16
17      // Permet à la boule d'accélérer moins vite
18      private static final float COMPENSATEUR = 8.0f;
19
20      // Utilisé pour compenser les rebonds
21      private static final float REBOND = 1.75f;
22
23      // Le rectangle qui correspond à la position de départ de
           la boule
24      private RectF mInitialRectangle = null;
25
```

```
26      // A partir du rectangle initial on détermine la position
            de la boule
27      public void setInitialRectangle(RectF pInitialRectangle) {
28          this.mInitialRectangle = pInitialRectangle;
29          this.mX = pInitialRectangle.left + RAYON;
30          this.mY = pInitialRectangle.top + RAYON;
31      }
32
33      // Le rectangle de collision
34      private RectF mRectangle = null;
35
36      // Coordonnées en x
37      private float mX;
38      public float getX() {
39          return mX;
40      }
41      public void setPosX(float pPosX) {
42          mX = pPosX;
43
44          // Si la boule sort du cadre, on rebondit
45          if(mX < RAYON) {
46              mX = RAYON;
47              // Rebondir, c'est changer la direction de la balle
48              mSpeedY = -mSpeedY / REBOND;
49          } else if(mX > mWidth - RAYON) {
50              mX = mWidth - RAYON;
51              mSpeedY = -mSpeedY / REBOND;
52          }
53      }
54
55      // Coordonnées en y
56      private float mY;
57      public float getY() {
58          return mY;
59      }
60
61      public void setPosY(float pPosY) {
62          mY = pPosY;
63          if(mY < RAYON) {
64              mY = RAYON;
65              mSpeedX = -mSpeedX / REBOND;
66          } else if(mY > mHeight - RAYON) {
67              mY = mHeight - RAYON;
68              mSpeedX = -mSpeedX / REBOND;
69          }
70      }
71
72      // Vitesse sur l'axe x
73      private float mSpeedX = 0;
74      // Utilisé quand on rebondit sur les murs horizontaux
```

```
75      public void changeXSpeed() {
76          mSpeedX = -mSpeedX;
77      }
78
79      // Vitesse sur l'axe y
80      private float mSpeedY = 0;
81      // Utilisé quand on rebondit sur les murs verticaux
82      public void changeYSpeed() {
83          mSpeedY = -mSpeedY;
84      }
85
86      // Taille de l'écran en hauteur
87      private int mHeight = -1;
88      public void setHeight(int pHeight) {
89          this.mHeight = pHeight;
90      }
91
92      // Taille de l'écran en largeur
93      private int mWidth = -1;
94      public void setWidth(int pWidth) {
95          this.mWidth = pWidth;
96      }
97
98      public Boule() {
99          mRectangle = new RectF();
100     }
101
102     // Mettre à jour les coordonnées de la boule
103     public RectF putXAndY(float pX, float pY) {
104         mSpeedX += pX / COMPENSATEUR;
105         if(mSpeedX > MAX_SPEED)
106             mSpeedX = MAX_SPEED;
107         if(mSpeedX < -MAX_SPEED)
108             mSpeedX = -MAX_SPEED;
109
110         mSpeedY += pY / COMPENSATEUR;
111         if(mSpeedY > MAX_SPEED)
112             mSpeedY = MAX_SPEED;
113         if(mSpeedY < -MAX_SPEED)
114             mSpeedY = -MAX_SPEED;
115
116         setPosX(mX + mSpeedY);
117         setPosY(mY + mSpeedX);
118
119         // Met à jour les coordonnées du rectangle de collision
120         mRectangle.set(mX - RAYON, mY - RAYON, mX + RAYON, mY +
                RAYON);
121
122         return mRectangle;
123     }
```

```
    // Remet la boule à sa position de départ
    public void reset() {
        mSpeedX = 0;
        mSpeedY = 0;
        this.mX = mInitialRectangle.left + RAYON;
        this.mY = mInitialRectangle.top + RAYON;
    }
}
```

Le moteur graphique

```
import java.util.List;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class LabyrintheView extends SurfaceView implements
    SurfaceHolder.Callback {
    Boule mBoule;
    public Boule getBoule() {
        return mBoule;
    }

    public void setBoule(Boule pBoule) {
        this.mBoule = pBoule;
    }

    SurfaceHolder mSurfaceHolder;
    DrawingThread mThread;

    private List<Bloc> mBlocks = null;
    public List<Bloc> getBlocks() {
        return mBlocks;
    }

    public void setBlocks(List<Bloc> pBlocks) {
        this.mBlocks = pBlocks;
    }

    Paint mPaint;

    public LabyrintheView(Context pContext) {
        super(pContext);
        mSurfaceHolder = getHolder();
        mSurfaceHolder.addCallback(this);
```

```
38            mThread = new DrawingThread();
39
40            mPaint = new Paint();
41            mPaint.setStyle(Paint.Style.FILL);
42
43            mBoule = new Boule();
44        }
45
46        @Override
47        protected void onDraw(Canvas pCanvas) {
48            // Dessiner le fond de l'écran en premier
49            pCanvas.drawColor(Color.CYAN);
50            if(mBlocks != null) {
51                // Dessiner tous les blocs du labyrinthe
52                for(Bloc b : mBlocks) {
53                    switch(b.getType()) {
54                    case DEPART:
55                        mPaint.setColor(Color.WHITE);
56                        break;
57                    case ARRIVEE:
58                        mPaint.setColor(Color.RED);
59                        break;
60                    case TROU:
61                        mPaint.setColor(Color.BLACK);
62                        break;
63                    }
64                    pCanvas.drawRect(b.getRectangle(), mPaint);
65                }
66            }
67
68            // Dessiner la boule
69            if(mBoule != null) {
70                mPaint.setColor(mBoule.getCouleur());
71                pCanvas.drawCircle(mBoule.getX(), mBoule.getY(),
                      Boule.RAYON, mPaint);
72            }
73        }
74
75        @Override
76        public void surfaceChanged(SurfaceHolder pHolder, int
             pFormat, int pWidth, int pHeight) {
77            //
78        }
79
80        @Override
81        public void surfaceCreated(SurfaceHolder pHolder) {
82            mThread.keepDrawing = true;
83            mThread.start();
84            // Quand on crée la boule, on lui indique les coordonné
                 es de l'écran
```

```
85          if(mBoule != null ) {
86              this.mBoule.setHeight(getHeight());
87              this.mBoule.setWidth(getWidth());
88          }
89      }
90
91      @Override
92      public void surfaceDestroyed(SurfaceHolder pHolder) {
93          mThread.keepDrawing = false;
94          boolean retry = true;
95          while (retry) {
96              try {
97                  mThread.join();
98                  retry = false;
99              } catch (InterruptedException e) {}
100         }
101
102     }
103
104     private class DrawingThread extends Thread {
105         boolean keepDrawing = true;
106
107         @Override
108         public void run() {
109             Canvas canvas;
110             while (keepDrawing) {
111                 canvas = null;
112
113                 try {
114                     canvas = mSurfaceHolder.lockCanvas();
115                     synchronized (mSurfaceHolder) {
116                         onDraw(canvas);
117                     }
118                 } finally {
119                     if (canvas != null)
120                         mSurfaceHolder.unlockCanvasAndPost(
                                canvas);
121                 }
122
123                 // Pour dessiner à 50 fps
124                 try {
125                     Thread.sleep(20);
126                 } catch (InterruptedException e) {}
127             }
128         }
129     }
130 }
```

Rien de formidable ici non plus, on se contente de reprendre le *framework* et d'ajouter les dessins dedans.

Le moteur physique

```
import java.util.ArrayList;
import java.util.List;

import sdz.chapitreCinq.Bloc.Type;
import android.app.Service;
import android.graphics.RectF;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;

public class LabyrintheEngine {
    private Boule mBoule = null;
    public Boule getBoule() {
        return mBoule;
    }

    public void setBoule(Boule pBoule) {
        this.mBoule = pBoule;
    }

    // Le labyrinthe
    private List<Bloc> mBlocks = null;

    private LabyrintheActivity mActivity = null;

    private SensorManager mManager = null;
    private Sensor mAccelerometre = null;

    SensorEventListener mSensorEventListener = new
        SensorEventListener() {

        @Override
        public void onSensorChanged(SensorEvent pEvent) {
            float x = pEvent.values[0];
            float y = pEvent.values[1];

            if(mBoule != null) {
                // On met à jour les coordonnées de la boule
                RectF hitBox = mBoule.putXAndY(x, y);

                // Pour tous les blocs du labyrinthe
                for(Bloc block : mBlocks) {
                    // On crée un nouveau rectangle pour ne pas
                        modifier celui du bloc
                    RectF inter = new RectF(block.getRectangle
                        ());
                    if(inter.intersect(hitBox)) {
```

```
46                              // On agit différement en fonction du
                                    type de bloc
47                              switch(block.getType()) {
48                              case TROU:
49                                  mActivity.showDialog(
                                        LabyrintheActivity.DEFEAT_DIALOG
                                        );
50                                  break;
51
52                              case DEPART:
53                                  break;
54
55                              case ARRIVEE:
56                                  mActivity.showDialog(
                                        LabyrintheActivity.
                                        VICTORY_DIALOG);
57                                  break;
58                              }
59                              break;
60                          }
61                      }
62                  }
63              }
64
65              @Override
66              public void onAccuracyChanged(Sensor pSensor, int
                    pAccuracy) {
67
68              }
69          };
70
71          public LabyrintheEngine(LabyrintheActivity pView) {
72              mActivity = pView;
73              mManager = (SensorManager) mActivity.getBaseContext().
                    getSystemService(Service.SENSOR_SERVICE);
74              mAccelerometre = mManager.getDefaultSensor(Sensor.
                    TYPE_ACCELEROMETER);
75          }
76
77          // Remet à zéro l'emplacement de la boule
78          public void reset() {
79              mBoule.reset();
80          }
81
82          // Arrête le capteur
83          public void stop() {
84              mManager.unregisterListener(mSensorEventListener,
                    mAccelerometre);
85          }
86
```

```
87      // Redémarre le capteur
88      public void resume() {
89          mManager.registerListener(mSensorEventListener,
                mAccelerometre, SensorManager.SENSOR_DELAY_GAME);
90      }
91
92      // Construit le labyrinthe
93      public List<Bloc> buildLabyrinthe() {
94          mBlocks = new ArrayList<Bloc>();
95          mBlocks.add(new Bloc(Type.TROU, 0, 0));
96          mBlocks.add(new Bloc(Type.TROU, 0, 1));
97          mBlocks.add(new Bloc(Type.TROU, 0, 2));
98          mBlocks.add(new Bloc(Type.TROU, 0, 3));
99          mBlocks.add(new Bloc(Type.TROU, 0, 4));
100         mBlocks.add(new Bloc(Type.TROU, 0, 5));
101         mBlocks.add(new Bloc(Type.TROU, 0, 6));
102         mBlocks.add(new Bloc(Type.TROU, 0, 7));
103         mBlocks.add(new Bloc(Type.TROU, 0, 8));
104         mBlocks.add(new Bloc(Type.TROU, 0, 9));
105         mBlocks.add(new Bloc(Type.TROU, 0, 10));
106         mBlocks.add(new Bloc(Type.TROU, 0, 11));
107         mBlocks.add(new Bloc(Type.TROU, 0, 12));
108         mBlocks.add(new Bloc(Type.TROU, 0, 13));
109
110         mBlocks.add(new Bloc(Type.TROU, 1, 0));
111         mBlocks.add(new Bloc(Type.TROU, 1, 13));
112
113         mBlocks.add(new Bloc(Type.TROU, 2, 0));
114         mBlocks.add(new Bloc(Type.TROU, 2, 13));
115
116         mBlocks.add(new Bloc(Type.TROU, 3, 0));
117         mBlocks.add(new Bloc(Type.TROU, 3, 13));
118
119         mBlocks.add(new Bloc(Type.TROU, 4, 0));
120         mBlocks.add(new Bloc(Type.TROU, 4, 1));
121         mBlocks.add(new Bloc(Type.TROU, 4, 2));
122         mBlocks.add(new Bloc(Type.TROU, 4, 3));
123         mBlocks.add(new Bloc(Type.TROU, 4, 4));
124         mBlocks.add(new Bloc(Type.TROU, 4, 5));
125         mBlocks.add(new Bloc(Type.TROU, 4, 6));
126         mBlocks.add(new Bloc(Type.TROU, 4, 7));
127         mBlocks.add(new Bloc(Type.TROU, 4, 8));
128         mBlocks.add(new Bloc(Type.TROU, 4, 9));
129         mBlocks.add(new Bloc(Type.TROU, 4, 10));
130         mBlocks.add(new Bloc(Type.TROU, 4, 13));
131
132         mBlocks.add(new Bloc(Type.TROU, 5, 0));
133         mBlocks.add(new Bloc(Type.TROU, 5, 13));
134
135         mBlocks.add(new Bloc(Type.TROU, 6, 0));
```

