TUTORIEL

www.openclassrooms.com

Créez des applications pour Android

SIGMATHS

Tutoriel : Créez des applications pour Android

Table des matières

Créez des applications pour Android

Bonjour à tous et bienvenue dans le monde merveilleux du développement d'applications Android !

Bugdroid, la mascotte d'Android

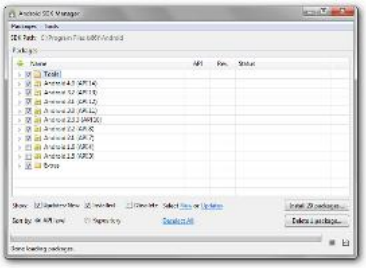
Avec l'explosion des ventes de smartphones ces dernières années, Android a pris une place importante dans la vie quotidienne. Ce système d'exploitation permet d'installer des applications de toutes sortes : jeux, bureautique, multimédia, etc. Que diriez-vous de développer vos propres applications pour Android, en les proposant au monde entier via le *Play Store*, le marché d'applications de Google ? Eh bien figurez-vous que c'est justement le but de ce cours : vous apprendre à créer des applications pour Android !

Cependant, pour suivre ce cours, il vous faudra quelques connaissances :

- Les applications Android étant presque essentiellement codées en **Java**, il vous faut connaître ce langage. Heureusement, le Site du Zéro propose un cours (http://www.siteduzero.com/tutoriel-3-10601-apprenez-a-programmer-en-java.html) et même un livre (http://www.siteduzero.com/boutique-614-430-apprenez-a-programmer-en-java.html) sur le Java.
- Connaître un minimum de **SQL** pour les requêtes (ça tombe bien, le Site du Zéro propose un cours sur MySQL (http://www.siteduzero.com/tutoriel-3-464494-administrez-vos-bases-de-donnees-avec-mysql.html)). Si vous ne connaissez absolument rien en SQL, vous pourrez tout de même suivre le cours dans son intégralité, mais constituer votre propre base de données sans théorie me semble risqué.
- Et enfin, être un minimum autonome en informatique : vous devez par exemple être capables d'installer Eclipse tout seul (vous voyez, je ne vous demande pas la lune).

Rien de bien méchant, comme vous pouvez le voir. Mais le développement pour Android est déjà assez complet comme cela, ce serait bien trop long de revenir sur ces bases-là. Ce cours débutera cependant en douceur et vous présentera d'abord les bases essentielles pour le développement Android afin que vous puissiez effectuer des applications simples et compatibles avec la majorité des terminaux. Puis nous verrons tout ce que vous avez besoin de savoir afin de créer de belles interfaces graphiques ; et enfin on abordera des notions plus avancées afin d'exploiter les multiples facettes que présente Android, dont les différentes bibliothèques de fonctions permettant de mettre à profit les capacités matérielles des appareils.

À la fin de ce cours, vous serez capables de réaliser des jeux, des applications de géolocalisation, un navigateur Web, des applications sociales, et j'en passe. En fait, le seul frein sera votre imagination !



L'univers Android

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 de l'année 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.

Le logo de l'OHA, une organisation qui cherche à développer des standards open source pour les appareils mobiles

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 ! :D

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 :

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

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 :

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

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'e
xécuter
float changer_vitesse(float facteur_de_vitesse, boolean acceleration)
  // S'il s'agit d'une accelération
  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.
  abstract private void decelerer();
}
```

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`.

```
//Interface des objets qui peuvent voler
interface PeutVoler {
  void décoller();
}

class Avion extends Vehicule implements PeutVoler {
  //Implémenter toutes les méthodes de PeutVoler et les méthodes abstraites de Vehicule
}
```

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.

Installation et configuration des outils

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.

Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

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 (http://www.siteduzero.com/tutoriel-3-489522-1-l-architecture-d-android.html#ss_part_4).

Rendez-vous ici (http://www.oracle.com/technetwork/java/javase/downloads/index.html) 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 suivante.

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.

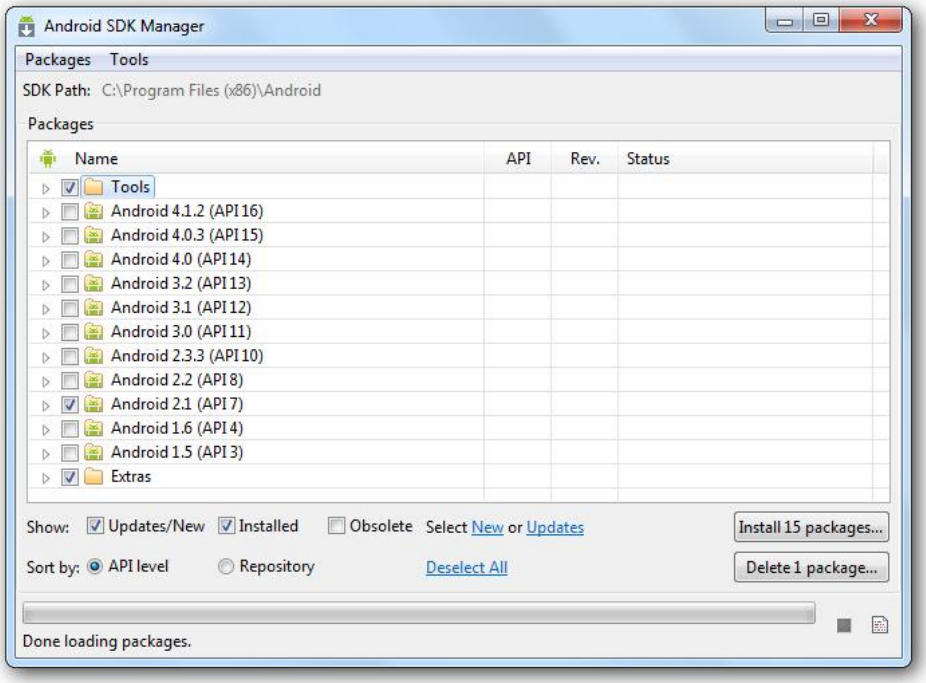
Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

Le SDK d'Android

> *C'est quoi un SDK?*

Un SDK, 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 ici (http://developer.android.com/sdk/index.html) et cliquez sur `USE AN EXISTING IDE` puis sur `Download the SDK Tools`. Au premier lancement du SDK, un écran semblable à la figure suivante s'affichera.



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.

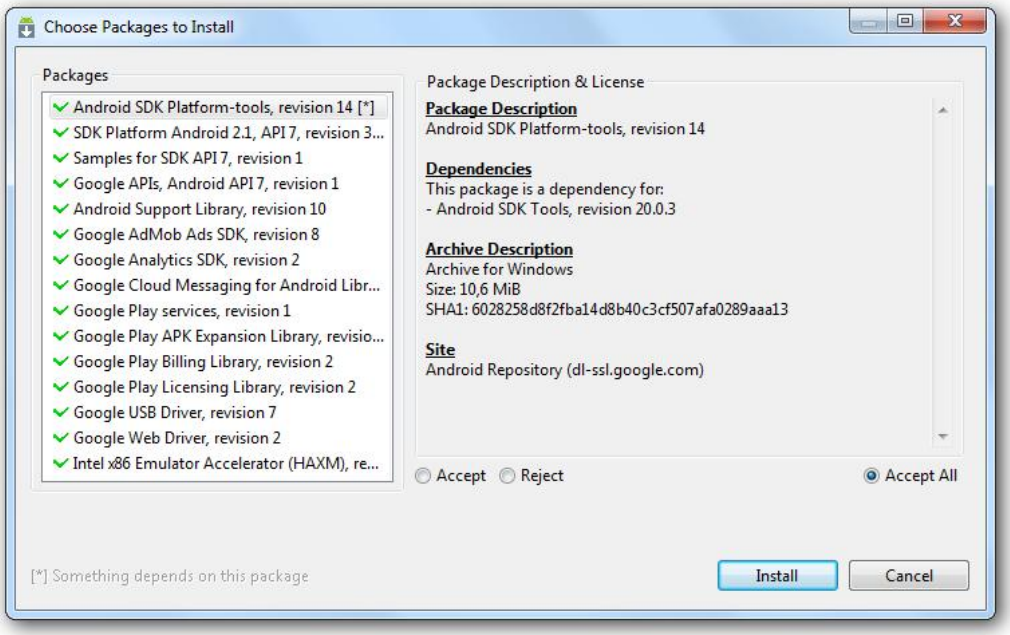
> *À quoi servent les autres paquets ?*

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 suivante.



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

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.

Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

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 tutoriel de me baser sur Eclipse : tout simplement parce qu'il est gratuit, puissant et recommandé par Google dans la documentation officielle d'Android (http://developer.android.com/sdk/requirements.html). Vous pouvez aussi opter pour d'autres IDE compétents tels que IntelliJ IDEA (http://www.jetbrains.com/idea/), NetBeans (http://fr.netbeans.org/) avec une extension (http://kenai.com/projects/nbandroid/) ou encore MoSync (http://www.mosync.com/).

Le tutoriel 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.

Cliquez ici (http://www.eclipse.org/downloads/) pour choisir une version d'Eclipse 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).

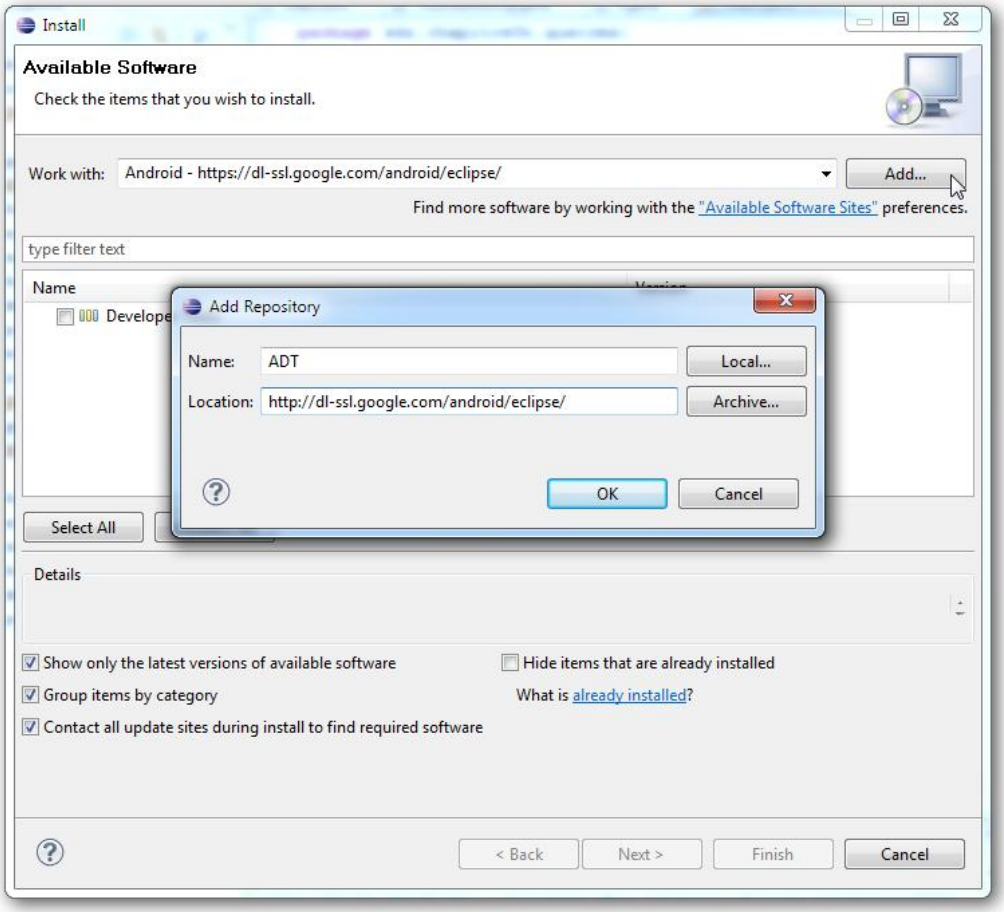
Il vous faudra **110 Mo** sur le disque pour installer la version d'Eclipse que j'ai choisie.

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 (http://developer.motorola.com/docstools/motodevstudio/) 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 suivante. 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`.



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

Si cette manipulation ne fonctionne pas, essayez avec cette adresse suivante : `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 (http://developer.android.com/sdk/installing/installing-adt.html#Troubleshooting), 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 suivante.

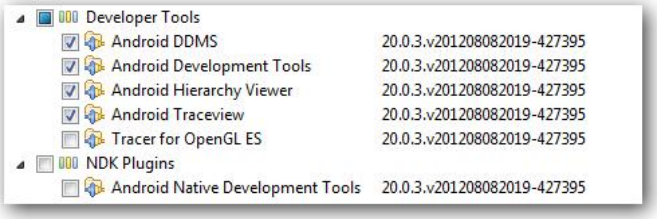
Package	Size	MD5 Checksum
ADT-20.0.3.zip	12390954 bytes	869a536b1c56d0cd920ed9ae259ae619

Téléchargez l'archive

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 suivante.



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 suivante.

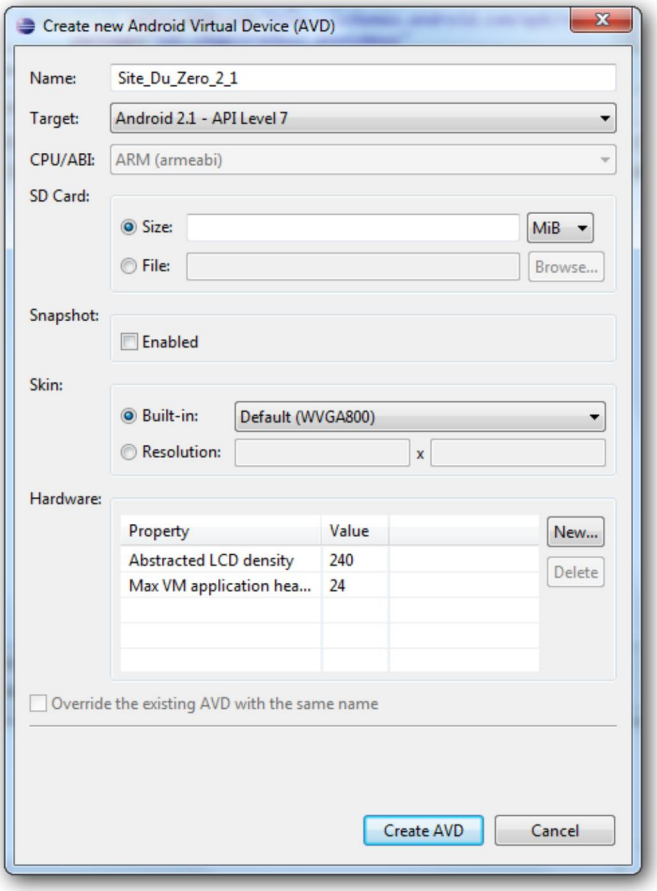
La barre d'outils d'Eclipse

Vous voyez le couple d'icônes représenté à la figure suivante ? 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.

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. :pirate:

Une fenêtre s'ouvre (voir figure suivante), 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 !



Créez votre propre émulateur

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 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 suivante.

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
```