```
136            mBlocks.add(new Bloc(Type.TROU, 6, 13));
137
138            mBlocks.add(new Bloc(Type.TROU, 7, 0));
139            mBlocks.add(new Bloc(Type.TROU, 7, 1));
140            mBlocks.add(new Bloc(Type.TROU, 7, 2));
141            mBlocks.add(new Bloc(Type.TROU, 7, 5));
142            mBlocks.add(new Bloc(Type.TROU, 7, 6));
143            mBlocks.add(new Bloc(Type.TROU, 7, 9));
144            mBlocks.add(new Bloc(Type.TROU, 7, 10));
145            mBlocks.add(new Bloc(Type.TROU, 7, 11));
146            mBlocks.add(new Bloc(Type.TROU, 7, 12));
147            mBlocks.add(new Bloc(Type.TROU, 7, 13));
148
149            mBlocks.add(new Bloc(Type.TROU, 8, 0));
150            mBlocks.add(new Bloc(Type.TROU, 8, 5));
151            mBlocks.add(new Bloc(Type.TROU, 8, 9));
152            mBlocks.add(new Bloc(Type.TROU, 8, 13));
153
154            mBlocks.add(new Bloc(Type.TROU, 9, 0));
155            mBlocks.add(new Bloc(Type.TROU, 9, 5));
156            mBlocks.add(new Bloc(Type.TROU, 9, 9));
157            mBlocks.add(new Bloc(Type.TROU, 9, 13));
158
159            mBlocks.add(new Bloc(Type.TROU, 10, 0));
160            mBlocks.add(new Bloc(Type.TROU, 10, 5));
161            mBlocks.add(new Bloc(Type.TROU, 10, 9));
162            mBlocks.add(new Bloc(Type.TROU, 10, 13));
163
164            mBlocks.add(new Bloc(Type.TROU, 11, 0));
165            mBlocks.add(new Bloc(Type.TROU, 11, 5));
166            mBlocks.add(new Bloc(Type.TROU, 11, 9));
167            mBlocks.add(new Bloc(Type.TROU, 11, 13));
168
169            mBlocks.add(new Bloc(Type.TROU, 12, 0));
170            mBlocks.add(new Bloc(Type.TROU, 12, 1));
171            mBlocks.add(new Bloc(Type.TROU, 12, 2));
172            mBlocks.add(new Bloc(Type.TROU, 12, 3));
173            mBlocks.add(new Bloc(Type.TROU, 12, 4));
174            mBlocks.add(new Bloc(Type.TROU, 12, 5));
175            mBlocks.add(new Bloc(Type.TROU, 12, 9));
176            mBlocks.add(new Bloc(Type.TROU, 12, 8));
177            mBlocks.add(new Bloc(Type.TROU, 12, 13));
178
179            mBlocks.add(new Bloc(Type.TROU, 13, 0));
180            mBlocks.add(new Bloc(Type.TROU, 13, 8));
181            mBlocks.add(new Bloc(Type.TROU, 13, 13));
182
183            mBlocks.add(new Bloc(Type.TROU, 14, 0));
184            mBlocks.add(new Bloc(Type.TROU, 14, 8));
185            mBlocks.add(new Bloc(Type.TROU, 14, 13));
```

```
186
187             mBlocks.add(new Bloc(Type.TROU, 15, 0));
188             mBlocks.add(new Bloc(Type.TROU, 15, 8));
189             mBlocks.add(new Bloc(Type.TROU, 15, 13));
190
191             mBlocks.add(new Bloc(Type.TROU, 16, 0));
192             mBlocks.add(new Bloc(Type.TROU, 16, 4));
193             mBlocks.add(new Bloc(Type.TROU, 16, 5));
194             mBlocks.add(new Bloc(Type.TROU, 16, 6));
195             mBlocks.add(new Bloc(Type.TROU, 16, 7));
196             mBlocks.add(new Bloc(Type.TROU, 16, 8));
197             mBlocks.add(new Bloc(Type.TROU, 16, 9));
198             mBlocks.add(new Bloc(Type.TROU, 16, 13));
199
200             mBlocks.add(new Bloc(Type.TROU, 17, 0));
201             mBlocks.add(new Bloc(Type.TROU, 17, 13));
202
203             mBlocks.add(new Bloc(Type.TROU, 18, 0));
204             mBlocks.add(new Bloc(Type.TROU, 18, 13));
205
206             mBlocks.add(new Bloc(Type.TROU, 19, 0));
207             mBlocks.add(new Bloc(Type.TROU, 19, 1));
208             mBlocks.add(new Bloc(Type.TROU, 19, 2));
209             mBlocks.add(new Bloc(Type.TROU, 19, 3));
210             mBlocks.add(new Bloc(Type.TROU, 19, 4));
211             mBlocks.add(new Bloc(Type.TROU, 19, 5));
212             mBlocks.add(new Bloc(Type.TROU, 19, 6));
213             mBlocks.add(new Bloc(Type.TROU, 19, 7));
214             mBlocks.add(new Bloc(Type.TROU, 19, 8));
215             mBlocks.add(new Bloc(Type.TROU, 19, 9));
216             mBlocks.add(new Bloc(Type.TROU, 19, 10));
217             mBlocks.add(new Bloc(Type.TROU, 19, 11));
218             mBlocks.add(new Bloc(Type.TROU, 19, 12));
219             mBlocks.add(new Bloc(Type.TROU, 19, 13));
220
221             Bloc b = new Bloc(Type.DEPART, 2, 2);
222             mBoule.setInitialRectangle(new RectF(b.getRectangle()))
                    ;
223             mBlocks.add(b);
224
225             mBlocks.add(new Bloc(Type.ARRIVEE, 8, 11));
226
227             return mBlocks;
228         }
229
230     }
```

L’activité

```
1   import java.util.List;
```

```
2
3 import android.app.Activity;
4 import android.app.AlertDialog;
5 import android.app.Dialog;
6 import android.content.DialogInterface;
7 import android.os.Bundle;
8
9 public class LabyrintheActivity extends Activity {
10     // Identifiant de la boîte de dialogue de victoire
11     public static final int VICTORY_DIALOG = 0;
12     // Identifiant de la boîte de dialogue de défaite
13     public static final int DEFEAT_DIALOG = 1;
14
15     // Le moteur graphique du jeu
16     private LabyrintheView mView = null;
17     // Le moteur physique du jeu
18     private LabyrintheEngine mEngine = null;
19
20     @Override
21     public void onCreate(Bundle savedInstanceState) {
22         super.onCreate(savedInstanceState);
23
24         mView = new LabyrintheView(this);
25         setContentView(mView);
26
27         mEngine = new LabyrintheEngine(this);
28
29         Boule b = new Boule();
30         mView.setBoule(b);
31         mEngine.setBoule(b);
32
33         List<Bloc> mList = mEngine.buildLabyrinthe();
34         mView.setBlocks(mList);
35     }
36
37     @Override
38     protected void onResume() {
39         super.onResume();
40         mEngine.resume();
41     }
42
43     @Override
44     protected void onPause() {
45         super.onStop();
46         mEngine.stop();
47     }
48
49     @Override
50     public Dialog onCreateDialog (int id) {
```

```
51          AlertDialog.Builder builder = new AlertDialog.Builder(
                this);
52          switch(id) {
53          case VICTORY_DIALOG:
54              builder.setCancelable(false)
55              .setMessage("Bravo, vous avez gagné !")
56              .setTitle("Champion ! Le roi des Zörglubienotchs
                    est mort grâce à vous !")
57              .setNeutralButton("Recommencer", new
                    DialogInterface.OnClickListener() {
58                  @Override
59                  public void onClick(DialogInterface dialog, int
                        which) {
60                      // L'utilisateur peut recommencer s'il le
                            veut
61                      mEngine.reset();
62                      mEngine.resume();
63                  }
64              });
65              break;
66
67          case DEFEAT_DIALOG:
68              builder.setCancelable(false)
69              .setMessage("La Terre a été détruite à cause de vos
                    erreurs.")
70              .setTitle("Bah bravo !")
71              .setNeutralButton("Recommencer", new
                    DialogInterface.OnClickListener() {
72                  @Override
73                  public void onClick(DialogInterface dialog, int
                        which) {
74                      mEngine.reset();
75                      mEngine.resume();
76                  }
77              });
78          }
79          return builder.create();
80      }
81
82      @Override
83      public void onPrepareDialog (int id, Dialog box) {
84          // A chaque fois qu'une boîte de dialogue est lancée,
                on arrête le moteur physique
85          mEngine.stop();
86      }
87  }
```

▷ Télécharger le projet
Code web : 437312

Améliorations envisageables

Proposer plusieurs labyrinthes

Ce projet est quand même très limité, il ne propose qu'un labyrinthe. Avouons que jouer au même labyrinthe *ad vitam aeternam* est assez ennuyeux. On va alors envisager un système pour charger plusieurs labyrinthes. La première chose à faire, c'est de rajouter un modèle pour les labyrinthes. Il contiendra au moins une liste de blocs, comme précédemment :

```
public class Labyrinthe {
  List<Bloc> mBlocs = null;
}
```

Il suffira ensuite de passer le labyrinthe aux moteurs et de tout réinitialiser. Ainsi, on redessinera le labyrinthe, on cherchera le nouveau départ et on y placera la boule.

Enfin, si on fait cela, notre problème n'est pas vraiment résolu. C'est vrai qu'on pourra avoir plusieurs labyrinthes et qu'on pourra alterner entre eux, mais si on doit créer chaque fois un labyrinthe bloc par bloc, cela risque d'être quand même assez laborieux. Alors, comment créer un labyrinthe autrement ?

Une solution élégante serait d'avoir les labyrinthes enregistrés sur un fichier de façon à n'avoir qu'à le lire pour récupérer un labyrinthe et le partager avec le monde. Imaginons un peu comment fonctionnerait ce système. On pourrait avoir un fichier texte et chaque caractère correspondrait à un type de bloc. Par exemple :

- `o` serait un trou ;
- `d`, le départ ;
- `a`, l'arrivée.

Si on envisage ce système, le labyrinthe précédent donnerait ceci :

```
oooooooooooooooooooo
o   o  o    o      o
o d o  o    o      o
o   o       o      o
o   o       o   o  o
o   o  oooooo   o  o
o   o  o        o  o
o   o           o  o
o   o  o    ooooo  o
o   o  oooooo      o
o   o  o           o
o      oa          o
o      o           o
oooooooooooooooooooo
```

C'est tout de suite plus graphique, plus facile à développer, à entretenir et à déboguer. Pour transformer ce fichier texte en labyrinthe, il suffit de créer une boucle qui lira le fichier caractère par caractère, puis qui créera un bloc en fonction de la présence ou non d'un caractère à l'emplacement lu :

```
1  InputStreamReader input = null;
2  BufferedReader reader = null;
3  Bloc bloc = null;
4  try {
5    input = new InputStreamReader(new FileInputStream(
         fichier_du_labyrinthe), Charset.forName("UTF-8"));
6    reader = new BufferedReader(input);
7
8    // L'indice qui correspond aux colonnes dans le fichier
9    int i = 0;
10   // L'indice qui correspond aux lignes dans le fichier
11   int j = 0;
12
13   // La valeur récupérée par le flux
14   int c;
15   // Tant que la valeur n'est pas de -1, c'est qu'on lit un
         caractère du fichier
16   while((c = reader.read()) != -1) {
17     char character = (char) c;
18     if(character == 'o')
19       bloc = new Bloc(Type.TROU, i, j);
20     else if(character == 'd')
21       bloc = new Bloc(Type.DEPART, i, j);
22     else if(character == 'a')
23       bloc = new Bloc(Type.ARRIVEE, i, j);
24     else if (character == '\n') {
25       // Si le caractère est un retour à la ligne, on retourne
           avant la première colonne
26       // Car on aura i++ juste après, ainsi i vaudra 0, la
           première colonne !
27       i = -1;
28       // Et on passe à la ligne suivante
29       j++;
30     }
31     // Si le bloc n'est pas nul, alors le caractère n'était pas
         un retour à la ligne
32     if(bloc != null)
33       // On l'ajoute alors au labyrinthe
34       labyrinthe.addBloc(bloc);
35     // On passe à la colonne suivante
36     i++;
37     // On remet bloc à null, utile quand on a un retour à la
         ligne pour ne pas ajouter de bloc qui n'existe pas
38     bloc = null;
39   }
40 } catch (IllegalCharsetNameException e) {
41   e.printStackTrace();
42 } catch (UnsupportedCharsetException e) {
43   e.printStackTrace();
44 } catch (FileNotFoundException e) {
```