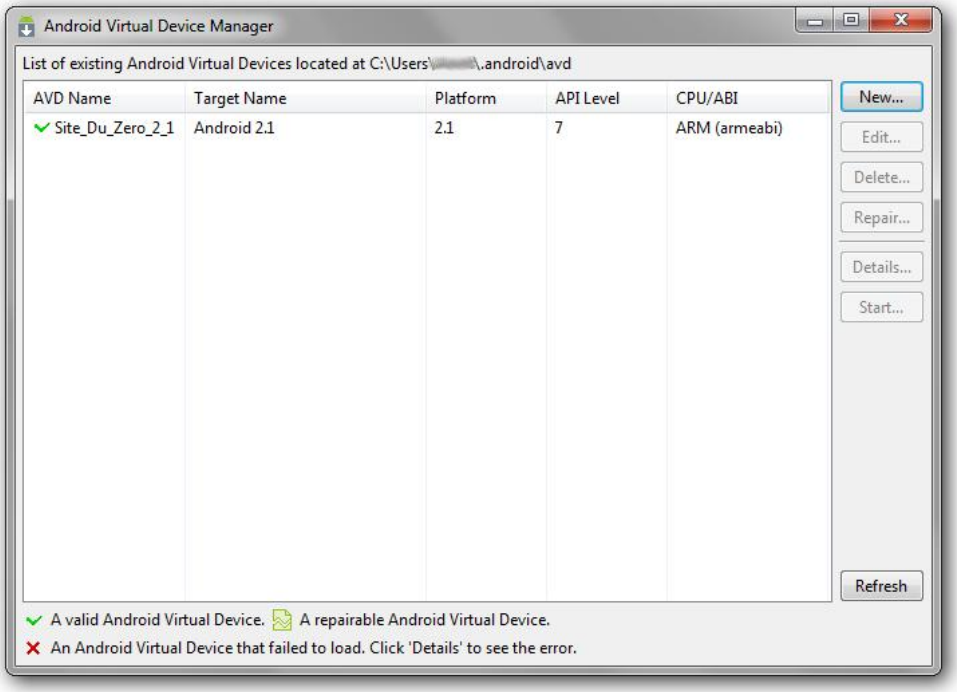
en

```
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 suivante.

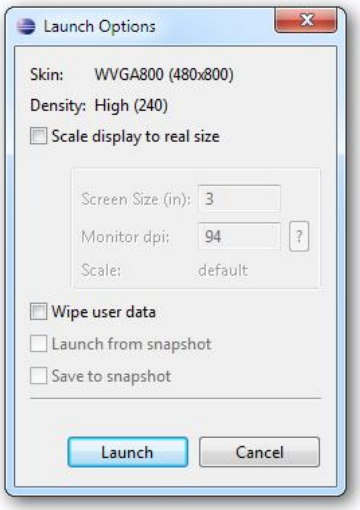


La liste des émulateurs que connaît votre AVD Manager

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.
- `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 suivante.



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 (de moins en moins utilisé).

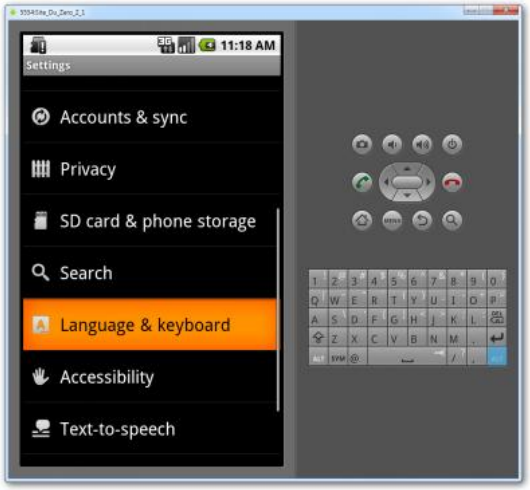
> *Mais ! L'émulateur n'est pas à l'heure ! En plus c'est de l'anglais !*

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 suivante.

Le menu est ouvert

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 suivante) ; 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 ont beaucoup d'imagination ;)).



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 suivante.

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 cette page (http://developer.android.com/tools/extras/oem-usb.html#Drivers).

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 (http://developer.android.com/guide/developing/device.html#VendorIds). 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/ru
les.d/51-android.rules
```

Si cette configuration ne vous correspond pas, je vous invite à lire la documentation de `udev` (http://www.reactivated.net/writing_udev_rules.html) afin de créer votre propre règle.

Et après ?

Ben rien ! :p 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 !).

- 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.

Votre première application

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 ?

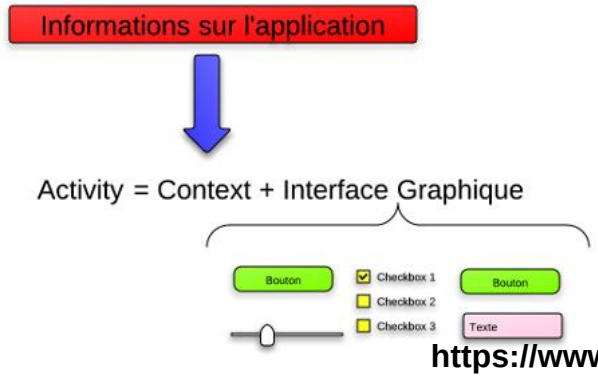
Si vous observez un peu l'architecture de la majorité des applications Android, vous remarquerez une construction toujours à peu près similaire. Prenons par exemple l'application du Play Store. Vous avez plusieurs fenêtres à l'intérieur même de cette application : si vous effectuez une recherche, une 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 suivante illustre ce concept.

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 suivante.



> 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.

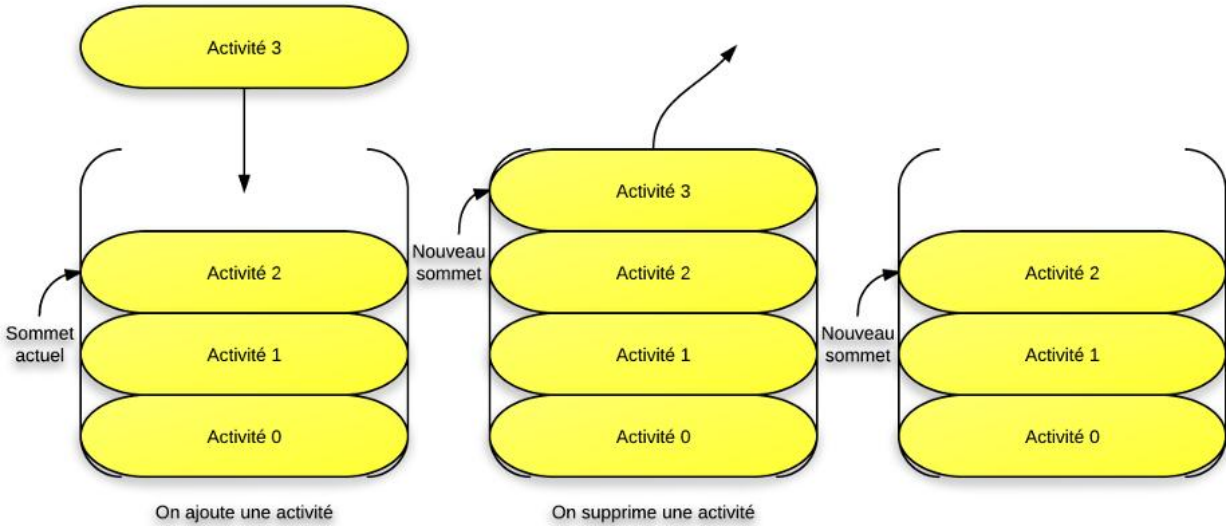
États d'une activité

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 suivante.



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 :

État	Visibilité	Description
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.

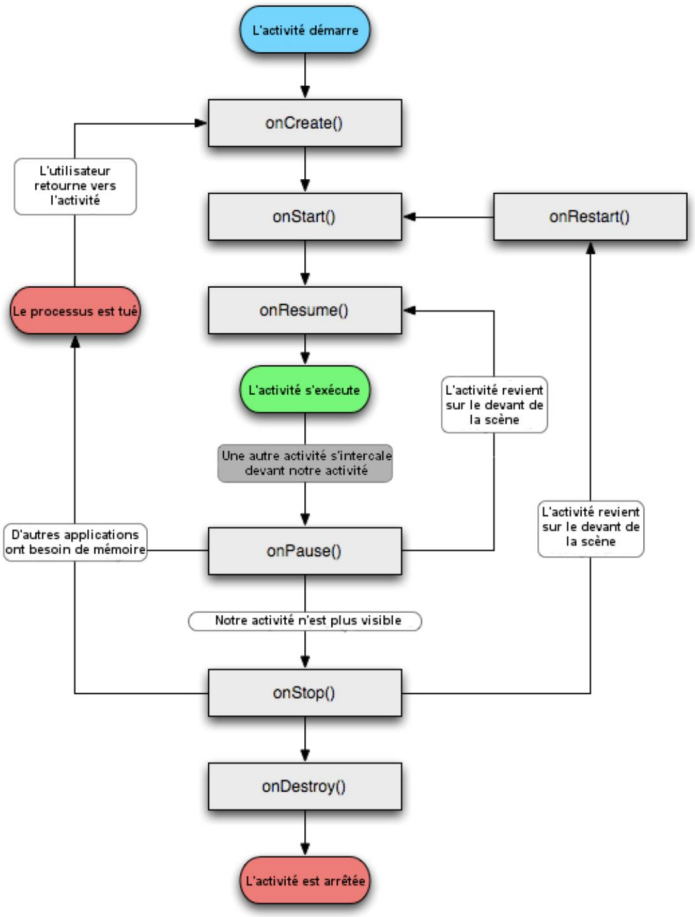
> *Mais j'ai pourtant déjà vu des systèmes Android avec deux applications visibles en même temps !*

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. Voici 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.



Cycle de vie d'une activité

Les activités héritent de la classe Activity (http://developer.android.com/reference/android/app/Activity.html). Or, la classe `Activity` hérite de l'interface Context (http://developer.android.com/reference/android/content/Context.html) 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 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 (http://www.siteduzero.com/tutoriel-3-10601-apprenez-a-programmer-en-java.html#part_10599), il est possible de disposer les vues à l'aide de conteneurs, nous verrons comment plus tard.

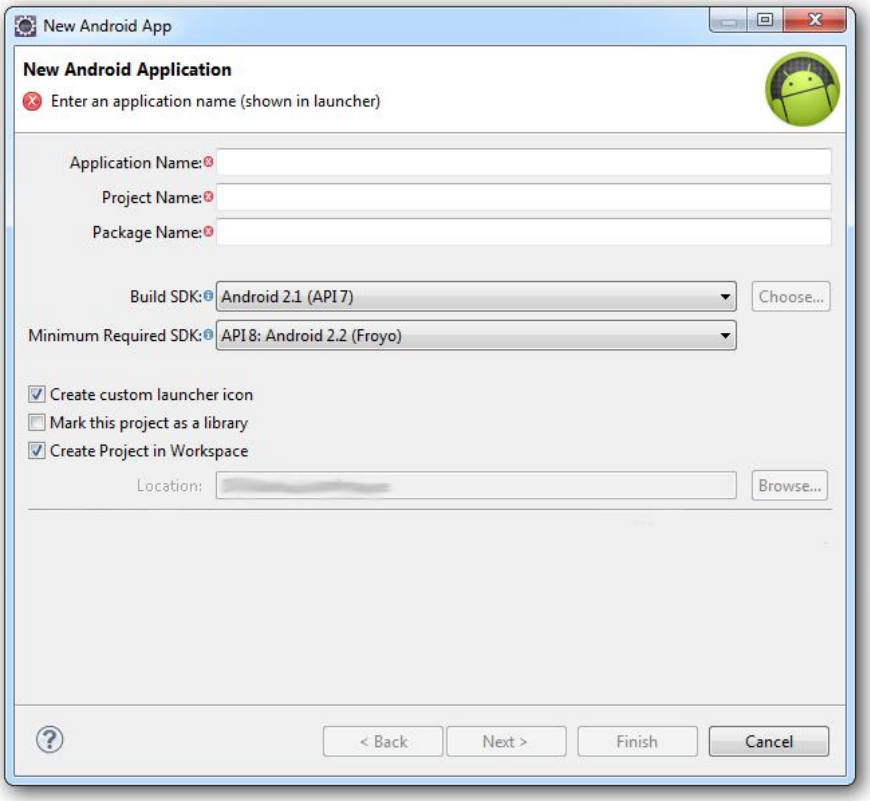
Les vues héritent de la classe `View`. On les trouve dans le package `android.view.View`.

Création d'un projet

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

Ces trois boutons permettent de gérer des projets Android

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



Création d'un nouveau projet

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 application.

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

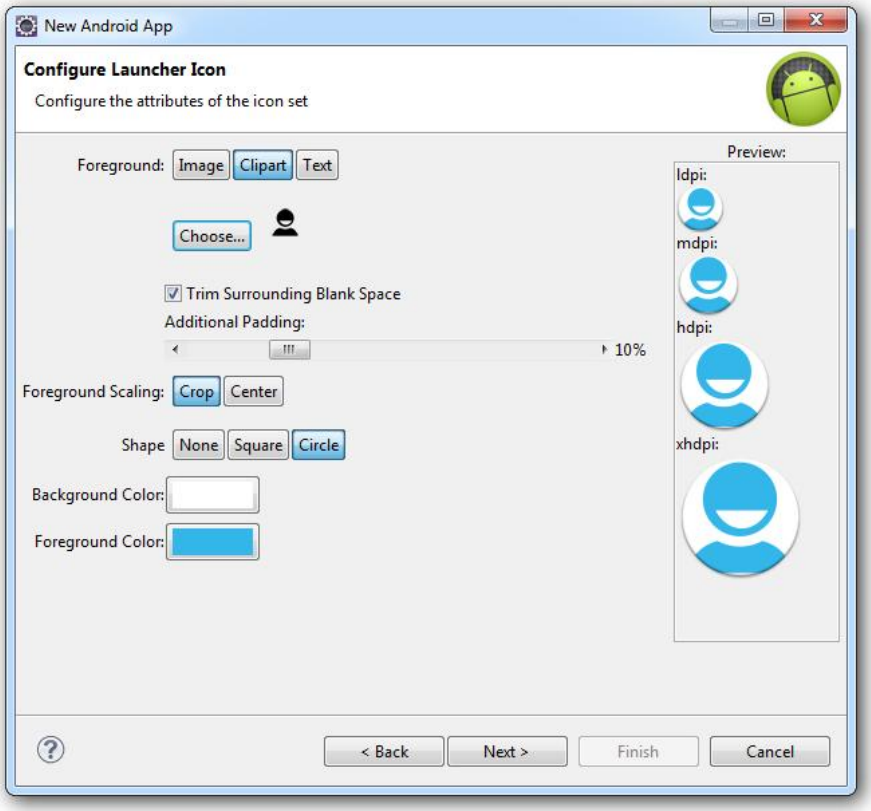
Vous vous retrouvez ensuite confronté à deux listes défilantes :

- 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 :

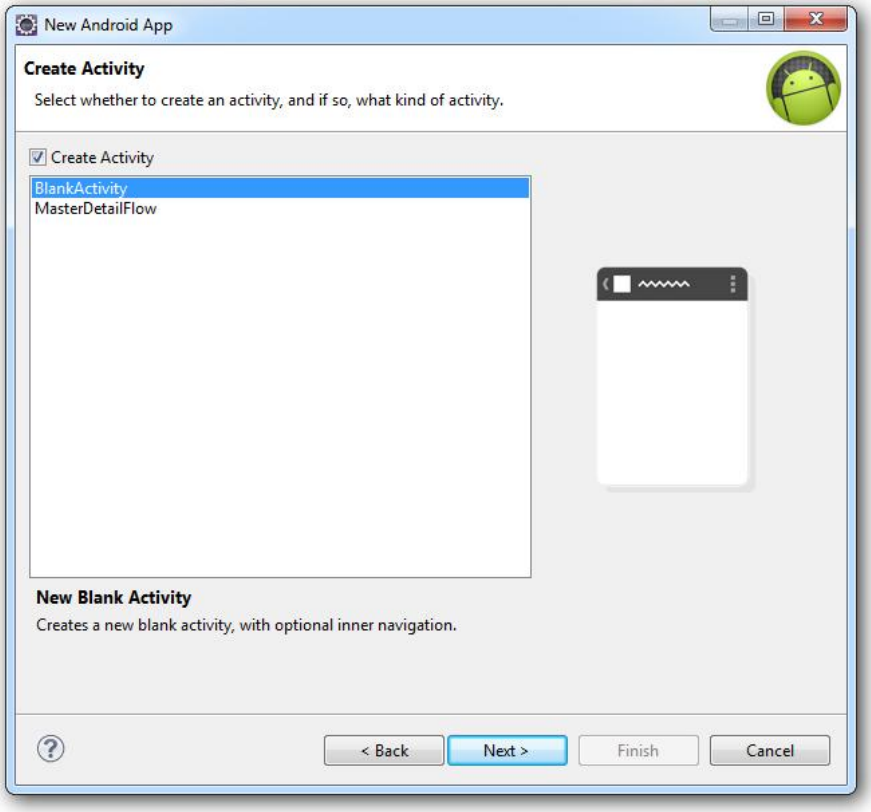
- La première, intitulée `Create custom launcher icon`, ouvrira à la fenêtre suivante un outil pour vous aider à construire une icône pour votre application à partir d'une image préexistante.
- Cochez la deuxième, `Mark this project as a library`, si votre projet est uniquement une bibliothèque de fonctions. 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 suivante qui s'affichera.



Cet outil facilite la création d'icônes

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 suivante).

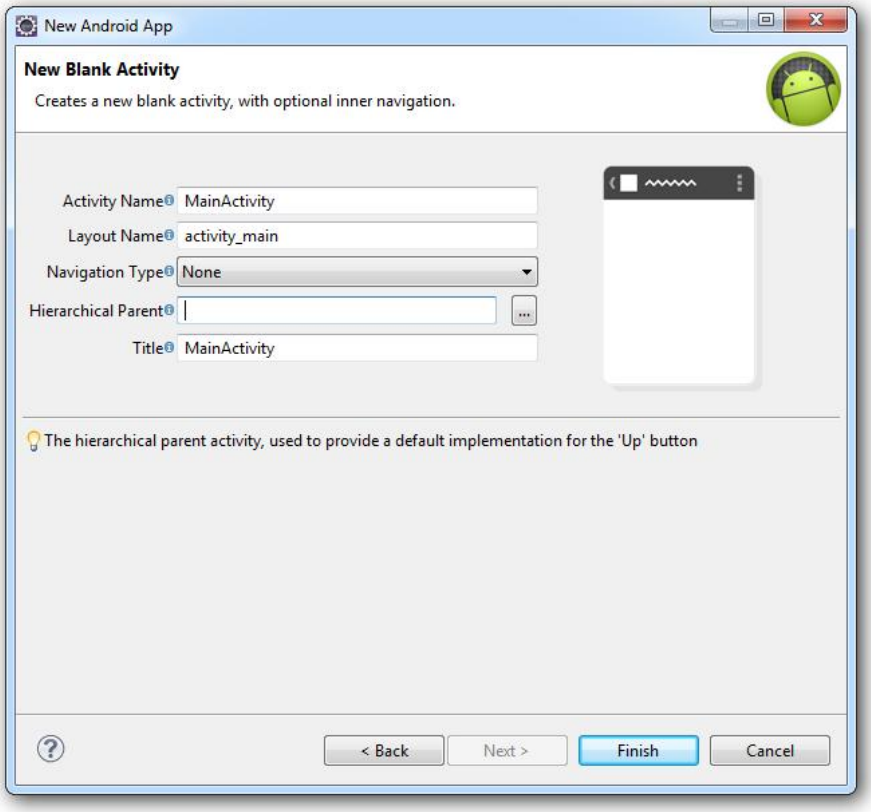


Vous pouvez ici choisir une mise en page standard

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.

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



Permet de créer une première activité facilement

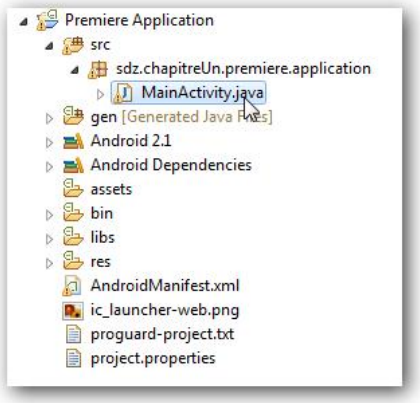
Ici encore une fois, on fait face à cinq champs à renseigner :

- `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 suivante).



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;
  }
}
```

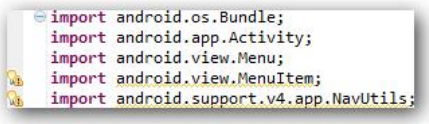
Ah ! On reconnaît certains termes que je viens tout juste d'expliquer ! Je vais prendre toutes les lignes une par une, histoire d'être certain de ne déstabiliser personne.

```
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.

```
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.MenuItem;
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 suivante.



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 !

```
public class MainActivity extends Activity {
  //…
}
```

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

```
@Override
public void onCreate(Bundle savedInstanceState) {
  //…
}
```

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.

```
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 :

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

sera expliquée dans le prochain chapitre.

En revanche, l'instruction suivante :

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

… sera expliquée bien, bien plus tard.