```
45      e.printStackTrace();
46    } catch (IOException e) {
47      e.printStackTrace();
48    } finally {
49      if(input != null)
50        try {
51          input.close();
52        } catch (IOException e1) {
53          e1.printStackTrace();
54        }
55      if(reader != null)
56        try {
57          reader.close();
58        } catch (IOException e) {
59          e.printStackTrace();
60        }
61    }
```

Pour les plus motivés d'entre vous, il est possible aussi de développer un éditeur de niveaux. Imaginez, vous possédez un menu qui permet de choisir le bloc à ajouter, puis il suffira à l'utilisateur de cliquer à l'endroit où il voudra que le bloc se place.

Vérifiez toujours qu'un labyrinthe a un départ et une arrivée, sinon l'utilisateur va tourner en rond pendant des heures ou n'aura même pas de boule !

Ajouter des sons

Parce qu'un peu de musique et des effets sonores permettent d'améliorer l'immersion. Enfin, si tant est qu'on puisse avoir de l'immersion dans ce genre de jeux avec de si jolis graphismes... Bref, il existe deux types de sons que devrait jouer notre jeu :

- Une musique de fond ;
- Des effets sonores. Par exemple, quand la boule de l'utilisateur tombe dans un trou, cela pourrait être amusant d'avoir le son d'une foule qui le hue.

Pour la musique, c'est simple, vous savez déjà le faire ! Utilisez un `MediaPlayer` pour jouer la musique en fond, ce n'est pas plus compliqué que cela. Si vous avez plusieurs musiques, vous pouvez aussi très bien créer une liste de lecture et passer d'une chanson à l'autre dès que la lecture d'une piste est terminée.

Pour les effets sonores, c'est beaucoup plus subtil. On va plutôt utiliser un `SoundPool`. En effet, il est possible qu'on ait à jouer plusieurs effets sonores en même temps, ce que `MediaPlayer` ne gère pas correctement ! De plus, `MediaPlayer` est lourd à utiliser, et on voudra qu'un effet sonore soit plutôt réactif. C'est pourquoi on va se pencher sur `SoundPool`.

Contrairement à `MediaPlayer`, `SoundPool` va devoir précharger les sons qu'il va jouer au lancement de l'application. Les sons vont être convertis en un format que supportera

mieux Android afin de diminuer la latence de leur lecture. Pour les plus minutieux, vous pouvez même gérer le nombre de flux audio que vous voulez en même temps. Si vous demandez à `SoundPool` de jouer un morceau de plus que vous ne l'avez autorisé, il va automatiquement fermer un flux précédent, généralement le plus ancien. Enfin, vous pouvez aussi préciser une priorité manuellement pour gérer les flux que vous souhaitez garder. Par exemple, si vous jouez la musique dans un `SoundPool`, il faudrait pouvoir la garder quoi qu'il arrive, même si le nombre de flux autorisés est dépassé. Vous pouvez donc donner à la musique de fond une grosse priorité pour qu'elle ne soit pas fermée.

Ainsi, le plus gros défaut de cette méthode est qu'elle prend du temps au chargement. Vous devez insérer chaque son que vous allez utiliser avec la méthode `int load(String path, int priority)`, `path` étant l'emplacement du son et `priority` la priorité que vous souhaitez lui donner (0 étant la valeur la plus basse possible). L'entier retourné sera l'identifiant de ce son, gardez donc cette valeur précieusement.

Si vous avez plusieurs niveaux, et que chaque niveau utilise un ensemble de sons différents, il est important que le chargement des sons se fasse en parallèle du chargement du niveau (dans un thread, donc) et surtout tout au début, pour que le chargement ne soit pas trop retardé par ce processus lent.

Une fois le niveau chargé, vous pouvez lancer la lecture d'un son avec la méthode `int play(int soundID, float leftVolume, float rightVolume, int priority, int loop, float rate)`, les paramètres étant :

- En tout premier l'identifiant du son, qui vous a été donné par la méthode `load()`.
- Le volume à gauche et le volume à droite, utile pour la lecture en stéréo. La valeur la plus basse est 0, la plus haute est 1.
- La priorité de ce flux. 0 est le plus bas possible.
- Le nombre de fois que le son doit être répété. On met 0 pour jamais, -1 pour toujours, toute autre valeur positive pour un nombre précis.
- Et enfin la vitesse de lecture. 1.0 est la vitesse par défaut, 2.0 sera deux fois plus rapide et 0.5 deux fois plus lent.

La valeur retournée est l'identifiant du flux. C'est intéressant, car cela vous permet de manipuler votre flux. Par exemple, vous pouvez arrêter un flux avec `void pause(int streamID)` et le reprendre avec `void resume(int streamID)`.

Enfin, une fois que vous avez fini un niveau, il vous faut appeler la méthode `void release()` pour libérer la mémoire, en particulier les sons retenus en mémoire. La référence au `SoundPool` vaudra `null`. Il vous faut donc créer un nouveau `SoundPool` par niveau, cela vous permet de libérer la mémoire entre chaque chargement.

Créer le moteur graphique et physique du jeu requiert beaucoup de temps et d'effort. C'est pourquoi il est souvent conseillé de faire appel à des moteurs préexistants comme AndEngine par exemple, qui est gratuit et *open source*. Son utilisation sort du cadre de ce cours ; cependant, si vous voulez faire un jeu, je vous conseille de vous y pencher sérieusement.

Sixième partie

Annexes

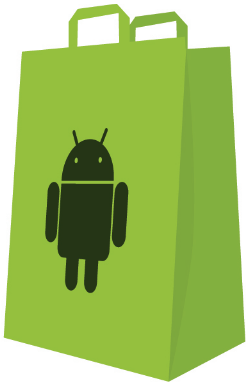
Chapitre 29

Publier et rentabiliser une application

Difficulté :

Vous avez développé, débogué, testé, re-débogué votre application, qui est impeccable. Vous choisissez déjà la voiture que vous allez acheter avec les recettes de votre application... mais en attendant, vous êtes le seul à l'utiliser sur un émulateur ou sur votre téléphone. C'est pourquoi nous allons parler d'une étape indispensable, celle pour laquelle vous avez tant travaillé : nous allons voir comment publier votre application !

Avant que vous puissiez distribuer votre application, je vais vous apprendre comment la préparer en vue de la distribuer, puis nous verrons ensuite les différentes manières de financer votre travail. Enfin, nous terminerons sur les supports qui permettent de mettre à disposition des autres votre application, en portant une attention particulière sur Google Play.



Préparez votre application à une distribution

Déjà, il faut que vous sachiez comment exporter votre application sous la forme d'un `.apk`. Un **APK** est un format de fichier qui permet de distribuer et d'installer des applications Android. Un APK est en fait une archive (comme les ZIP ou les RAR) qui contient tous les fichiers nécessaires organisés d'une certaine manière. Pour exporter un de vos projets, il suffit de faire un clic droit dessus dans votre explorateur de fichiers, puis de cliquer sur `Android Tools > Export Unsigned Application Package...`

La différence entre cette méthode de compilation et celle que nous utilisons d'habitude est que l'application générée sera en version *release*, alors qu'en temps normal l'application générée est en version *debug*. Vous trouverez plus de détails sur ces termes dans les paragraphes qui suivent.

Modifications et vérifications d'usage

Effectuez des tests exhaustifs

Avant toute chose, avez-vous bien testé à fond votre application ? Et sur tous les types de support ? L'idéal serait bien entendu de pouvoir tester sur une grande variété de périphériques réels, mais je doute que tout le monde ait les moyens de posséder autant de terminaux. Une solution alternative plus raisonnable est d'utiliser l'AVD, puisqu'il permet d'émuler de nombreux matériels différents, alors n'hésitez pas à en abuser pour être certains que tout fonctionne correctement. Le plus important étant surtout de supporter le plus d'écrans possible.

Attention au nom du package

Ensuite, il vous faut faire attention au package dans lequel vous allez publier votre application. Il jouera un rôle d'identifiant pour votre application à chaque fois que vous la soumettrez, il doit donc être unique et ne pas changer entre deux soumissions. Si vous mettez à jour votre application, ce sera toujours dans le même package. Une technique efficace consiste à nommer le package comme est nommé votre site web, mais à l'envers. Par exemple, les applications Google sont dans le package `com.google`.

Arrêtez la journalisation

Supprimez toutes les sorties vers le Logcat de votre application (toutes les instructions du genre `Log.d` ou `Log.i` par exemple), ou au moins essayez de les minimiser. Alors bien entendu, enlever directement toutes les sorties vers le Logcat serait contre-productif puisqu'il faudrait les remettre dès qu'on en aurait besoin pour déboguer… Alors comment faire ?

Le plus pratique serait de les activer uniquement quand l'application est une version *debug*. Cependant, comment détecter que notre application est en version *debug* ou en version *release* ? C'est simple, il existe une variable qui change en fonction de la

version. Elle est connue sous le nom de `BuildConfig.DEBUG` et se trouve dans le fichier `BuildConfig.java`, lui-même situé dans le répertoire `gen`. Vous pouvez par exemple entourer chaque instance de `Log` ainsi :

```
if(BuildConfig.DEBUG)
{
  //Si on se trouve en version debug, alors on affiche des
      messages dans le Logcat
  Log.d(...);
}
```

Désactivez le débogage

N'oubliez pas non plus de désactiver le débogage de votre application ! Ainsi, si vous aviez inséré l'attribut `android:debuggable` dans votre Manifest, n'oubliez pas de l'enlever (il vaut `false` par défaut) ou d'insérer la valeur `false` à la place de `true`.

Nettoyez votre projet

Il se peut que vous ayez créé des fichiers qui ne sont pas nécessaires pour la version finale, qui ne feront qu'alourdir votre application, voire la rendre instable. Je pense par exemple à des jeux de test particuliers ou des éléments graphiques temporaires. Ainsi, les répertoires les plus susceptibles de contenir des déchets sont les répertoires `res/` ou encore `assets/`.

Faire attention au numéro de version

Le numéro de version est une information capitale, autant pour vous que pour l'utilisateur. Pour ce dernier, il permet de lui faire savoir que votre application a été mise à jour, et le rassure quant à l'intérêt d'un achat qu'il a effectué si l'application est régulièrement mise à jour. Pour vous, il vous permet de tracer les progrès de votre application, de vous placer des jalons et ainsi de mieux organiser le développement de votre projet.

Vous vous rappelez les attributs `android:versionName` et `android:versionCode` ? Le premier permet de donner une valeur sous forme de chaîne de caractères à la version de votre application (par exemple « 1.0 alpha » ou « 2.8.1b »). Cet attribut sera celui qui est montré à l'utilisateur, à l'opposé de `android:versionCode` qui ne sera pas montré à l'utilisateur et qui ne peut contenir que des nombres entiers. Ainsi, si votre ancien `android:versionCode` était « 1 », il vous suffira d'insérer un nombre supérieur à « 1 » pour que le marché d'applications sache qu'il s'agit d'une version plus récente.

On peut par exemple passer de :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/
    android"
  package="sdz.monprojet"
  android:versionCode="5"
```

```
    android:versionName="1.1b" >

  ...

</manifest>
```

... à :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="sdz.monprojet"
    android:versionCode="6"
    android:versionName="1.2" >

  ...

</manifest>
```

Enfin, de manière générale, il existe une syntaxe à respecter pour choisir la version d'un projet. Il s'agit d'écrire des nombres séparés par des points, sachant que les nombres les plus à gauche sont ceux qui indiquent les plus gros changements, avec la majorité du temps soit deux, soit trois nombres. C'est un peu compliqué à comprendre, alors voici un exemple. On présente les numéros de version ainsi : **<gros changement>.<moins gros changement>[.<encore plus petit changement qu'on n'indique pas forcément>]**. Si on avait une application en version **1.1** et que l'on a complètement changé l'interface graphique, on peut considérer passer en version **2.0**. En revanche, si on est à la version **1.3.1** et qu'on a effectué deux corrections de bug, alors on pourrait passer en version **1.3.2**. Il n'y a pas de démarche standard à ce sujet, alors je vous laisse juger vous-mêmes comment faire évoluer le numéro de version.

Manifestez-vous !

Le Manifest est susceptible de contenir des déchets que vous avez oublié de nettoyer. Vérifiez tout d'abord que les permissions que vous demandez ne sont pas trop abusives, car c'est une source de suspicions (justifiée) de la part des utilisateurs.

Indiquez aussi une version cohérente de `android:minSdkVersion` de façon à cibler le plus d'utilisateurs possible et à ne pas rendre votre application disponible à des utilisateurs qui ne pourraient pas l'utiliser. En effet, n'oubliez pas que c'est cette valeur qui détermine à qui sera proposée l'application.

Gérez les serveurs de test

Si vous utilisez des serveurs de test, vérifiez bien que vous changez les URL pour faire appel aux serveurs de production, sinon vos utilisateurs risquent d'avoir de grosses surprises. Et pas des bonnes. De plus, vérifiez que vos serveurs sont configurés pour une entrée en production et qu'ils sont sécurisés. Ce n'est cependant pas l'objet de ce cours, je ne vais pas vous donner de conseils à ce niveau-là.

Dessinez une icône attractive

Le succès de votre application pourrait dépendre de certains détails particuliers ! Votre icône est-elle esthétique ? Est-elle définie pour toutes les résolutions d'écran, histoire qu'elle ne se résume pas à une bouillie de pixels ? Il s'agit quand même du premier contact de l'utilisateur avec votre application, c'est avec l'icône qu'il va la retrouver dans la liste des applications, sur le marché d'applications, etc.

Comme il est possible d'avoir une icône par activité, vous pouvez aussi envisager d'exploiter cette fonctionnalité pour aider vos utilisateurs à se repérer plus facilement dans votre application.

Google a concocté un guide de conduite pour vous aider à dessiner une icône correcte.

▷ Guide de conduite
Code web : 436174

Protégez-vous légalement ainsi que votre travail

Si vous voulez vous protéger ou protéger vos projets, vous pouvez définir une licence de logiciel. Cette licence va définir comment peut être utilisée et redistribuée votre application. N'étant pas moi-même un expert dans le domaine, je vous invite à consulter un juriste pour qu'il vous renseigne sur les différentes opportunités qui s'offrent à vous.

Enfin, vous pouvez tout simplement ne pas instaurer de licence si c'est que vous désirez. De manière générale, on en trouve assez peu dans les applications mobiles, parce qu'elles sont pénibles à lire et ennuient l'utilisateur.

Signer l'application

Pour qu'une application puisse être installée sous Android, elle doit obligatoirement être signée. Signer une application signifie lui attribuer un certificat qui permet au système de l'authentifier. Vous allez me dire que jusqu'à maintenant vous n'avez jamais signé une application, puisque vous ignorez ce dont il s'agit, et que pourtant vos applications se sont toujours installées. Sauf qu'en fait Eclipse a toujours émis un certificat pour vous. Le problème est qu'il génère une clé de *debug*, et que ce type de clé, n'étant pas définie par un humain, n'est pas digne de confiance et n'est pas assez sûre pour être utilisée de manière professionnelle pour envoyer vos projets sur un marché d'applications. Si vous voulez publier votre application, il faudra générer une clé privée unique manuellement.

Pourquoi ?

Parce que cette procédure permet de sécuriser de manière fiable votre application, il s'agit donc d'une démarche importante. On peut considérer au moins deux avantages :

– Si plusieurs applications sont signées avec la même clé, alors elles peuvent commu-

niquer et être traitées comme une seule et même application, c'est pourquoi il est conseillé d'utiliser toujours le même certificat pour toutes vos applications. Comme vous savez que ce sont vos applications, vous leur faites confiance, alors il n'y a pas de raison qu'une de ces applications exécute un contenu malicieux pour un autre de vos projets ou pour le système.
– Une application ne peut être mise à jour que si elle possède une signature qui provient du même certificat. Si vous utilisez deux clés différentes pour une version d'une application et sa mise à jour, alors le marché d'applications refusera d'exécuter la mise à jour.

C'est pourquoi il faut que vous fassiez attention à deux choses très importantes :

– Tout d'abord, ne perdez pas vos clés, sinon vous serez dans l'impossibilité de mettre à jour vos applications. Vous pourrez toujours créer une nouvelle page pour votre projet, mais cette page ne serait plus liée à l'ancienne et elle perdrait tous ses commentaires et statistiques. En plus, dire à vos utilisateurs actuels qu'ils doivent désinstaller leur version du programme pour télécharger une autre application qui est en fait une mise à jour de l'ancienne application est un véritable calvaire, rien qu'à expliquer.
– Ensuite, utilisez des mots de passe qui ne soient pas trop évidents et évitez de vous les faire voler. Si quelqu'un vous vole votre clé et qu'il remplace votre application par un contenu frauduleux, c'est vous qui serez dans l'embarras, cela pourrait aller jusqu'à des soucis juridiques.

Comme on n'est jamais trop prudent, n'hésitez pas à faire des sauvegardes de vos clés, afin de ne pas les perdre à cause d'un bête formatage. Il existe des solutions de stockage sécurisées gratuites qui vous permettront de mettre vos clés à l'abri des curieux.

La procédure

Il existe deux manières de faire. Sans Eclipse, nous avons besoin de deux outils qui sont fournis avec le JDK : *Keytool* afin de créer le certificat et *Jarsigner* pour signer l'APK (c'est-à-dire lui associer un certificat). Nous allons plutôt utiliser l'outil d'Eclipse pour créer nos certificats et signer nos applications. Pour cela, faites un clic droit sur un projet et allez dans le menu que nous avons utilisé précédemment pour faire un APK, sauf que cette fois nous allons le signer grâce à `Android Tools > Export Signed Application Package...`

Cette action ouvrira l'écran visible à la figure 29.1. La première chose à faire est de choisir un projet qu'il vous faudra signer. Vous pouvez ensuite cliquer sur `Next`.

Une nouvelle fenêtre, visible à la figure 29.2, apparaît. Vous pouvez alors choisir soit un *keystore* existant déjà, soit en créer un nouveau. Le *keystore* est un fichier qui contiendra un ou plusieurs de vos certificats. Pour cela, vous aurez besoin d'un mot de passe qui soit assez sûr pour le protéger. Une fois votre choix fait, cliquez sur `Next`.

La fenêtre visible à la figure 29.3 s'affiche. C'est seulement maintenant que nous allons créer une clé.

Il vous faut entrer des informations pour les quatre premiers champs :

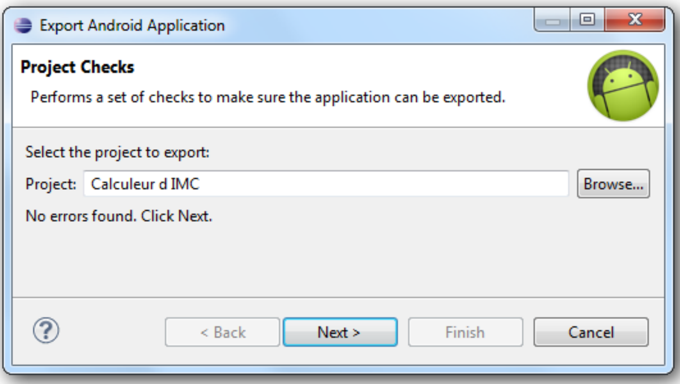


FIGURE 29.1 – Export Android Application

FIGURE 29.2 – Choisissez un keystore

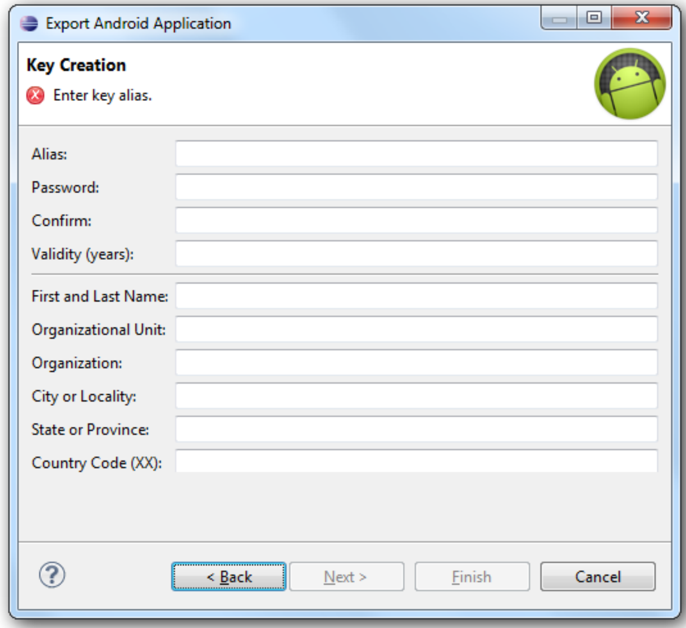


FIGURE 29.3 – Cette fenêtre permet de créer une clé

- Un alias qui permettra d'identifier votre clé.
- Un mot de passe et sa confirmation.
- Une limite de validité en années, sachant que la clé doit être disponible jusqu'au 22 octobre 2033 au moins. De manière générale, utilisez une valeur supérieure ou égale à 25. Moi, j'utilise 50.

Pour les champs suivants, il vous faut en renseigner *au moins* un. Cliquez ensuite sur `Next`. Une fenêtre s'ouvre (voir figure 29.4). Choisissez l'emplacement où sera créé l'APK et terminez en cliquant sur `Finish`.

Les moyens de distribution

Google Play

Les avantages d'utiliser Google Play sont plutôt nombreux. Déjà Google Play est énorme, il contient 600 000 applications en tout, et 1,5 milliard d'applications sont téléchargées tous les mois. Il vous permet de mettre à disposition d'un très grand nombre d'utilisateurs tous vos travaux, dans 190 pays et territoires, à moindre frais. En revanche, vous ne pourrez vendre vos applications que dans 132 pays et territoires, pour des raisons légales. De plus, Google Play dispose d'outils pour vous permettre d'analyser le comportement des consommateurs, de traquer les bugs qui traînent dans

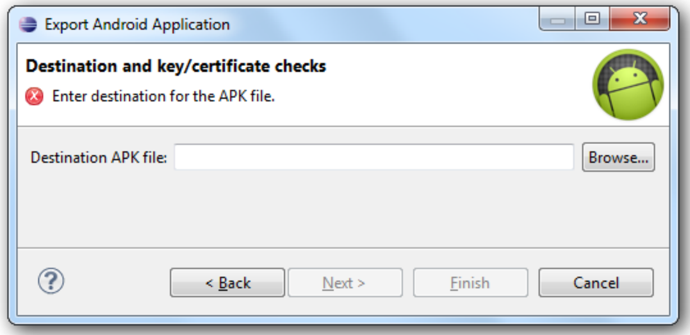


Figure 29.4 – Choisissez l'emplacement de l'APK

votre application et de gagner de l'argent en récompense de votre labeur.

La première chose à faire est d'avoir au moins un compte Google valide.

▷ Créer un compte
Code web : 490545

Ce site étant en français, j'imagine que vous vous débrouillerez comme des chefs durant les étapes de la création. Ensuite, il vous faut créer un compte développeur Android à.

▷ Créer un compte développeur
Code web : 637246

On vous demandera :

- De créer un compte développeur.
- De signer virtuellement la charte de distribution des applications Android.
- Puis de payer la somme de 25$ (vous aurez besoin d'une carte de crédit valide).

Une fois cela fait, vous pourrez publier **autant d'applications que vous le souhaitez !**

Une fois votre compte créé, le premier écran auquel vous vous trouverez confrontés est la console pour développeurs (voir figure 29.5). C'est dans cet écran que tout se fait, vous pouvez :

- Ajouter un développeur avec qui vous travaillez en équipe. Vous pourrez déterminer les différentes parties auxquelles ont accès vos collègues. Tous les développeurs n'ont par exemple pas besoin de voir les revenus financiers de vos projets.
- Publier une application et avoir des informations dessus.
- Se constituer un compte Google marchand pour pouvoir vendre vos applications.

Les applications

Si vous cliquez sur `Publier une application`, vous vous retrouverez confrontés à une deuxième fenêtre (voir figure 29.6) qui vous permettra de sélectionner l'APK qui sera

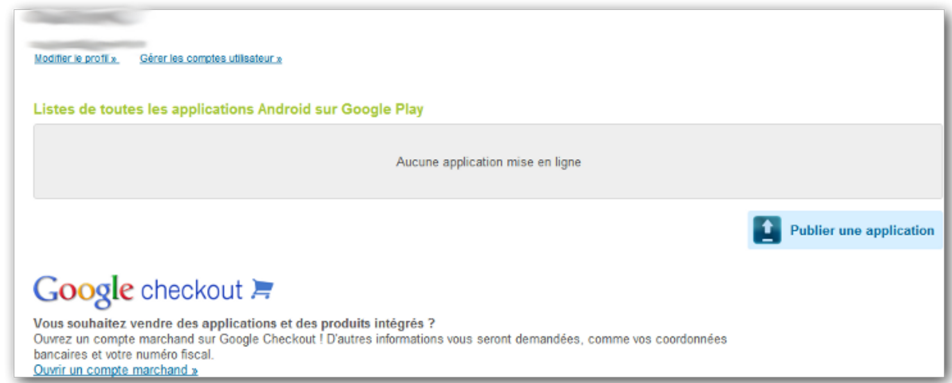


FIGURE 29.5 – La console pour développeurs

mis en ligne. Comme vous pouvez le voir, j'ai choisi de publier l'APK de ma superbe application qui dit salut aux Zéros et je m'apprête à l'importer.

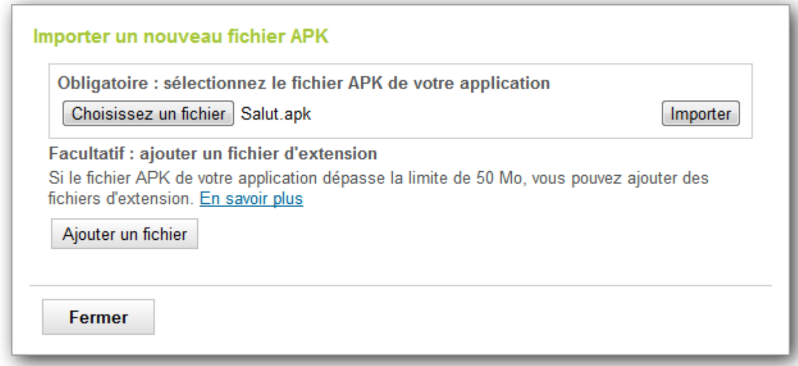


FIGURE 29.6 – Sélectionnez l'APK à mettre en ligne

Si votre application est un jeu, alors il y a des risques pour que l'APK fasse plus de 50 Mo avec les fichiers sonores et graphiques, et Google Play n'accepte que les archives qui font moins de 50 Mo. Il existe alors deux solutions, soit vous faites télécharger les fichiers supplémentaires sur un serveur distant — ce qui a un coût —, soit vous utilisez le bouton `Ajouter un fichier` pour ajouter ces fichiers supplémentaires qui doivent être mis en ligne — ce qui est gratuit mais demande plus de travail. Le problème avec l'hébergement sur un serveur distant est que les utilisateurs sont rarement satisfaits d'avoir à télécharger 500 Mo au premier lancement de l'application, c'est pourquoi il est quand même préférable d'opter pour la seconde option.

Vous pourrez ajouter deux fichiers qui font jusqu'à 2 Go. Un de ces fichiers contient toutes les données indispensables au lancement de l'application, alors que le second est

juste un patch afin de ne pas avoir à envoyer un APK complet sur le Store. De cette manière, les utilisateurs n'ont pas à télécharger encore une fois un gros fichier mais juste des modifications contenues dans ce fichier pendant une mise à jour.

Une fois votre APK importé, vous remarquerez que le site a réussi à extraire certaines informations depuis votre application, comme son nom et son icône, et tout cela à l'aide des informations contenues dans le Manifest.

C'est aussi à cet endroit que Google Play va vérifier la cohérence de ce que vous faites. En effet, si vous avez déjà une application qui a le même nom et que le numéro de version est identique ou inférieur à celui de l'application déjà en ligne, alors vous risquez bien de vous retrouver confrontés à un mur.

En cliquant sur l'autre onglet, vous vous retrouvez devant un grand nombre d'options, dont certaines sont obligatoires. Par exemple, il vous faut au moins deux captures d'écran de votre application ainsi qu'une icône en haute résolution, pour qu'elle soit affichée sur le Play Store.

L'encart suivant, visible à la figure 29.7, est tout aussi important, il vous permet de donner des indications quant à votre application.

Comme j'ai traduit mon application en anglais, j'ai décidé d'ajouter une description en anglais en cliquant sur `ajouter une langue`.

Si vous rajoutez un texte promotionnel, vous devrez aussi rajouter une image promotionnelle dans la partie précédente.

Enfin, la dernière partie vous permettra de régler certaines options relatives à la publication de votre application. L'une des sections les plus importantes ici étant la `catégorie de contenu`, qui vous permet de dire aux utilisateurs à qui est destinée cette application. Comme mon application ne possède aucun contenu sensible, j'ai indiqué qu'elle était accessible à tout public. Vous en saurez plus sur le support. On vous demandera aussi si vous souhaitez activer une protection contre la copie, mais je ne le recommande pas, puisque cela alourdit votre application et que le processus va bientôt être abandonné.

▷ En savoir plus
Code web : 473211