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

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

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

public class MainActivity extends Activity {
  private TextView coucou = null;

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

    coucou = new TextView(this);
    coucou.setText("Bonjour, vous me devez 1 000 000€.");
    setContentView(coucou);
  }

}
```

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. :D 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.

Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

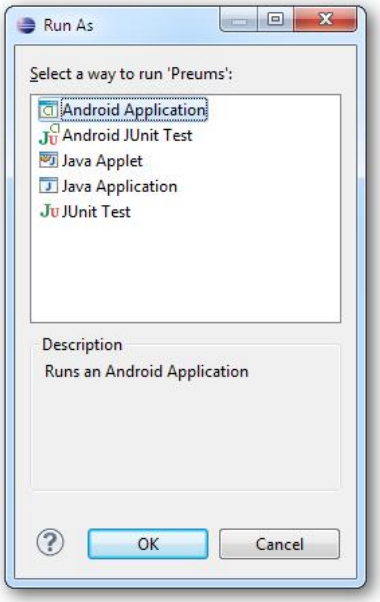
Lancement de l'application

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 looooong. :(

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

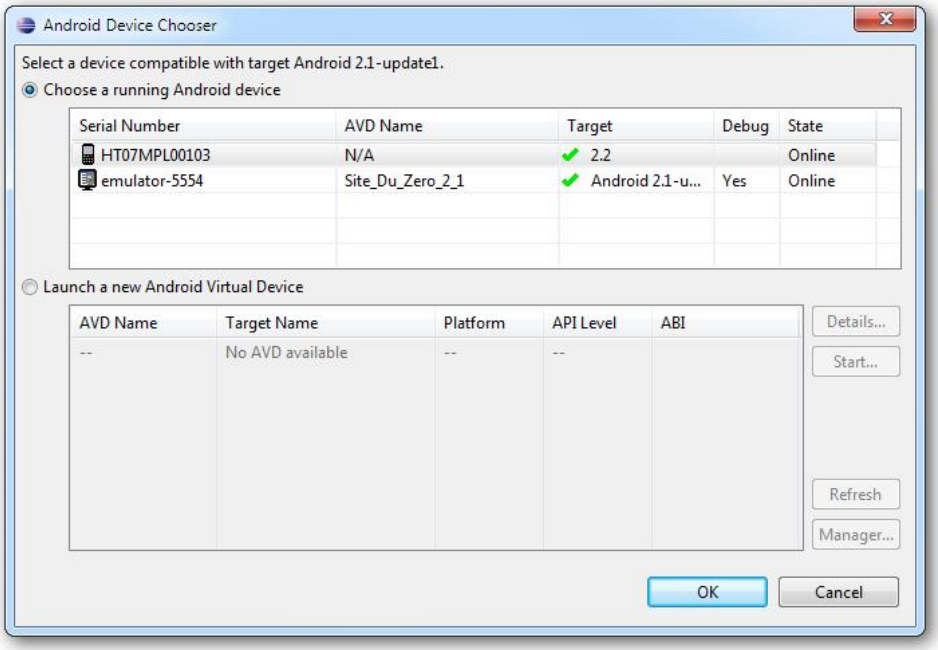
Les outils pour exécuter votre code

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



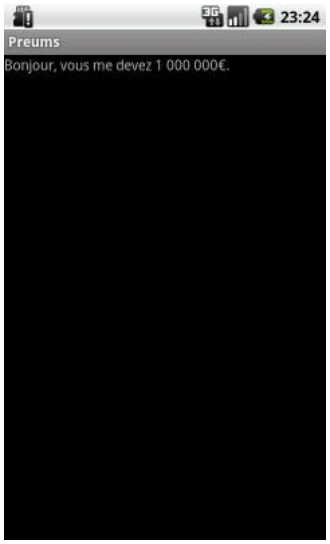
Sélectionnez « Android Application »

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



Choisissez le terminal de test

On vous demande sur quel terminal vous voulez lancer votre application. Vous pouvez valider en cliquant sur `OK`. Le résultat devrait s'afficher sur votre terminal ou dans l'émulateur (voir figure suivante). Génial ! L'utilisateur (naïf) vous doit 1 000 000 € !



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.

- 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.

Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

Les ressources

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. Imaginons qu'une application ait à afficher une image. Si on prend une petite image, il faut l'agrandir pour qu'elle n'ait pas une dimension ridicule sur un grand écran. Mais en faisant cela, l'image perdra en qualité. Une solution serait donc d'avoir une image pour les petits écrans, une pour les écrans moyens et une pour les grands écrans. C'est ce genre de précautions qu'il faut prendre quand on veut développer pour les appareils mobiles.

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 :

```
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 :

```
Anaïs : 1111111111
Romain: 2222222222
Thomas: 3333333333
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.

```
<?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 :

```
<?xml version="1.0" encoding="utf-8"?>
<bibliotheque>
  <livre style="fantaisie">
    <auteur>George R. R. MARTIN</auteur>
    <titre>A Game Of Thrones</titre>
    <langue>klingon</langue>
    <prix>10.17</prix>
  </livre>
  <livre style="aventure">
    <auteur>Alain Damasio</auteur>
    <titre>La Horde Du Contrevent</titre>
    <prix devise="euro">9.40</prix>
    <recommandation note="20"/>
  </livre>
</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**.

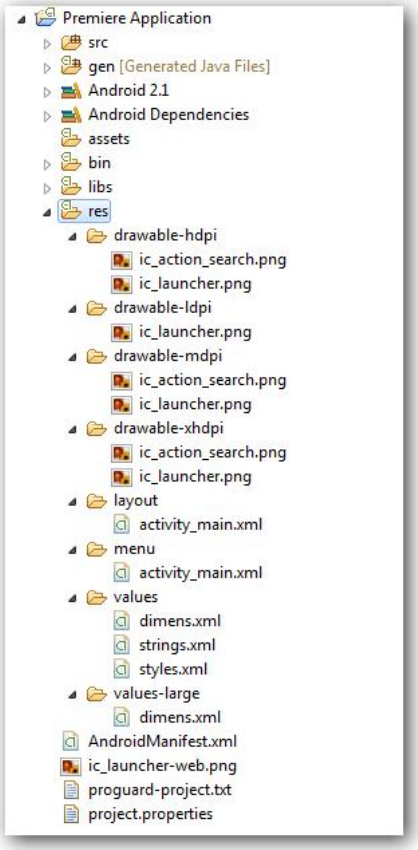
Les nœuds peuvent avoir des **attributs** pour indiquer des informations. Dans notre exemple, le nœud `<prix>` a l'attribut `devise` afin de préciser en quelle devise est exprimé ce prix : `<prix devise="euro">9.40</prix>` pour *La Horde Du Contrevent*, qui vaut donc 9€40. Vous remarquerez que pour *A 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 suivante.



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**

Type	Description	Analyse syntaxique
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`.

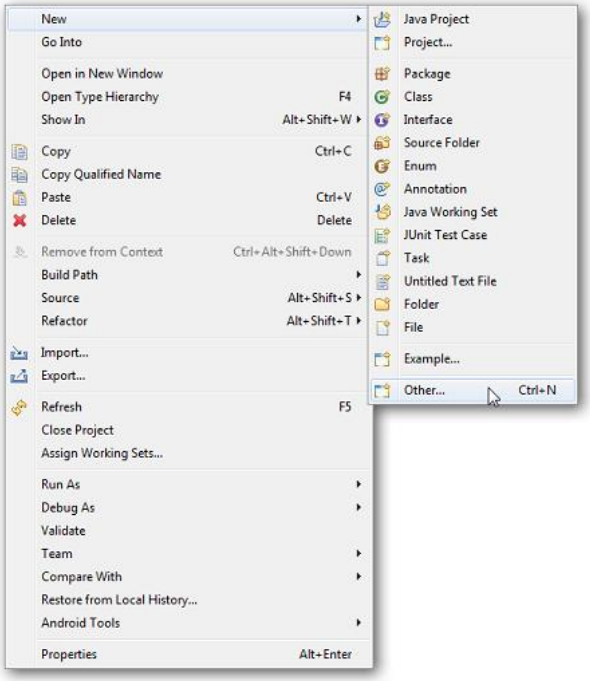
Une mise en page pour chaque orientation et des images adaptées pour chaque résolution. Le quantificateur de l'orientation est surtout utile pour l'interface graphique. Le quantificateur de la résolution sert plutôt à ne pas avoir à ajuster une image et par conséquent à ne pas perdre de qualité.

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

Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

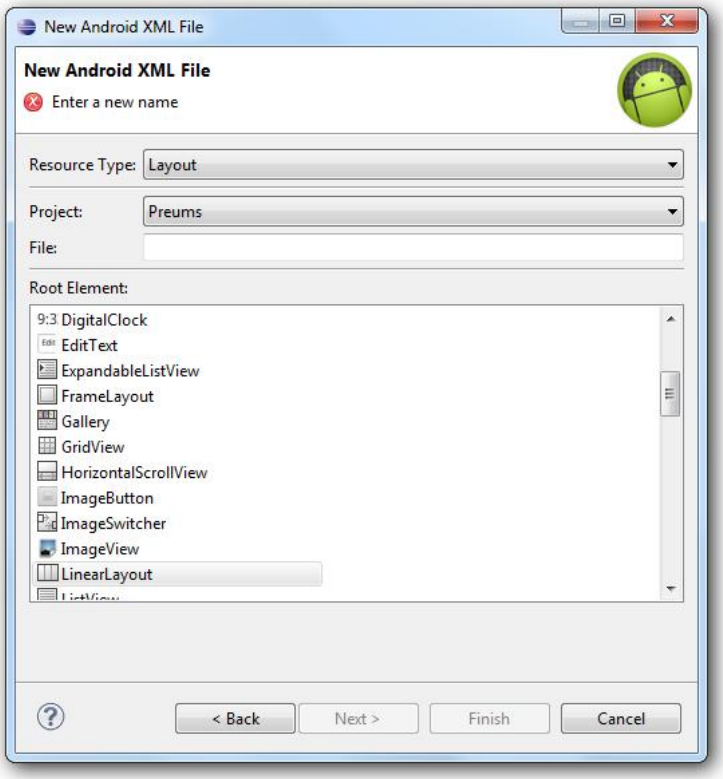
Ajouter un fichier avec Eclipse

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 suivante, qui s'affichera.



L'ADT permet d'ajouter des répertoires facilement

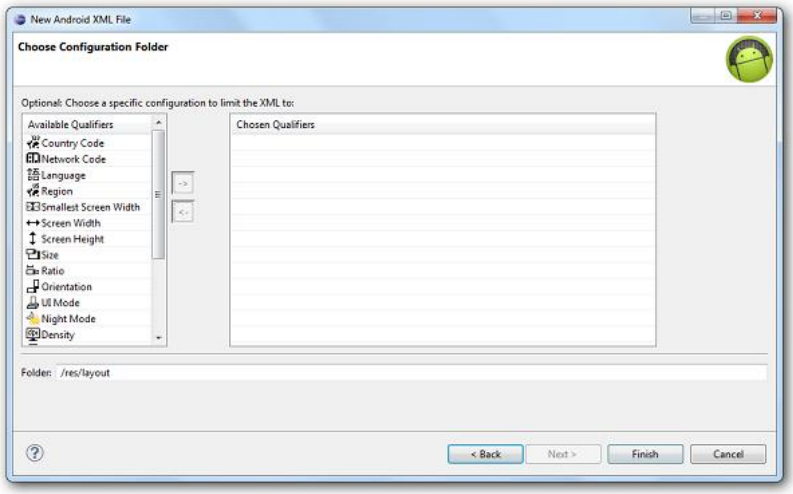
Dans le sous-menu `New`, soit vous cliquez directement sur `Android XML File`, soit, s'il n'est pas présent, vous devrez cliquer sur `Other…`, puis chercher `Android XML File` dans le répertoire `Android`. Cette opération ouvrira un assistant de création de fichiers XML visible à la figure suivante.



L'assistant de création de fichiers XML

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



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. :p

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.

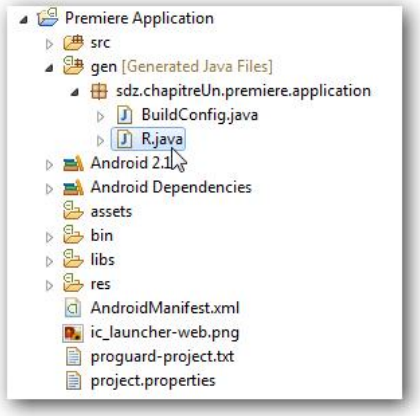
Le `Folder` doit contenir **exactement** `/res/layout-fr-rBE-port-mdpi-v8`.

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 suivante.

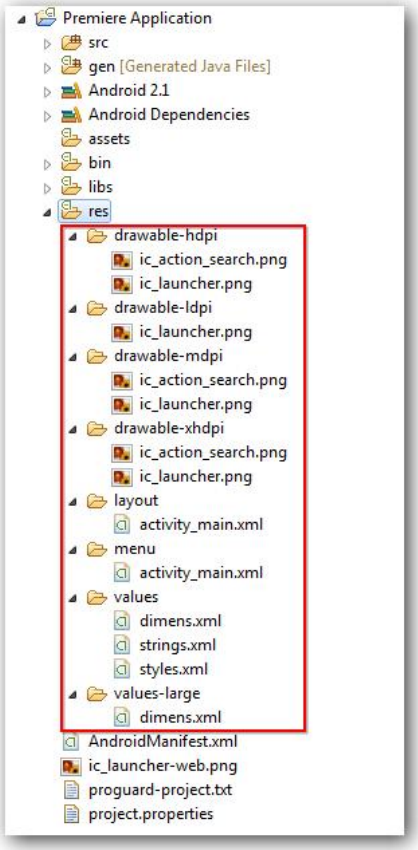


On retrouve le fichier R.java dans gen//

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 suivante).



Tiens, ces noms me disent quelque chose…

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`.
- 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 :