C'est aussi à cet endroit que vous déterminerez le prix de votre application. Notez que, si vous déclarez que votre application est gratuite, alors elle devra le rester tout le temps. Enfin, si vous voulez faire payer pour votre application, alors il vous faudra un compte marchand dans Google Checkout, comme nous le verrons plus loin.

Voilà, maintenant que vous avez tout configuré, activez votre APK dans l'onglet `Fichiers APK` et publiez votre application. Elle ne sera pas disponible immédiatement puisqu'il faut quand même qu'elle soit validée à un certain niveau (cela peut prendre quelques heures).

Informations sur l'application

Langue ajouter une langue	| *français (fr) | English (en) | L'astérisque (*) désigne la langue par défaut.
	x Supprimer la fiche en Français
Titre (Français)	Salut les Zéros ! 17 caractères (30 max.)
Description (Français)	C'est une application qui dit salut aux Zéros. 46 caractères (4000 max.)
Modifications récentes (Français) versionName: 1.0 [En savoir plus]	Avant elle disait pas salut aux Zéros, maintenant si. 53 caractères (500 max.)
Texte promotionnel (Français)	A télécharger si vous voulez qu'on vous dise salut et que vous êtes un Zéro. 76 caractères (80 max.)
Type d'application	Applications
Catégorie	Divertissement

FIGURE 29.7 – Renseignez quelques informations

Plusieurs APK pour une application

Comme vous le savez, un APK n'est disponible que pour une configuration bien précise de terminal, par exemple tous ceux qui ont un écran large, moyen ou petit. Il se peut cependant que vous ayez un APK spécial pour les écrans très larges, si votre application est compatible avec Google TV. En ce cas, il est possible d'avoir plusieurs APK pour une même application. En fait, l'APK qui sera téléchargé par l'utilisateur dépendra de l'adéquation entre sa configuration matérielle et celle précisée dans le Manifest. Le problème avec cette pratique, c'est qu'elle est contraignante puisqu'il faut entretenir plusieurs APK pour une même application... En général, cette solution est adoptée uniquement quand un seul APK fait plus de 50 Mo.

Informations sur une application

Elles sont accessibles à partir de la liste de vos applications, comme le montre la figure 29.8.

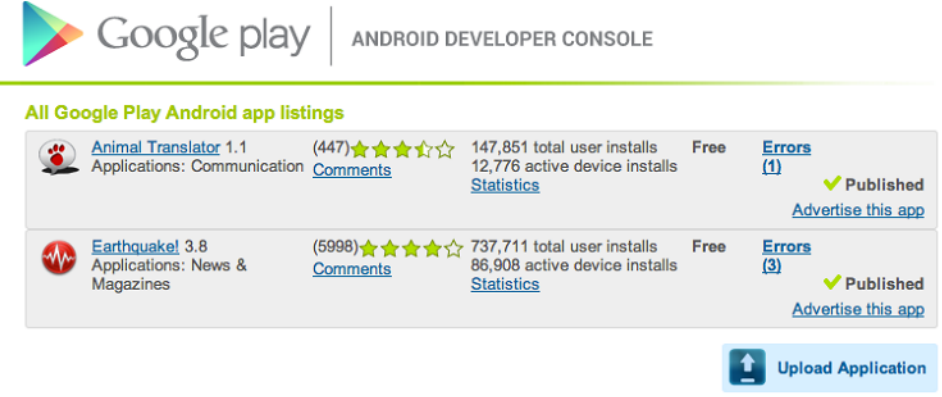


FIGURE 29.8 – Des applications sur le Google Play

Cliquer sur le nom de votre application vous permettra de modifier les informations que nous avons définies juste avant, et permet de mettre à jour votre application. Vous pouvez aussi voir les commentaires que laissent les utilisateurs au sujet de vos applications, comme à la figure 29.9.

Très souvent, les utilisateurs vous laissent des commentaires très constructifs que vous feriez bien de prendre en compte, ou alors ils vous demandent des fonctionnalités auxquelles vous n'aviez pas pensé et qui seraient une véritable plus-value pour votre produit. Ce sont les utilisateurs qui déterminent le succès de votre application, c'est donc eux qu'il faut contenter et prendre en considération. En plus, très bientôt il sera possible pour un éditeur de répondre à un utilisateur, afin d'approfondir encore plus la relation avec le client.

Un autre onglet vous permet de visualiser des statistiques détaillées sur les utilisateurs,

Application Comments

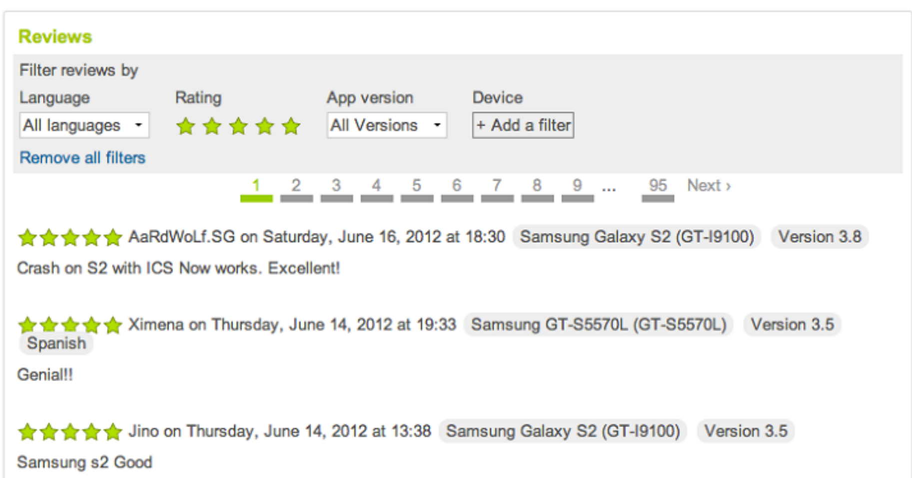


FIGURE 29.9 – Des utilisateurs ont laissé des commentaires sur une application

la version de votre application qu'ils utilisent et leur terminal, comme le montre la figure 29.10.

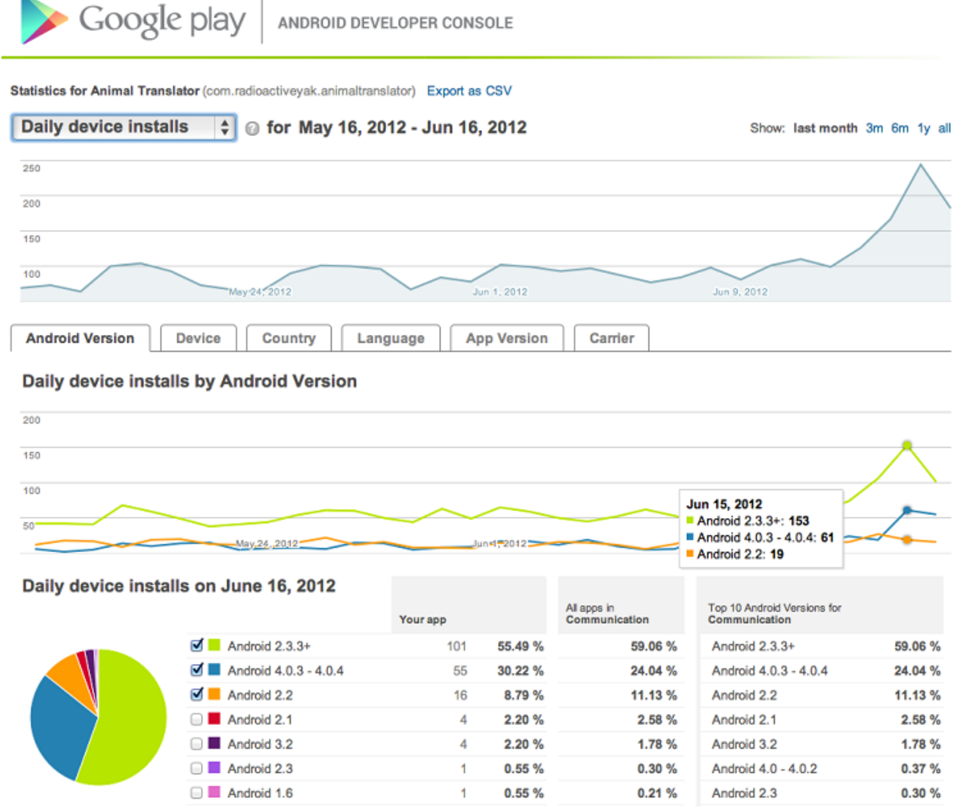


FIGURE 29.10 – Il est possible d'avoir des statistiques détaillées

Ces informations vous permettent de déterminer les tendances, de manière à anticiper à quelles périodes faire des soldes ou des annonces. Une utilisation intéressante serait de regarder quels sont les pays les plus intéressés par votre projet afin de faire des efforts de traduction. Il est aussi possible d'exporter les données afin de les exploiter même hors ligne.

Il existe d'autres solutions d'analyses, qui fournissent d'autres renseignements sur les manières d'utiliser vos applications, quelle activité est visitée à quelle fréquence, quel est le comportement typique d'un utilisateur, etc. Je ne citerai que Google Analytics, Mint ou Piwik, qui sont gratuits et puissants.

De plus, il existe un service qui récolte les erreurs et plantages que rencontrent vos utilisateurs afin que vous puissiez facilement y avoir accès et les corriger. Corriger des erreurs augmente le taux de satisfaction des utilisateurs et par conséquent le succès de

votre application.

Enfin, vous pouvez demander à ce que vos applications soient incluses dans les publicités sur AdMob, mais attention, il s'agit bien entendu d'un service payant.

Les autres types de distribution

Les autres marchés d'applications

Il existe d'autres marchés d'applications qui vous permettent de mettre vos application à disposition, par exemple AndroidPit, l'Appstore d'Amazon ou encore AppsLib qui est lui plutôt destiné aux applications pour tablettes. Je ne vais pas les détailler, ils ont chacun leurs pratiques et leurs services, à vous de les découvrir.

Distribuer par e-mail

Cela semble un peu fou, mais Google a tout à fait anticipé ce cas de figure en incluant un module qui détecte si un e-mail contient un APK en fichier joint. Le problème, c'est qu'il faut quand même que l'utilisateur accepte les applications qui proviennent de sources inconnues, ce qui est assez contraignant.

Enfin, le problème ici est que la distribution par e-mail n'est pratique que pour un public très restreint et qu'il est très facile de pirater une application de cette manière. En fait, je vous conseille de n'utiliser la distribution par e-mail que pour des personnes en qui vous avez confiance.

Sur votre propre site

Solution qui permet de toucher un plus large public : il vous suffit de mettre l'APK de votre application à disposition sur votre site, gratuitement ou contre une certaine somme, et l'utilisateur pourra l'installer. Cette méthode souffre des mêmes défauts que la distribution par e-mail, puisque l'utilisateur devra accepter les applications provenant de sources inconnues et que le risque de piratage est toujours aussi élevé.

Rentabilisez votre application

Il existe au moins quatre façons de vous faire de l'argent en exploitant les solutions proposées par Google. La première question à vous poser est : « quelle solution est la plus adaptée à mon application afin de la rentabiliser ? ». Il faut donc vous poser les bonnes questions afin d'obtenir les bonnes réponses, mais quelles sont ces questions ? On peut voir les choses d'une manière un peu simplifiée à l'aide du schéma proposé à la figure 29.11.

L'une des plus grandes décisions à prendre est de savoir si vous allez continuer à ajouter

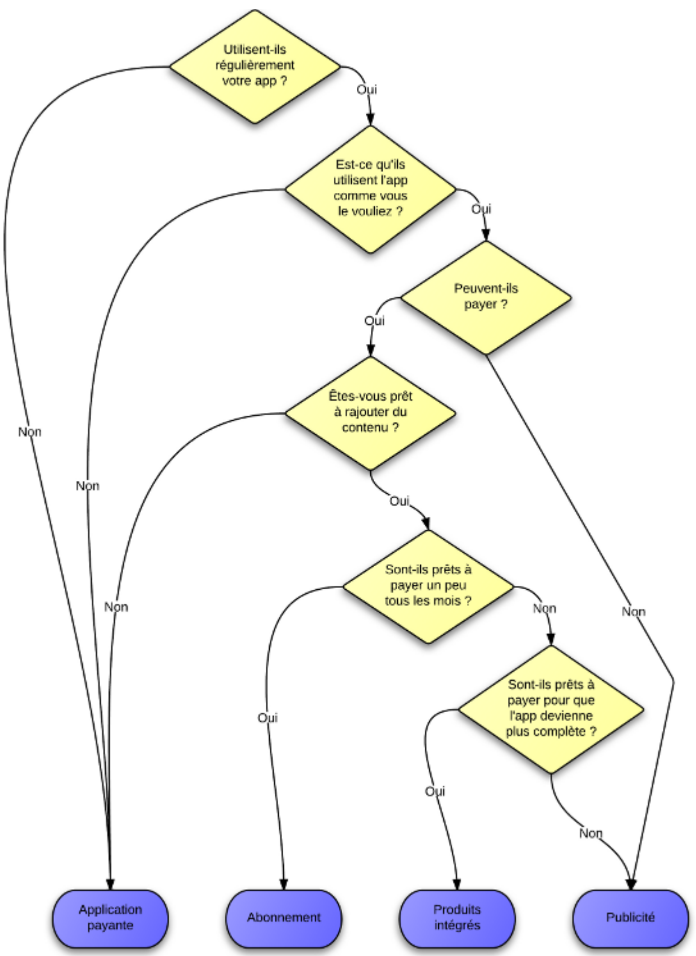


FIGURE 29.11 – « ils » réfère aux utilisateurs bien entendu

du contenu à l'application. Si non, alors faites payer une fois l'utilisateur tout en continuant les mises à jour. Si oui, alors il existe trois façons d'envisager une rémunération.

Le guide suivant suppose que vous avez distribué votre application sous Google Play. La première chose à faire est de créer un compte marchand pour Google Checkout de manière à pouvoir recevoir des revenus de la part de Google.

Créer un compte marchand pour Google Checkout

Si vous voulez faire de l'argent avec les moyens que met à disposition Google, alors vous devrez tout d'abord vous créer un compte Google marchand. Pour cela, il faut cliquer sur le lien `Ouvrir un compte marchand` dans la console de votre compte développeur Android. Remplissez bien toutes les informations, il s'agit d'une affaire de légalité. Quand on vous demandera votre raison sociale, indiquez si vous êtes un particulier, une association ou une entreprise. L'inscription est très rapide et se fait sur un écran. Il faut ensuite associer un compte bancaire à votre compte afin que Google puisse vous transmettre les paiements.

Faire payer l'application

Le moyen le plus simple est de faire en sorte que les utilisateurs payent afin de télécharger votre application sur Google Play. L'un des principaux avantages des applications payantes est qu'elles permettent de se débarrasser des publicités qui encombrent beaucoup d'applications.

Une question qui reviendra souvent est de savoir si les gens seraient prêts à payer pour les fonctionnalités que met à leur disposition votre application. Un moyen simple de vérifier est de regarder ce que font vos concurrents sur le Store. S'ils font payer pour un contenu similaire ou de qualité inférieure, alors pourquoi pas vous ?

Vient ensuite la question du prix. Encore une fois, c'est le marché qui va déterminer le meilleur prix pour votre application. Pour la majorité des applications, on parle de « biens typiques », c'est-à-dire que la demande des consommateurs diminue quand le prix augmente. En revanche, pour certaines autres applications, on parle plutôt de « biens atypiques », c'est-à-dire que la demande augmente quand le prix augmente (dans une certaine proportion, bien entendu). C'est le cas des applications pour lesquelles les utilisateurs souhaitent s'assurer de la qualité, et pour lesquelles ils évaluent la qualité du produit en fonction de son tarif. C'est un raisonnement très courant, plus un produit est cher, plus on pense qu'il est de qualité. D'ailleurs, si vous mettez à disposition plusieurs versions de votre projet, il y a des chances pour que la version ayant le plus de qualités soit aussi la plus chère.

Vous pouvez aussi envisager d'avoir deux versions de votre application, une gratuite et une payante, la première servant de fonction d'évaluation. Si l'application plaît à un utilisateur, il pourrait acheter la version complète pour pouvoir exploiter toutes ses fonctionnalités.

Sachez que 30% des revenus seront reversés à Google.

Attention cependant, le piratage des applications Android est un fléau puisqu'il est très facile à réaliser. Une technique pour éviter de perdre de l'argent à cause du piratage serait de créer un certificat pour l'utilisateur sur cette machine et de faire vérifier à un serveur distant si ce certificat est correct. S'il l'est, alors on accorde à l'utilisateur l'accès à l'application. Il y a des risques pour que les pirates aient toujours une longueur d'avance sur vous. Il vous est aussi possible de faire en sorte que votre application vérifie auprès de Google Store que l'utilisateur a bien acheté ce produit, à l'aide d'une licence.

▷ En savoir plus sur les licences
Code web : 371099

Ajouter de la publicité

Ajouter un ou plusieurs bandeaux publicitaires, voire une publicité interstitielle de temps à autre, de manière à ce qu'un annonceur vous rémunère pour chaque clic, est possible. L'avantage est que l'application reste gratuite et que les consommateurs adorent ce qui est gratuit.

Un bandeau publicitaire est un simple ruban qui se place sur les bords de votre application et qui affiche des publicités. Une publicité interstitielle prend tout l'écran et empêche l'utilisateur de continuer à utiliser votre application tant qu'elle n'a pas disparu.

Ici, je vous parlerai d'AdMob, qui est une régie qui fait le lien entre les développeurs et les annonceurs. Avec Google Play, vous pouvez être développeurs comme d'habitude, mais aussi annonceurs comme nous l'avons vu précédemment.

L'important quand on développe une application avec des publicités, c'est de penser à l'interface graphique en incluant cette publicité. Il faut lui réserver des emplacements cohérents, sinon le résultat n'est pas vraiment bon.

Il existe également un lien et un temps pour les publicités. Faire surgir des publicités en plein milieu d'un niveau risque d'en énerver plus d'un, alors qu'à la fin d'un niveau sur l'écran des scores, pourquoi pas ?

Dernière chose, essayez de faire en sorte que l'utilisateur clique intentionnellement sur vos pubs. Si c'est accidentel ou caché, il risque d'être vraiment énervé et de vous laisser une mauvaise note. Il vaut mieux qu'un utilisateur ne clique jamais sur une publicité plutôt qu'il clique une fois dessus par mégarde et supprime votre application en pestant dans les commentaires. En plus, le système réagira si vous obligez vos utilisateurs à cliquer sur les publicités, et la valeur d'un clic diminuera et vous serez au final moins rémunéré.

La première chose à faire est de créer un compte sur AdMob. Il y a un gros bouton

pour cela sur la page d'accueil. Encore une fois, l'inscription est simple puisqu'il suffit d'entrer vos coordonnées personnelles. Notez juste qu'on vous demande si vous êtes un éditeur ou un annonceur. Un éditeur est quelqu'un qui intégrera les publicités dans son produit, alors qu'un annonceur veut qu'on fasse de la publicité pour son produit.

Une fois votre compte validé, on vous demandera si vous souhaitez commencer à faire de la publicité ou monétiser vos applications pour mobile. Je ne vais bien sûr présenter que la seconde option. On vous demandera encore des informations, remplissez-les (vous pouvez voir votre numéro IBAN et le numéro SWIFT — on l'appelle parfois code BIC — sur un RIB).

Cliquez ensuite sur `Application Android` puisque c'est ce que nous faisons. Vous devrez alors décrire votre application afin qu'AdMob puisse déterminer les pubs les plus adaptées à vos utilisateurs. Enfin si vous n'aviez pas téléchargé le SDK auparavant, le site vous proposera de le faire dans la page suivante.

Au niveau technique, la première chose à faire sera d'inclure une bibliothèque dans votre projet. Pour cela, faites un clic droit sur votre projet et cliquez sur `Properties`. Ensuite, cliquez sur `Java Build Path`, puis sur l'onglet `Libraries` et enfin sur `Add External JARs...` Naviguez ensuite à l'endroit où vous avez installé le SDK AdMob pour ajouter le fichier `GoogleAdMobAdsSdk-6.0.1.jar`.

Il nous faut ensuite ajouter une activité dans notre Manifest, qui contient ces informations :