```
int deuxNorm = 2; // Valide !
int deuxHexa = 0x00000002; // Valide, et vaut la même chose que « deuxNorm »
int deuxRed = 0x2; // Valide, et vaut la même chose que « deuxNorm » et « deuxHexa » (évidemment, 000
00002, c'est la même chose que 2 !)
//Ici, nous allons toujours écrire les nombres hexadécimaux avec huit chiffres, même les 0 inutiles !
int beaucoup = 0x0AFA1B00; // Valide !
int marcheraPas = 1x0AFA1B00; // Non ! Un nombre hexadécimal commence toujours par « 0x » !
int marcheraPasNonPlus = 0xG00000000; // Non ! Un chiffre hexadécimal va de 0 à F, on n'accepte pas l
es autres lettres !
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;
import android.os.Bundle;
import android.widget.TextView;

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

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

    text = new TextView(this);
    text.setText(R.string.hello_world);

    setContentView(text);
  }
}
```

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.

- 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.

Constitution des interfaces graphiques

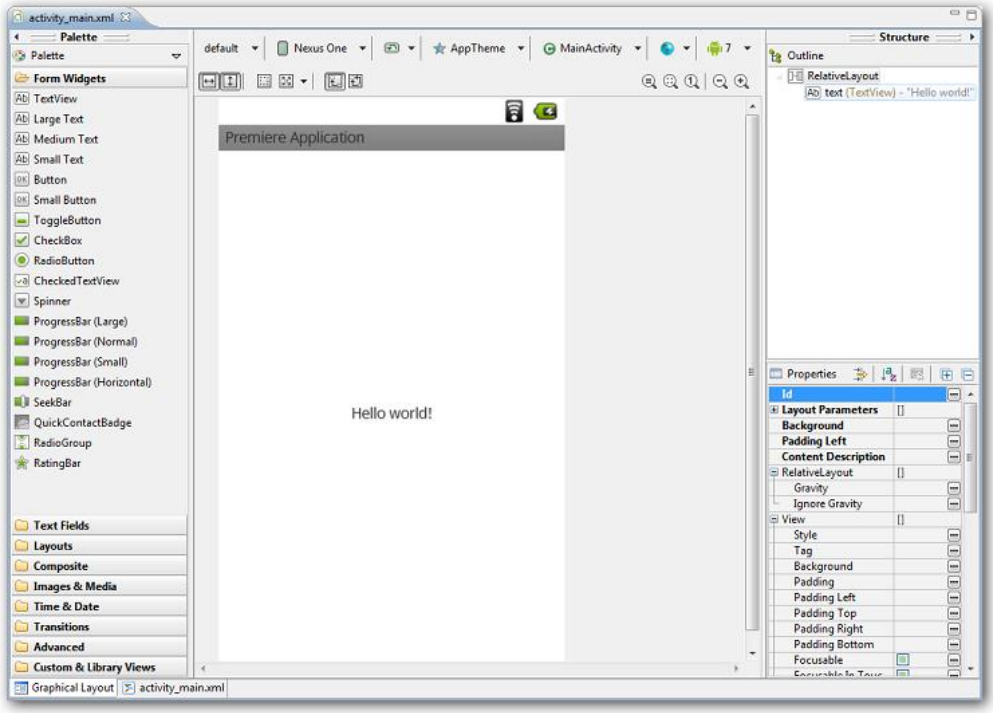
Bien, maintenant que vous avez compris le principe et l'utilité des ressources, voyons comment appliquer nos nouvelles connaissances aux interfaces graphiques. Avec la diversité des machines sous lesquelles fonctionne Android, il faut vraiment exploiter toutes les opportunités offertes par les ressources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

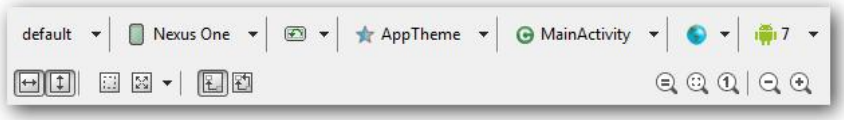


Le fichier est ouvert

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.

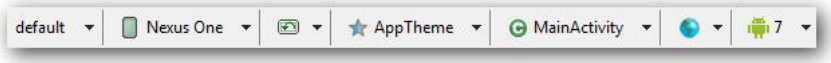
Présentation de l'outil

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



Menu d'options

Ce menu est divisé en deux parties : les icônes du haut et celles du bas. Nous allons nous concentrer sur les icônes du haut pour l'instant (voir figure suivante).



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 imes 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 cette image prise sur Wikipédia (http://upload.wikimedia.org/wikipedia/commons/c/ca/Video_Standards.svg).
- 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.

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

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 suivante.

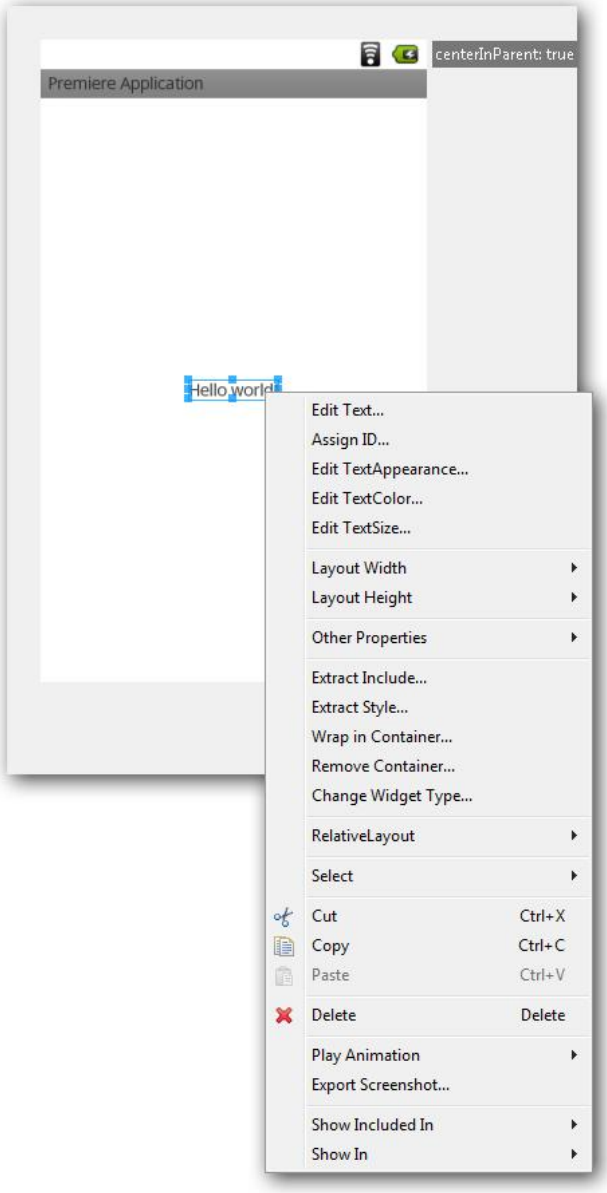
Les icônes de droite du bas menu

- 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.

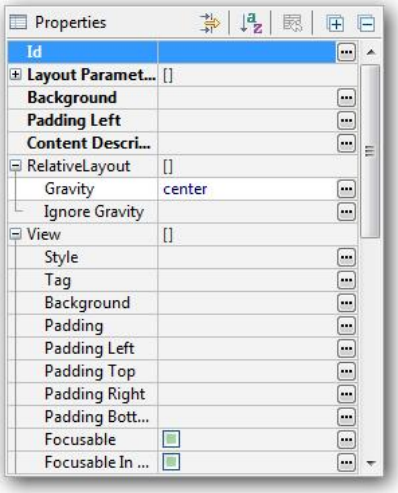
Utilisation

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 suivante.



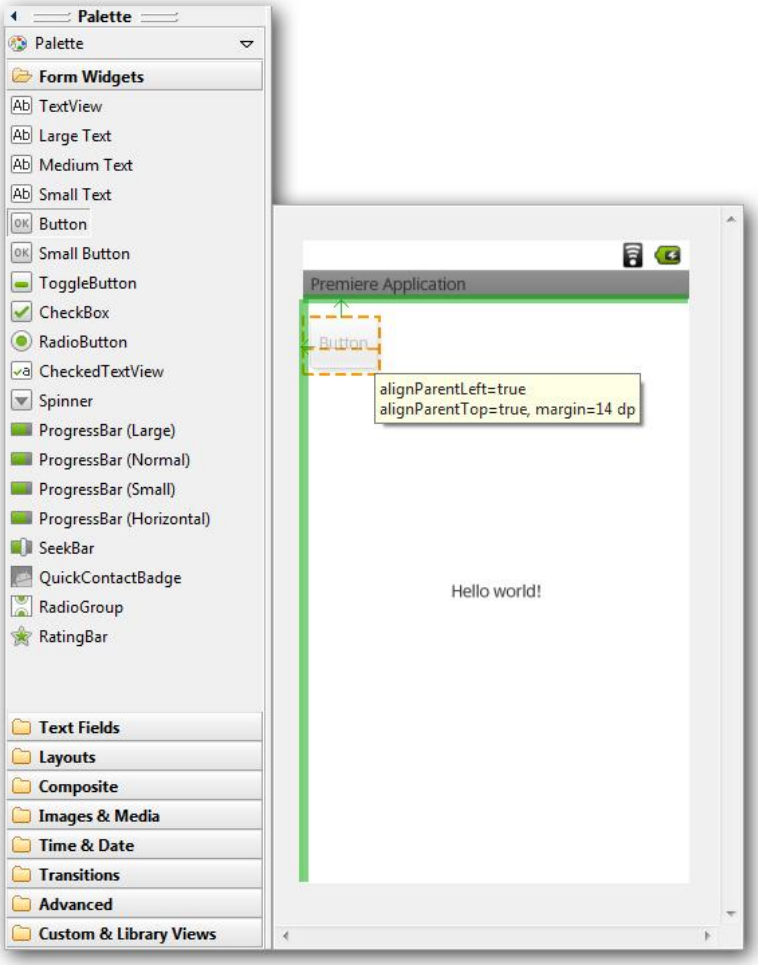
Un menu apparaît lors d'un clic droit sur une vue

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 suivante).



L'encart « Properties »

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 suivante.



Il est possible de faire un cliquer/glisser

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 suivante.

Vous pouvez redimensionner les vues

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.

Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

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 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 :

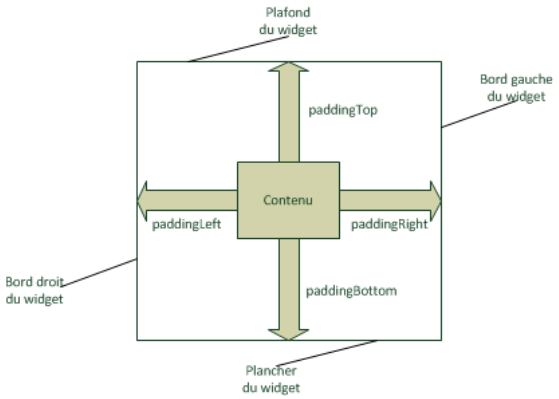
- `dp` ou `dip` : il s'agit d'une unité qui est indépendante de la résolution de l'écran. En effet, il existe d'autres unités comme le pixel (px) ou le millimètre (`mm`), mais celles-ci varient d'un écran à l'autre… Par exemple si vous mettez une taille de 500 dp pour un widget, il aura toujours la même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



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 {
    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;
  }
}
```

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 :

```
//avec « class Class_1 extends Classe_2 »
Classe_2 objetDeux = null;
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.

```
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);
    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;
import android.widget.TextView;

public class TroimsActivity extends Activity {
  RelativeLayout layout = null;
  TextView text = null;

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

    // On récupère notre layout par désérialisation. La méthode inflate retourne un View
    // C'est pourquoi on caste (on convertit) le retour de la méthode avec le vrai type de notre layo
ut, c'est-à-dire RelativeLayout
    layout = (RelativeLayout) RelativeLayout.inflate(this, R.layout.activity_main, null);
    // … puis on récupère TextView grâce à son identifiant
    text = (TextView) layout.findViewById(R.id.text);
    text.setText("Et cette fois, ça fonctionne !");
    setContentView(layout);
    // On aurait très bien pu utiliser « setContentView(R.layout.activity_main) » bien sûr !
  }
}
```

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 ?

```
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);
    monTexte.setPadding(50, 60, 70, 90);

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

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.

- 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.

Les widgets les plus simples

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

```
<TextView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="@string/textView"
  android:textSize="8sp"
  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

```
TextView textView = new TextView(this);
textView.setText(R.string.textView);
textView.setTextSize(8);
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 suivante.

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

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

- Au lieu d'utiliser `android:text`, on utilise `android:hint`. Le problème avec `android:text` 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 ici (http://developer.android.com/reference/android/R.styleable.html#SearchView_inputType).

- Enfin, on peut préciser la taille en lignes que doit occuper l'`EditText` avec `android:lines`.

Exemple en Java

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

Rendu

Le rendu se trouve à la figure suivante.

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 suivante.

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 suivante.

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 suivante.