```
1  <activity android:name="com.google.ads.AdActivity"
2            android:configChanges="keyboard|keyboardHidden|
                 orientation|screenLayout|uiMode|screenSize|
                 smallestScreenSize"/>
```

Ensuite, vous aurez besoin d'au moins deux permissions pour votre application : une pour accéder à internet et une autre pour connaître l'état du réseau :

```
1  <uses-permission android:name="android.permission.INTERNET"/>
2  <uses-permission android:name="android.permission.
       ACCESS_NETWORK_STATE"/>
```

Et voilà, il vous suffit maintenant d'ajouter la vue qui contiendra la pub, c'est-à-dire l'`AdView` en XML ou en Java.

En XML, il faut rajouter le namespace suivant afin de pouvoir utiliser les attributs particuliers de la vue :
`xmlns:ads="http://schemas.android.com/apk/lib/com.google.ads"` Vous aurez besoin de :

- Préciser un format pour la publicité à l'aide de `ads:adSize`, par exemple `ads:adSize="BANNER"` pour insérer une bannière.
- Spécifier votre référence éditeur AdMob à l'aide de `ads:adUnitId`.
- Déclarer si vous souhaitez que la publicité se charge maintenant ou plus tard avec `ads:loadAdOnCreate`.

En phase de tests, vous risquez de vous faire désactiver votre compte si vous cliquez sur les publicités puisque vos clics fausseraient les résultats. Pour demander des publicités de test, utilisez l'attribut XML `ads:testDevices` et donnez-lui l'identifiant unique de votre téléphone de test. L'identifiant unique de votre téléphone vous sera donné par le SDK dans le Logcat.

Voici un exemple bien complet :

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:ads="http://schemas.android.com/apk/lib/com.google.ads"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">

  <com.google.ads.AdView android:id="@+id/adView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    ads:adUnitId="VOTRE_ID_EDITEUR"
    ads:adSize="BANNER"
    ads:testDevices="TEST_EMULATOR, ID_DE_VOTRE_APPAREIL"
    ads:loadAdOnCreate="true"/>
</LinearLayout>
```