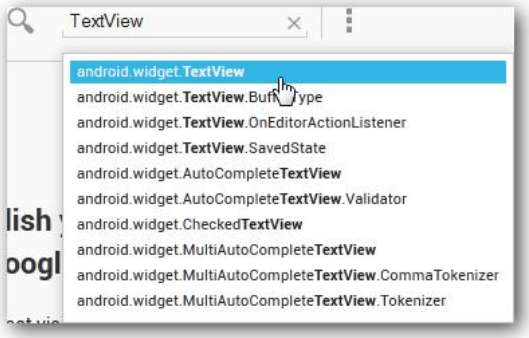
Le bouton radio de droite est sélectionné

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.

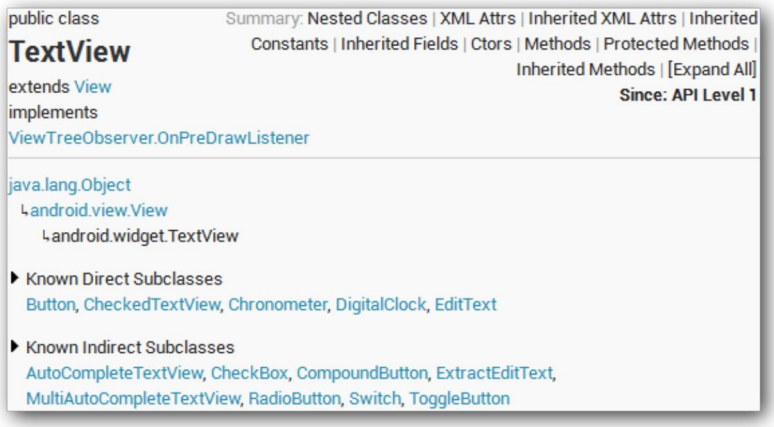
Je vais procéder à l'aide d'un exemple. Je me demande comment faire pour changer la couleur du texte de ma `TextView`. Pour cela, je me dirige vers la documentation officielle : http://developer.android.com/ (http://developer.android.com/).

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 suivante.



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 suivante).



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`).

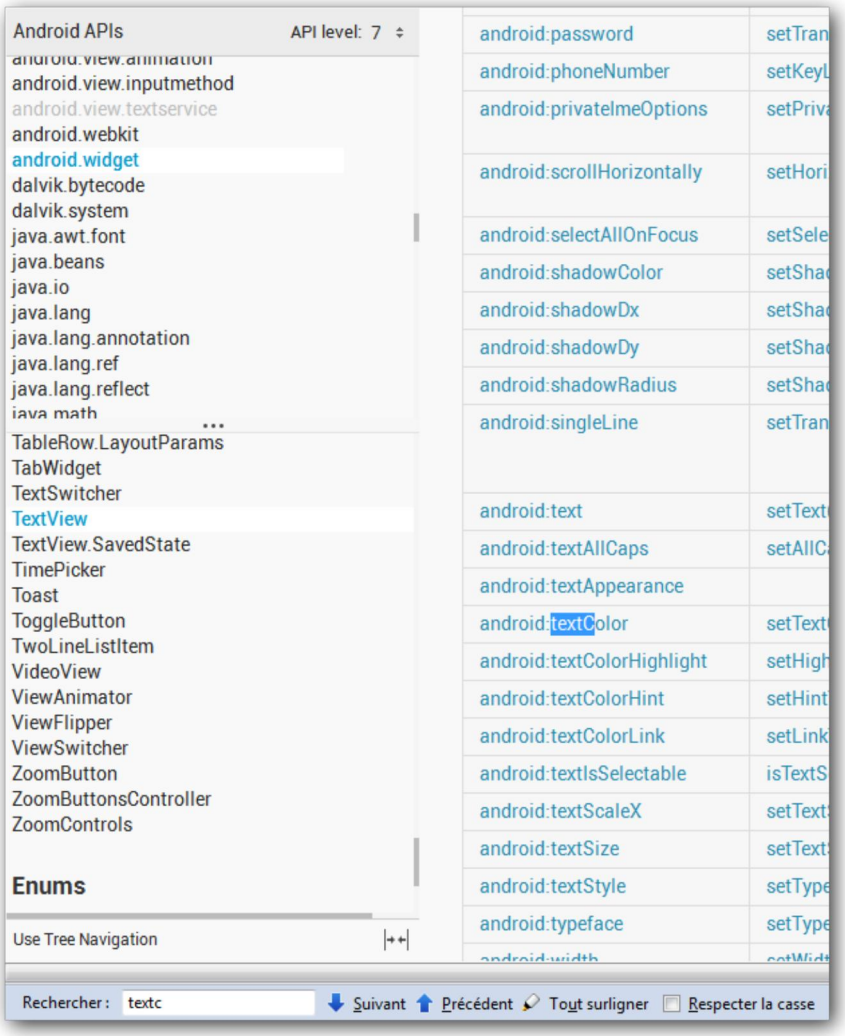
Enfin, on trouve en haut à droite un résumé des différentes sections qui se trouvent dans le document (je vais aussi parler de certaines sections qui ne se trouvent pas dans cette classe mais que vous pourrez rencontrer dans d'autres classes) :

- `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 suivante), ou alors je peux effectuer une recherche sur la page (le raccourci standard pour cela est `Ctrl` + `F`).



Apprenez à utiliser les recherches

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 ? 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.

> 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 suivante.

> 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`. Vous pourrez trouver d'autres valeurs pour indiquer une couleur à cet endroit (http://www.netalya.com/fr/palette.asp).
- 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
    android:id="@+id/group"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checkedButton="@+id/radio2"
    android:orientation="horizontal"
  >
    <RadioButton
      android:id="@+id/radio1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Mètre"
    />
    <RadioButton
      android:id="@+id/radio2"
      android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="Centimètre"
      />
    </RadioGroup>
    <CheckBox
      android:id="@+id/mega"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Mega fonction !"
    />
    <Button
      android:id="@+id/calcul"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Calculer l'IMC"
    />
    <Button
      android:id="@+id/raz"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="RAZ"
    />
    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Résultat:"
    />
    <TextView
      android:id="@+id/result"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:text="Vous devez cliquer sur le bouton « Calculer l'IMC » pour obtenir un résultat."
    />
  </LinearLayout>
```

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. :)

Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

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 :

```
Bouton b = new Button(getContext());
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 :

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

// Notre activité détectera les touchers et les clics sur les vues qui se sont inscrites
public class Main extends Activity implements View.OnTouchListener, View.OnClickListener {
  private Button b = null;

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

    setContentView(R.layout.main);

    b = (Button) findViewById(R.id.boutton);
    b.setOnTouchListener(this);
    b.setOnClickListener(this);
  }

  @Override
  public boolean onTouch(View v, MotionEvent event) {
    /* Réagir au toucher */
    return true;
  }

  @Override
  public void onClick(View v) {
    /* Réagir au clic */
  }
}
```

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 :

```
public void onClick(View v) {
  // On récupère l'identifiant de la vue, et en fonction de cet identifiant…
  switch(v.getId()) {

    // Si l'identifiant de la vue est celui du premier bouton
    case R.id.bouton1:
    /* Agir pour bouton 1 */
    break;

    // Si l'identifiant de la vue est celui du deuxième bouton
    case R.id.bouton2:
    /* Agir pour bouton 2 */
    break;

    /* etc. */
  }
}
```

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;
      }
    });

    touchAndClick.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        // Réagir au clic
      }
    });

    clickOnly.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        // Réagir au clic
      }
    });
  }
}
```

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.

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

public class Main extends Activity {
  private OnClickListener clickListenerBoutons = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
      /* Réagir au clic pour les boutons 1 et 2*/
    }
  };

  private OnTouchListener touchListenerBouton1 = new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
      /* Réagir au toucher pour le bouton 1*/
      return onTouch(v, event);
    }
  };

  private OnTouchListener touchListenerBouton3 = new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
      /* Réagir au toucher pour le bouton 3*/
      return super.onTouch(v, event);
    }
  };

  Button b1 = null;
  Button b2 = null;
  Button b3 = null;

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

    setContentView(R.layout.main);

    b1 = (Button) findViewById(R.id.bouton1);
    b2 = (Button) findViewById(R.id.bouton2);
    b3 = (Button) findViewById(R.id.bouton3);

    b1.setOnTouchListener(touchListenerBouton1);
    b1.setOnClickListener(clickListenerBoutons);
    b2.setOnClickListener(clickListenerBoutons);
    b3.setOnTouchListener(touchListenerBouton3);
  }
}
```

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 (http://developer.android.com/reference/android/view/MotionEvent.html)`, 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 détecté 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
    int hauteur = bouton.getHeight();

    // On récupère la coordonnée sur l'abscisse (X) de l'évènement
    float x = event.getX();
    // On récupère la coordonnée sur l'ordonnée (Y) de l'évènement
    float y = event.getY();

    // Puis on change la taille du texte selon la formule indiquée dans l'énoncé
    bouton.setTextSize(Math.abs(x - largeur / 2) + Math.abs(y - hauteur / 2));
    // Le toucher est fini, on veut continuer à détecter les touchers d'après
    return true;
  }
}
```

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{\mbox{poids (en kilogrammes)}}{{\mbox{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 : $\frac{171 \mbox{ centimetres}}{100} = 1.71 \mbox{ metres}$.
- 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

- Voici la syntaxe pour construire un `Toast` :
  `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 la méthode `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()`. Plus d'informations sur Editable (http://developer.android.com/reference/android/text/Editable.html).
- 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 (http://developer.android.com/reference/android/text/TextWatcher.html). C'est comme un listener, mais ça n'en porte pas le nom !

Ma solution

```
import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.View.OnKeyListener;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.RadioGroup;
import android.widget.TextView;
import android.widget.Toast;

public class IMCActivity extends Activity {
  // La chaîne de caractères par défaut
  private final String defaut = "Vous devez cliquer sur le bouton « Calculer l'IMC » pour obtenir un
 résultat.";
  // La chaîne de caractères de la megafonction
  private final String megaString = "Vous faites un poids parfait ! Wahou ! Trop fort ! On dirait Bra
d Pitt (si vous êtes un homme)/Angelina Jolie (si vous êtes une femme)/Willy (si vous êtes un orque)
 !";

  Button envoyer = null;
  Button raz = null;

  EditText poids = null;
  EditText taille = null;

  RadioGroup group = null;

  TextView result = null;

  CheckBox mega = null;

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

    // On récupère toutes les vues dont on a besoin
    envoyer = (Button)findViewById(R.id.calcul);

    raz = (Button)findViewById(R.id.raz);

    taille = (EditText)findViewById(R.id.taille);
    poids = (EditText)findViewById(R.id.poids);

    mega = (CheckBox)findViewById(R.id.mega);

    group = (RadioGroup)findViewById(R.id.group);
```

```
  result = (TextView)findViewById(R.id.result);

  // On attribue un listener adapté aux vues qui en ont besoin
  envoyer.setOnClickListener(envoyerListener);
  raz.setOnClickListener(razListener);
  taille.addTextChangedListener(textWatcher);
  poids.addTextChangedListener(textWatcher);

  // Solution avec des onKey
  //taille.setOnKeyListener(modificationListener);
  //poids.setOnKeyListener(modificationListener);
  mega.setOnClickListener(checkedListener);
}

/*
// Se lance à chaque fois qu'on appuie sur une touche en étant sur un EditText
private OnKeyListener modificationListener = new OnKeyListener() {
  @Override
  public boolean onKey(View v, int keyCode, KeyEvent event) {
    // On remet le texte à sa valeur par défaut pour ne pas avoir de résultat incohérent
    result.setText(defaut);
    return false;
  }
};*/

private TextWatcher textWatcher = new TextWatcher() {

  @Override
  public void onTextChanged(CharSequence s, int start, int before, int count) {
    result.setText(defaut);
  }

  @Override
  public void beforeTextChanged(CharSequence s, int start, int count,
    int after) {

  }

  @Override
  public void afterTextChanged(Editable s) {

  }
};

// Uniquement pour le bouton "envoyer"
private OnClickListener envoyerListener = new OnClickListener() {
  @Override
  public void onClick(View v) {
    if(!mega.isChecked()) {
      // Si la megafonction n'est pas activée
      // On récupère la taille
```

```
        String t = taille.getText().toString();
        // On récupère le poids
        String p = poids.getText().toString();

        float tValue = Float.valueOf(t);

        // Puis on vérifie que la taille est cohérente
        if(tValue == 0)
          Toast.makeText(IMCActivity.this, "Hého, tu es un Minipouce ou quoi ?", Toast.LENGTH_SHORT).
show();
        else {
          float pValue = Float.valueOf(p);
          // Si l'utilisateur a indiqué que la taille était en centimètres
          // On vérifie que la Checkbox sélectionnée est la deuxième à l'aide de son identifiant
          if(group.getCheckedRadioButtonId() == R.id.radio2)
            tValue = tValue / 100;

          tValue = (float)Math.pow(tValue, 2);
          float imc = pValue / tValue;
          result.setText("Votre IMC est " + String.valueOf(imc));
        }
      } else
        result.setText(megaString);
    }
  };

  // Listener du bouton de remise à zéro
  private OnClickListener razListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
      poids.getText().clear();
      taille.getText().clear();
      result.setText(defaut);
    }
  };

  // Listener du bouton de la megafonction.
  private OnClickListener checkedListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
      // On remet le texte par défaut si c'était le texte de la megafonction qui était écrit
      if(!((CheckBox)v).isChecked() && result.getText().equals(megaString))
        result.setText(defaut);
    }
  };
}
```

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

Curieux va ! :p 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 !

- 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.

Organiser son interface avec des layouts

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 ! :D

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

```
<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/premier"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Premier bouton" />

  <Button
    android:id="@+id/second"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Second bouton" />
</LinearLayout>
```

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

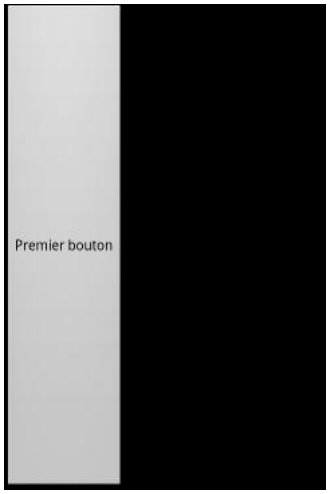
Les deux boutons prennent toute la largeur

- 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

```
<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/premier"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:text="Premier bouton" />

  <Button
    android:id="@+id/second"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:text="Second bouton" />
</LinearLayout>
```

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



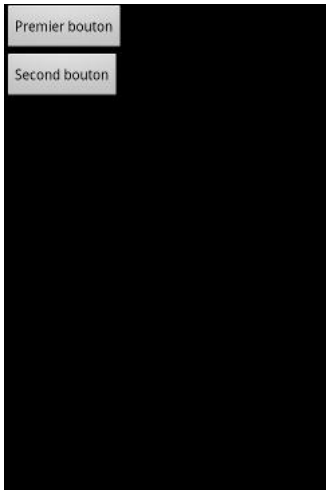
Le premier bouton fait toute la hauteur, on ne voit donc pas le deuxième bouton

- 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

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content" >
  <Button
    android:id="@+id/premier"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:text="Premier bouton" />
  <Button
    android:id="@+id/second"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:text="Second bouton" />
</LinearLayout>
```

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

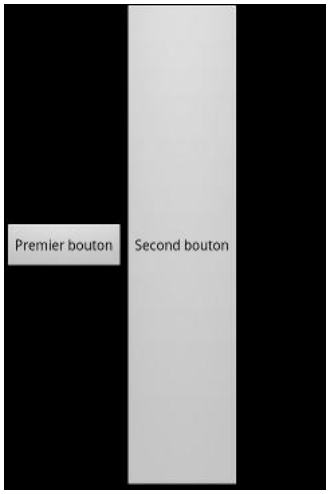


- Le `LinearLayout` est vertical et prend toute la place en largeur mais uniquement la taille nécessaire en hauteur : dans ce cas précis, la taille nécessaire sera calculée en fonction de la taille des enfants.
- Le premier bouton prend toute la place possible dans le parent. Comme le parent prend le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

Quatrième exemple

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="horizontal"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
  <Button
    android:id="@+id/premier"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Premier bouton" />
  <Button
    android:id="@+id/second"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:text="Second bouton" />
</LinearLayout>
```

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



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.

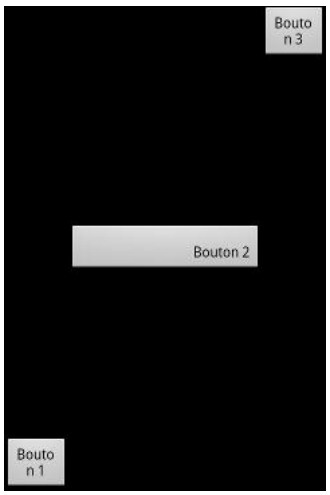
> Une astuce que j'utilise souvent consiste à faire en sorte que la somme des poids dans un même layout fasse 100. C'est une manière plus évidente pour répartir les poids.

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 ? :)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="horizontal"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
  <Button
    android:id="@+id/bouton1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom"
    android:layout_weight="40"
    android:text="Bouton 1" />
  <Button
    android:id="@+id/bouton2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_weight="20"
    android:gravity="bottom|right"
    android:text="Bouton 2" />
  <Button
    android:id="@+id/bouton3"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="top"
    android:layout_weight="40"
    android:text="Bouton 3" />
</LinearLayout>
```

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



Trois boutons placés différemment

Comme 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 suivante.

Essayez d'obtenir la même interface

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.

```
<?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">
  <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"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
  >
    <TextView
      android:layout_width="wrap_content"
      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"
      android:layout_weight="1"
    />
  </LinearLayout>
  <RadioGroup
    android:id="@+id/group"
```

```
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:checkedButton="@+id/radio2"
  android:orientation="horizontal"
>
  <RadioButton
    android:id="@+id/radio1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Mètre"
  />
  <RadioButton
    android:id="@+id/radio2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Centimètre"
  />
</RadioGroup>
<CheckBox
  android:id="@+id/mega"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Mega fonction !"
/>
<LinearLayout
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:orientation="horizontal"
>
  <Button
    android:id="@+id/calcul"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Calculer l'IMC"
    android:layout_weight="1"
    android:layout_marginLeft="25dip"
    android:layout_marginRight="25dip"
  />
  <Button
    android:id="@+id/raz"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="RAZ"
    android:layout_weight="1"
    android:layout_marginLeft="25dip"
    android:layout_marginRight="25dip"
  />
</LinearLayout>
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Résultat:"
```

```
    />
    <TextView
      android:id="@+id/result"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:text="Vous devez cliquer sur le bouton « Calculer l'IMC » pour obtenir un résultat."
    />
</LinearLayout>
```

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.

Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

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 suivante.

Deux vues sont empilées

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` ;
- `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

```
<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="En haut !"
    android:layout_alignParentTop="true" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="En bas !"
    android:layout_alignParentBottom="true" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="A gauche !"
    android:layout_alignParentLeft="true" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="A droite !"
    android:layout_alignParentRight="true" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Ces soirées là !"
    android:layout_centerInParent="true" />
</RelativeLayout>
```

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

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.

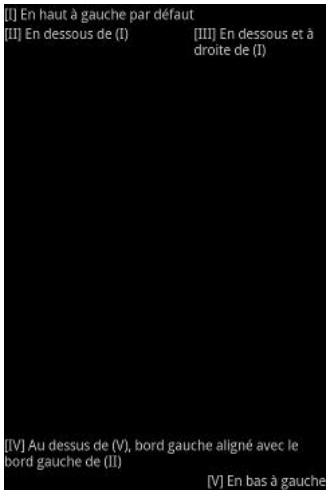
Enfin, il ne faut pas oublier que le principal intérêt de ce layout est de pouvoir placer les éléments les uns par rapport aux autres. Pour cela il existe deux catégories d'attributs :

- 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) et `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 suivante.



Les TextView sont bien placés

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

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView
    android:id="@+id/textPoids"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Poids : "
    android:textStyle="bold"
    android:textColor="#FF0000"
  />
  <EditText
    android:id="@+id/poids"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="Poids"
    android:inputType="numberDecimal"
    android:layout_toRightOf="@id/textPoids"
    android:layout_alignParentRight="true"
  />
  <TextView
    android:id="@+id/textTaille"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Taille : "
    android:textStyle="bold"
    android:textColor="#FF0000"
    android:gravity="left"
    android:layout_below="@id/poids"
  />
  <EditText
    android:id="@+id/taille"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="Taille"
    android:inputType="numberDecimal"
    android:layout_below="@id/poids"
    android:layout_toRightOf="@id/textTaille"
    android:layout_alignParentRight="true"
  />
  <RadioGroup
    android:id="@+id/group"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checkedButton="@+id/radio2"
    android:orientation="horizontal"
    android:layout_below="@id/taille"
  >
    <RadioButton
      android:id="@+id/radio1"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Mètre"
      />
      <RadioButton
        android:id="@+id/radio2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Centimètre"
      />
    </RadioGroup>
    <CheckBox
      android:id="@+id/mega"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Mega fonction !"
      android:layout_below="@id/group"
    />
    <Button
      android:id="@+id/calcul"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Calculer l'IMC"
      android:layout_below="@id/mega"
      android:layout_marginLeft="25dip"
    />
    <Button
      android:id="@+id/raz"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="RAZ"
      android:layout_below="@id/mega"
      android:layout_alignRight="@id/taille"
      android:layout_marginRight="25dip"
    />
    <TextView
      android:id="@+id/resultPre"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Résultat:"
      android:layout_below="@id/calcul"
    />
    <TextView
      android:id="@+id/result"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:text="Vous devez cliquer sur le bouton « Calculer l'IMC » pour obtenir un résultat."
      android:layout_below="@id/resultPre"
    />
  </RelativeLayout>
```

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.

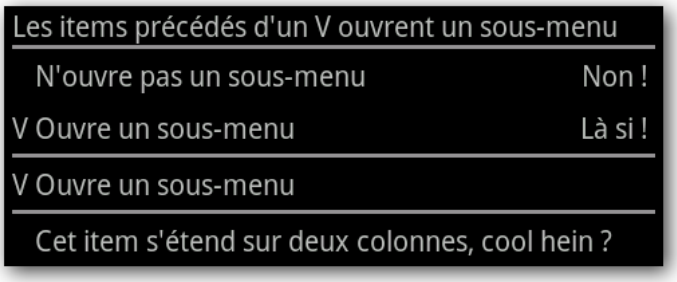
TableLayout : placer les éléments comme dans un tableau

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"
  />
  <TableRow>
    <TextView
      android:text="V"
    />
    <TextView
      android:text="Ouvre un sous-menu"
      android:padding="3dip"
    />
```

```
    </TableRow>
    <View
      android:layout_height="2dip"
      android:background="#FF909090"
    />
    <TableRow>
      <TextView
        android:layout_column="1"
        android:layout_span="2"
        android:text="Cet item s'étend sur deux colonnes, cool hein ?"
        android:padding="3dip"
      />
    </TableRow>
  </TableLayout>
```

Ce qui donne la figure suivante.



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>`.

```
<TextView
  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.

Il est possible de 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`.

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

```
<?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">
  <TableRow>
    <TextView
      android:text="Poids : "
      android:textStyle="bold"
      android:textColor="#FF0000"
      android:gravity="center"
    />
    <EditText
      android:id="@+id/poids"
      android:hint="Poids"
      android:inputType="numberDecimal"
      android:layout_span="2"
    />
  </TableRow>
  <TableRow>
    <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"
      android:layout_span="2"
    />
  </TableRow>
  <RadioGroup
    android:id="@+id/group"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checkedButton="@+id/radio2"
    android:orientation="horizontal">
    <RadioButton
      android:id="@+id/radio1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Mètre"
    />
    <RadioButton
      android:id="@+id/radio2"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Centimètre"
      />
    </RadioGroup>
    <CheckBox
      android:id="@+id/mega"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Mega fonction !"
    />
    <TableRow>
      <Button
        android:id="@+id/calcul"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Calculer l'IMC"
      />
      <Button
        android:id="@+id/raz"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="RAZ"
        android:layout_column="2"
      />
    </TableRow>
    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Résultat:"
    />
    <TextView
      android:id="@+id/result"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:text="Vous devez cliquer sur le bouton « Calculer l'IMC » pour obtenir un résultat."
    />
  </TableLayout>
```


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 :

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

- `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. Attention de ne lui donner qu'un fils et de ne pas fournir des vues déjà « scrollable » sinon il y aura des conflits.

Les autres ressources

Maintenant que vous avez parfaitement compris ce qu'étaient les ressources, pourquoi et comment les utiliser, je vous propose de voir… comment les créer. :D 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.