En Java, il est possible de recharger une publicité avec la méthode `void loadAd(AdRequest)`. Il vous est aussi possible de personnaliser les couleurs de vos bannières à l'aide d'`extras`, comme pour les intents :

```
Map<String, Object> extras = new HashMap<String, Object>();
// Couleur de l'arrière-plan
extras.put("color_bg", "ABCDEF");
// Couleur du dégradé de l'arrière-plan (à partir du plafond)
extras.put("color_bg_top", "000000");
// Couleur des contours
extras.put("color_border", "FF0123");
// Couleur des liens
extras.put("color_link", "ABCCCC");
// Couleur du texte
extras.put("color_text", "FFFFFF");
// Couleur de l'URL
extras.put("color_url", "CCCCCC");

AdRequest adRequest = new AdRequest();
adRequest.setExtras(extras);
adView.loadAd(adRequest);
//On aurait aussi pu mettre adView.loadAd(new AdRequest()) si on ne voulait pas d'une publicité personnalisée
```

Il existe d'autres personnalisations possibles pour un `AdRequest`, dont le sexe de l'utilisateur (`setGender(AdRequest.Gender.MALE)` pour un homme et `setGender(AdRequest.`

`Gender.FEMALE)` pour une femme), sa date d'anniversaire (attention, au format US : par exemple, pour quelqu'un né le 25/07/1989, on aura `setBirthday("19890725")`) ou la localisation géographique de l'utilisateur avec la méthode `void setLocation(Location location)`.

De plus, si vous faites implémenter l'interface `AdListener`, vous pourrez exploiter cinq fonctions de *callback* :

- La méthode `void onReceiveAd(Ad ad)` se déclenche dès qu'une publicité est reçue correctement.
- La méthode `void onFailedToReceiveAd(Ad ad, AdRequest.ErrorCode error)` se déclenche dès qu'une pub n'a pas été reçue correctement.
- La méthode `void onPresentScreen(Ad ad)` est déclenchée quand le clic sur une publicité affiche une activité dédiée à la publicité en plein écran.
- La méthode `void onDismissScreen(Ad ad)` est déclenchée dès que l'utilisateur quitte l'activité lancée par `onPresentScreen`.
- Enfin, la méthode `void onLeaveApplication(Ad ad)` est déclenchée dès que l'utilisateur clique sur une publicité et qu'il quitte l'application.

Enfin, vous pouvez aussi insérer une publicité interstitielle avec l'objet `InterstitielAd`, qui s'utilise comme un `AdView`.

Freemium : abonnement ou vente de produits intégrés

Cette technique suppose que l'application est gratuite et exploite l'*In-App Billing*. Il existe deux types de ventes en freemium :

- L'abonnement, où les prélèvements d'argent se font à intervalles réguliers.
- La vente de produits intégrés, où le prélèvement ne se fait qu'une fois. Elle permet l'achat de contenu virtuel, comme par exemple l'achat d'armes pour un jeu, de véhicules, ou autres mises à jour de contenu.

L'avantage de l'In-App Billing, c'est qu'il exploite les mêmes fonctions que le Play Store et que par conséquent ce n'est pas votre application qui gère la transaction, mais bien Google. Le désavantage, c'est que Google garde 30% des revenus. L'In-App Billing est en fait une API qui vous permet de vendre du contenu directement à l'intérieur de l'application.

Bien entendu, ce type de paiement n'est pas adapté à toutes les applications. Il fonctionne très bien dans les jeux, mais n'imaginez pas faire de même dans les applications professionnelles.

Un moyen de mieux vendre ce type de contenu est d'ajouter une monnaie dans le jeu, qu'il est possible de gagner naturellement en jouant, mais de manière lente, ou bien en convertissant de l'argent réel en monnaie virtuelle, ce qui est plus rapide pour l'utilisateur. Une idée intéressante à ce sujet est d'avoir une monnaie virtuelle similaire pour tous vos produits, afin que l'utilisateur soit plus enclin à acheter de la monnaie et surtout à utiliser vos autres produits.

Ce qui est important quand on vend des produits intégrés, c'est d'être sûr que l'utilisateur retrouvera ces produits s'il change de terminal. Il n'y a rien de plus frustrant

que de gâcher de l'argent parce que l'éditeur n'a pas été capable de faire en sorte que le contenu soit lié au compte de l'utilisateur. Ainsi, il existe deux types de paiement pour les produits intégrés :

- *Managed* : Google Play retient les transactions qui ont été effectuées.
- *Unmanaged* : c'est à vous de retenir le statut des transactions.

Sachez aussi qu'encore une fois 30% des revenus seront reversés à Google.

Vous trouverez plus de détails sur la documentation.

▷ Plus de détails
Code web : 347535

Chapitre 30

L'architecture d'Android

Difficulté :

Après quelques réflexions et quelques recherches, je me suis dit que c'était peut-être une bonne idée de présenter aux plus curieux l'architecture d'Android. Vous pouvez considérer ce chapitre comme facultatif s'il vous ennuie ou vous semble trop compliqué, vous serez tout de même capables de développer correctement sous Android, mais un peu de culture technique ne peut pas vous faire de mal.

Le noyau Linux

La figure 30.1 schématise l'architecture d'Android.

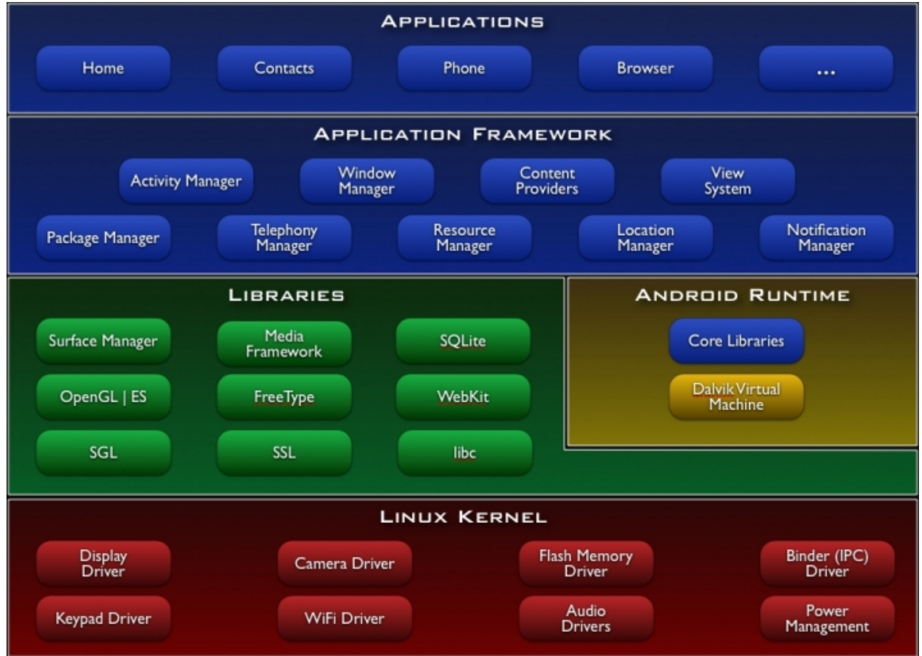


FIGURE 30.1 – L'architecture d'Android

On peut y observer toute une pile de composants qui constituent le système d'exploitation. Le sens de lecture se fait de bas en haut, puisque le composant de plus bas niveau (le plus éloigné des utilisateurs) est le noyau Linux et celui de plus haut niveau (le plus proche des utilisateurs) est constitué par les applications.

Je vous avais déjà dit que le système d'exploitation d'Android se basait sur Linux. Si on veut être plus précis, c'est le noyau (« *kernel* » en anglais) de Linux qui est utilisé. Le noyau est l'élément du système d'exploitation qui permet de faire le pont entre le matériel et le logiciel. Par exemple, les pilotes WiFi permettent de contrôler la puce WiFi. Quand Android veut activer la puce WiFi, on peut imaginer qu'il utilise la fonction « `allumerWifi()` », et c'est au constructeur de spécifier le comportement de « `allumerWifi()` » pour sa puce. On aura donc une fonction unique pour toutes les puces, mais le contenu de la fonction sera unique pour chaque matériel.

La version du noyau utilisée avec Android est une version conçue spécialement pour l'environnement mobile, avec une gestion avancée de la batterie et une gestion particulière de la mémoire. C'est cette couche qui fait en sorte qu'Android soit compatible avec tant de supports différents.

Cela ne signifie pas qu'Android est une distribution de Linux, il a le même cœur mais c'est tout. Vous ne pourrez pas lancer d'applications destinées à GNU/Linux sans passer par de petites manipulations, mais si vous êtes bricoleurs. . .

Si vous regardez attentivement le schéma, vous remarquerez que cette couche est la seule qui gère le matériel. Android en soi ne s'occupe pas de ce genre de détails. Je ne veux pas dire par là qu'il n'y a pas d'interactions entre Android et le matériel, juste que, quand un constructeur veut ajouter un matériel qui n'est pas pris en compte par défaut par Android, il doit travailler sur le *kernel* et non sur les couches au-dessus, qui sont des couches spécifiques à Android.

Le moteur d'exécution d'Android

C'est cette couche qui fait qu'Android n'est pas qu'une simple « implémentation de Linux pour portables ». Elle contient certaines bibliothèques de base du Java accompagnées de bibliothèques spécifiques à Android et la machine virtuelle « Dalvik ».

Un moteur d'exécution (« *runtime system* » en anglais) est un programme qui permet l'exécution d'autres programmes. Vous savez peut-être que pour utiliser des applications développées en Java sur votre ordinateur vous avez besoin du JRE (« Java Runtime Environment »). Eh bien, il s'agit du moteur d'exécution nécessaire pour lancer des applications écrites en Java.

La figure 30.2 est un schéma qui indique les étapes nécessaires à la compilation et à l'exécution d'un programme Java standard.

Votre code est une suite d'instructions que l'on trouve dans un fichier `.java` qui sera traduit en une autre suite d'instructions dans un autre langage que l'on appelle le « bytecode ». Ce code est contenu dans un fichier `.class`. Le bytecode est un langage spécial qu'une machine virtuelle Java peut comprendre et interpréter. Les différents fichiers `.class` sont ensuite regroupés dans un `.jar`, et c'est ce fichier qui est exécutable. En ce qui concerne Android, la procédure est différente. En fait, ce que vous appelez Java est certainement une variante particulière de Java qui s'appelle « Java SE ». Or, pour développer des applications pour Android, on n'utilise pas vraiment Java SE. Pour ceux qui savent ce qu'est « Java ME », ce n'est pas non plus ce framework que l'on utilise (Java ME est une version spéciale de Java destinée au développement mobile, mais pas pour Android donc).

À noter que sur le schéma le JDK et le JRE sont réunis, mais il est possible de télécharger le JRE sans télécharger le JDK.

La version de Java qui permet le développement Android est une version réduite amputée de certaines fonctionnalités qui n'ont rien à faire dans un environnement mobile. Par exemple, la bibliothèque graphique Swing n'est pas supportée, on trouve à la place un système beaucoup plus adapté. Mais Android n'utilise pas une machine virtuelle

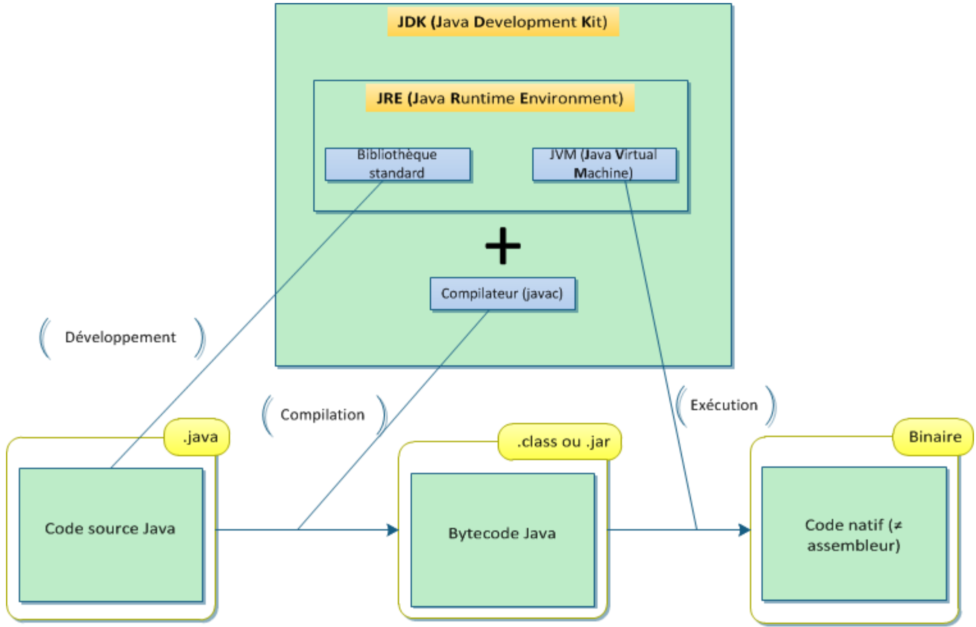


FIGURE 30.2 – Architecture Java

Java ; une machine virtuelle tout étudiée pour les systèmes embarqués a été développée, et elle s'appelle « Dalvik ». Cette machine virtuelle est optimisée pour mieux gérer les ressources physiques du système. Elle permet par exemple de laisser moins d'empreinte mémoire (la quantité de mémoire allouée à une application pendant son exécution) ou d'utiliser moins de batterie qu'une machine virtuelle Java.

La plus grosse caractéristique de Dalvik est qu'elle permet d'instancier (terme technique qui signifie « créer une occurrence de ». Par exemple, quand vous créez un objet en java, on instancie une classe puisqu'on crée une occurrence de cette classe) un nombre très important d'occurrences de lui-même : chaque programme a sa propre occurrence de Dalvik et elles peuvent vivre sans se perturber les unes les autres. La figure 30.3 est un schéma qui indique les étapes nécessaires à la compilation et à l'exécution d'un programme Android standard.

On voit bien que le code Java est ensuite converti en bytecode Java comme auparavant. Mais souvenez-vous, je vous ai dit que le bytecode Java ne pouvait être lu que par une machine virtuelle Java, mais que Dalvik n'était pas une machine virtuelle Java. Il faut donc procéder à une autre conversion à l'aide d'un programme qui s'appelle « dx » qui s'occupe de traduire les applications de bytecode Java en bytecode Dalvik, qui, lui, est compréhensible par la machine virtuelle.

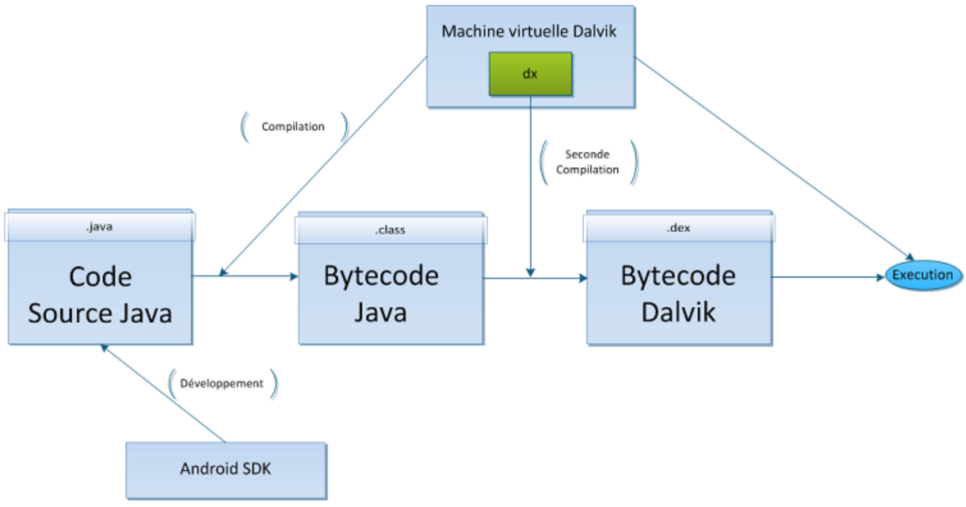


FIGURE 30.3 – Dalvik

La puissante machine virtuelle Dalvik est destinée uniquement à Android, mais il est possible de développer une machine virtuelle similaire et qui ne fonctionne pas sous Android. Par exemple, le **Nokia N9** pourra exécuter des applications Android sans utiliser ce système.

Index

A

accéléromètre . . . 501
accès au disque . . . 308
ActionBar . . . 233
activité . . . 34
`Adapter` . . . 190, 204
`AdapterView` . . . 195
ADT . . . 19
alarme . . . 398
`AlertDialog` . . . 214
Android . . . 4
animation . . . 154
ANR . . . 357
APK . . . 538
application
- freemium . . . 558
- payante . . . 554
- publication . . . 537
- publicité . . . 555
- signature . . . 541

AppWidget . . . 417
architecture . . . 561
`ArrayAdapter` . . . 191
`AsyncTask` . . . 369
`AudioRecord` . . . 485
`AutoCompleteTextView` . . . 221
AVD . . . 21

B

baromètre . . . 504
base de données . . . 339
- CRUD . . . 347
- DAO . . . 346

`Bitmap` . . . 446
boîte de dialogue . . . 208
`Bundle` . . . 268
`Button` . . . 92
bytecode . . . 563

C

`Calendar` . . . 215
calque . . . 464
`Camera` . . . 487
`Canvas` . . . 446
capteur . . . 495
carte . . . 460
chaîne de caractères . . . 145
`CheckBox` . . . 92
compilation . . . 563
`ConnectivityManager` . . . 436
`ContentProvider` . . . 403
`ContentResolver` . . . 403
`context` . . . 35
`ContextMenu` . . . 231
curseur . . . 351
cycle d'une activité . . . 37, 264

D

Dalvik . . . 563
`DatePicker` . . . 217
débogage . . . 169
désérialiser . . . 205
`Dialog` . . . 208
documentation . . . 94
drawable . . . 149

E

Eclipse . . . 18

EditText . . . 91
émulateur . . . 21

F

fournisseur de contenu . . . 401
FrameLayout . . . 137

G

GeoPoint . . . 464
Google Maps . . . 460
Google Play . . . 544
GPS . . . 455
GridView . . . 202
gyroscope . . . 501

H

handler . . . 363
HTTP . . . 438

I

ImageView . . . 219
inflater . . . 205
Intent . . . 275
- action . . . 288
- avec retour . . . 283
- BroadcastIntent . . . 297
- catégorie . . . 291
- donnée . . . 291
- extra . . . 277
- implicite . . . 285
- PendingIntents . . . 390
- résolution . . . 288
- sécurité . . . 299

interface graphique . . . 34, 73
Internet . . . 435

J

Java . . . 7, 14
JDK . . . 14

L

layout . . . 115
LayoutInflater . . . 205
LinearLayout . . . 116
liste . . . 190
ListView . . . 196
LocationManager . . . 456
Logcat . . . 169

M

magnétomètre . . . 502
Manifest . . . 258
MapActivity . . . 462
MapView . . . 463
marqueur . . . 468
MediaPlayer . . . 481
MediaRecorder . . . 484
Menu . . . 225
menu contextuel . . . 231
MenuItem . . . 229
message . . . 363
MIME . . . 287
multimédia . . . 479
MyLocationOverlay . . . 467

N

notification . . . 392
noyau . . . 562

O

OHA . . . 4
OverlayItem . . . 468

P

Paint . . . 446
Parcelable . . . 278
permission . . . 263
PhoneStateListener . . . 473
photomètre . . . 504
Play Store . . . 544
préférences . . . 302
processus . . . 358

R

RadioButton . . . 93
RadioGroup . . . 93
référence faible . . . 371
RelativeLayout . . . 125
ressource . . . 51

S

ScrollView . . . 138
SDK . . . 15
Sensor . . . 496

SensorEventListener 499
SensorManager 498
service 379
  IntentService 388
  premier plan 396
SharedPreference 302
SimpleAdapter 192
SMS 476
SmsManager 477
SparseBooleanArray 198
Spinner 202
SQLiteDatabase 342
SQLiteOpenHelper 342
style 153
SurfaceView 449

T

TableLayout 132
téléphonie 472
TelephonyManager 472
TextView 90
thermomètre 504
thread 358
TimePicker 217

U

URI 286

V

VideoView 483
ViewHolder 207
vue 39, 78

W

WeakReference 372
widget 89
  évènement 101

X

XML 52

Notes

Notes

Dépôt légal : décembre 2012
ISBN : 979-10-90085-37-4
Code éditeur : 979-10-90085
Imprimé en France

Achevé d'imprimer le 19 décembre 2012
sur les presses de Corlet Imprimeur (Condé-sur-Noireau)
Numéro imprimeur : 151216

Mentions légales :
Conception couverture : Fan Jiyong
Illustrations chapitres : Fan Jiyong

CRÉEZ DES APPLICATIONS POUR ANDROID

Vous aimeriez apprendre à créer des applications pour Android mais ne savez pas par où commencer ? Ce livre est fait pour vous ! **Conçu pour les débutants**, il vous apprendra pas à pas **comment développer des applications pour Android.**

30 chapitres de difficulté progressive
3 travaux pratiques pour vous exercer
Un livre **entièrement en couleur**

Débutant en développement Android ?

- ► Le seul prérequis est de connaître le Java
- ► De nombreux exemples commentés et expliqués
- ► Une pédagogie pensée pour ne perdre aucun lecteur

Réalisez les applications dont vous avez toujours rêvé !

- ► De quoi avez-vous besoin pour créer des applications Android ?
- ► Installez les outils nécessaires et configurez votre environnement de développement
- ► Embellissez vos applications grâce aux interfaces graphiques
- ► Tirez profit des possibilités de votre appareil : géolocalisation, multimédia, accéléromètre…
- ► Pratiquez grâce aux TP et créez un bloc-notes, un explorateur de fichier ou encore un jeu de labyrinthe
- ► Publiez vos applications sur le Play Store

À qui ce livre est-il destiné ?

- ► Aux passionnés qui veulent approfondir leurs connaissances en informatique
- ► Aux étudiants dans le domaine des nouvelles technologies qui recherchent un support de cours
- ► À tous les développeurs Java qui ont besoin de se former ou de se convertir au développement d'applications Android

À propos de l'auteur

Frédéric Espiau

Ayant toujours été passionné d'informatique, c'est assez naturellement que Frédéric s'oriente vers des études dans ce domaine, et finit par intégrer la branche Génie Informatique de l'Université de Technologie de Compiègne.
Son premier téléphone portable sous Android est une révélation, il décide alors de s'investir pour maîtriser complètement ce système novateur. Son fort désir d'écrire et d'enseigner un domaine qu'il affectionne très particulièrement l'a poussé à rédiger ce livre, espérant faire découvrir un outil d'aujourd'hui et de demain.

Ce livre est issu du Site du Zéro

Retrouvez dans ce livre les cours du Site du Zéro dans une édition revue et corrigée.

Téléchargez les codes source en ligne grâce aux « codes web » inclus dans ce livre.

ISBN : 979-10-90085-37-4

Prix public : 29 € TTC

www.siteduzero.com