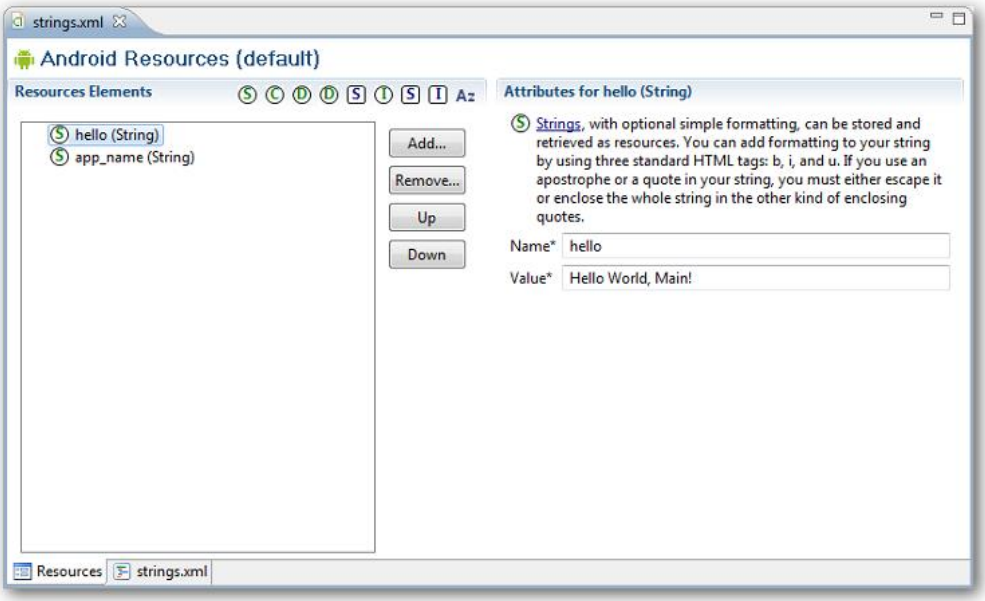
Un dernier conseil avant d'entrer dans le vif du sujet : créez le plus de ressources possible, dès que vous le pouvez. Ainsi, vos applications seront plus flexibles, et le développement sera plus évident.

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 suivante).



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.

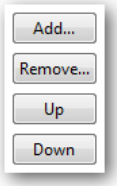
- Ou alors mettre toutes les données d'une activité dans un fichier, ce qui demande moins de travail, mais nécessite une meilleure organisation afin de pouvoir s'y retrouver.

  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`)
- Az Az
  : 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 suivante) vous permet de créer ou supprimer des données.



- 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 suivante.

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 :

```
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 :

```
@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 `Android.R.type_de_ressource.nom_de_la_ressource` en Java et `@android:type_de_ressource/nom_de_la_ressource` en XML.

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 apostrophes. 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 `&laquo;` et à la place de > il faut insérer `&raquo;`. 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` :

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

Et le code Java associé :

```
import android.app.Activity;
import android.os.Bundle;
import android.text.Html;
import android.text.Spanned;
import android.widget.Button;

public class StringExampleActivity extends Activity {
  Button button = null;
  String hist = null;

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

    // On récupère notre ressource au format String
    hist = getResources().getString(R.string.histoire);
    // On le convertit en Spanned
    Spanned marked_up = Html.fromHtml(hist);

    button = new Button(this);
    // Et on attribue le Spanned au bouton
    button.setText(marked_up);

    setContentView(button);
  }
}
```

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 cette page (http://developer.android.com/reference/java/util/Formatter.html)).

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

On peut le faire en récupérant la ressource :

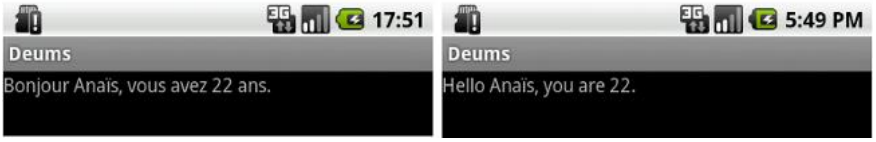
```
Resources res = getResources();
// Anaïs ira en %1 et 22 ira en %2
String chaine = res.getString(R.string.hello, "Anaïs", 22);
```

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 suivante.



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"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

</LinearLayout>
```

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

```
import android.app.Activity;
import android.content.res.Resources;
import android.os.Bundle;
import android.widget.TextView;

public class DeumsActivity extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    Resources res = getResources();
    // Anaïs se mettra dans %1 et 22 ira dans %2, mais le reste changera en fonction de la langue du
 terminal !
    String chaine = res.getString(R.string.hello, "Anaïs", 22);
    TextView vue = (TextView)findViewById(R.id.vue);
    vue.setText(chaine);
  }
}
```

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

Les drawables

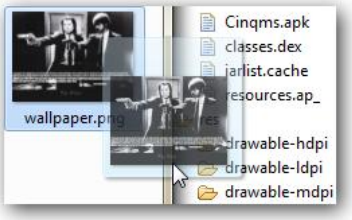
La dénomination « drawable » rassemble tous les fichiers « dessinables » (oui, ce mot n'existe pas en français, mais « drawable » n'existe pas non plus en anglais après tout :p), c'est-à-dire les dessins ou les images. Je ne parlerai que des images puisque ce sont les drawables les plus utilisés et les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



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

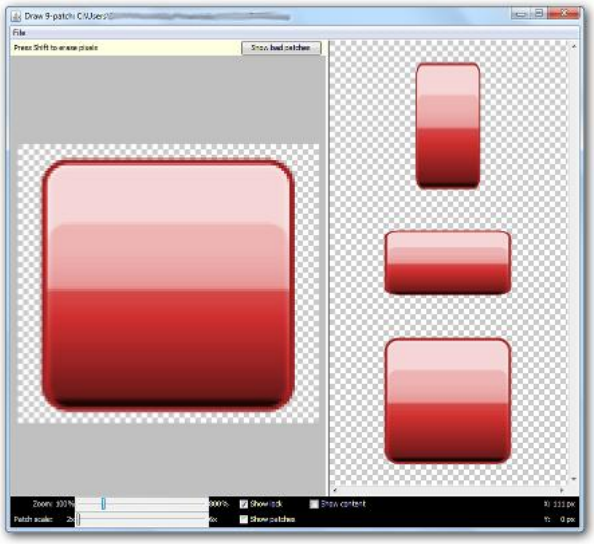
Le nom du fichier déterminera l'identifiant du drawable, et il pourra contenir toutes les lettres minuscules, tous les chiffres et des underscores (_), mais attention, pas de majuscules. Puis, on pourra récupérer le drawable à l'aide de `R.drawable.nom_du_fichier_sans_l_extension`.

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 suivante, qui est aimablement prêtée par ce grand monsieur (http://android9patch.blogspot.com/), qui nous autorise à utiliser ses images, même pour des projets professionnels, un grand merci à lui.

Nous allons utiliser cette image pour l'exemple

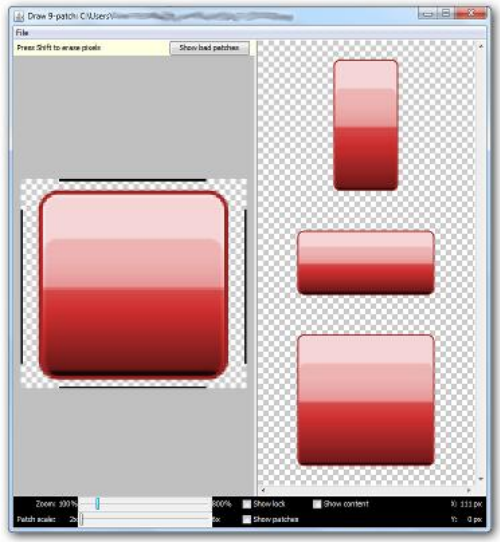
Cette image ne paye pas de mine, mais elle pourra être étendue jusqu'à former une image immense sans pour autant être toute pixellisée. L'astuce consiste à indiquer quelles parties de l'image peuvent être étendues, et le SDK d'Android contient un outil pour vous aider dans votre démarche. Par rapport à l'endroit où vous avez installé le SDK, il se trouve dans `\Android ools\draw9patch.bat`. Vous pouvez directement glisser l'image dans l'application pour l'ouvrir ou bien aller dans `File > Open 9-patch…` (voir figure suivante).



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 suivante.



Indiquez les zones extensibles ainsi que l'emplacement du contenu

Vous voyez la différence ? Les images étirées montrent beaucoup moins de pixels et les 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 suivante vous montre plus clairement à quoi correspondent les bords :

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

- 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. Par exemple pour changer la couleur d'un texte, on va utiliser l'attribut `android:textColor`.

- 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 :

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

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

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

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

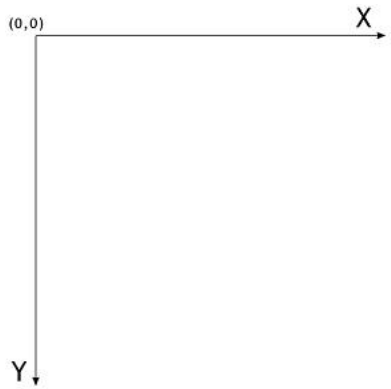
> Il est possible de n'avoir qu'un seul parent pour un style, ce qui peut être très vite pénible, alors organisez-vous à l'avance !

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 le constructeur d'une vue (http://developer.android.com/reference/android/view/View.html#View%28android.content.Context,%20android. : `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 :

```
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 suivante, 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 :

- 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 :

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <scale
    android:fromXScale="1.0"
    android:fromYScale="1.0"
    android:toXScale="2.0"
    android:toYScale="0.5"
    android:pivotX="50%"
    android:pivotY="50%" />
  <alpha
    android:fromAlpha="1.0"
    android:toAlpha="0.0" />
</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 :

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <scale
    android:fromXScale="1.0"
    android:fromYScale="1.0"
    android:toXScale="2.0"
    android:toYScale="0.5"
    android:pivotX="50%"
    android:pivotY="50%"
    android:duration="300"/>
  <alpha
    android:fromAlpha="1.0"
    android:toAlpha="0.0"
    android:startOffset="300"
    android:duration="2000"/>
</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`.

```
// On crée un utilitaire de configuration pour cette animation
Animation animation = AnimationUtils.loadAnimation(contexte_dans_lequel_se_situe_la_vue, identifiant_
de_l_animation);
// On l'affecte au widget désiré, et on démarre l'animation
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 :

```
<?xml version="1.0" encoding="utf-8"?>
<layoutAnimation xmlns:android="http://schemas.android.com/apk/res/android"
  android:delay="10%"
  android:animationOrder="random"
  android:animation="@anim/animation_standard"
/>
```

Puis on peut l'utiliser dans le code Java avec :

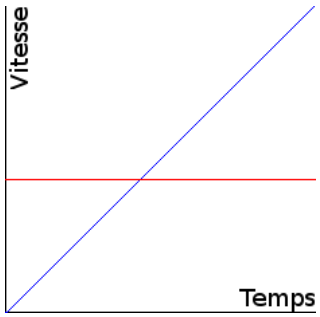
```
LayoutAnimationController animation = AnimationUtils.loadLayoutAnimation(contexte_dans_lequel_se_situe_la_vue, identifiant_de_l_animation);
layout.setLayoutAnimation(animation);
```

> 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 suivante : 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. :D



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"`.

En ce qui concerne 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 ?
  }
});
```

- 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.

TP : un bloc-notes

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 suivante vous montre ce que j'obtiens. Ce n'est pas très joli, mais ça fonctionne.

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 suivante).

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 suivante.

Le texte est mis en forme en temps téel dans la zone de prévisualisation

---

Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

Spécifications techniques

Fichiers à utiliser

On va d'abord utiliser les smileys du Site du Zéro :

.

Pour les boutons, j'ai utilisé les 9-patches visibles à la figure suivante.

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>`

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 :

```
public class Exemple implements ImageGetter {
  @Override
  public Drawable getDrawable(String smiley) {
    Drawable retour = null;

    Resources resources = context.getResources();

    retour = resources.getDrawable(R.drawable.ic_launcher);

    // On délimite l'image (elle va de son coin en haut à gauche à son coin en bas à droite)
    retour.setBounds(0, 0, retour.getIntrinsicWidth(), retour.getIntrinsicHeight());
    return retour;
  }
}
```

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 :

```
<nom.du.package.NomDeLaClasse>
```

Liens

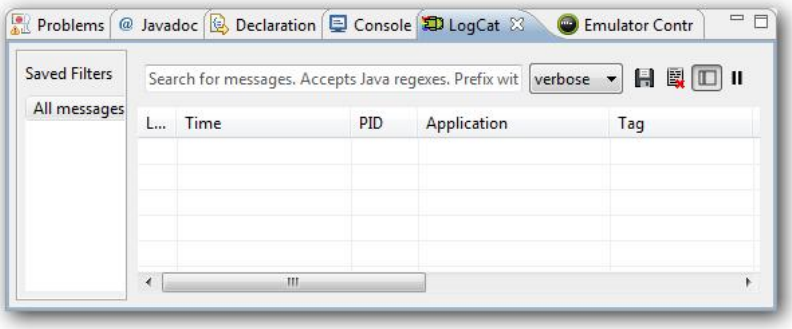
Plus d'informations :

- EditText (http://developer.android.com/reference/android/widget/EditText.html)
- Html (http://developer.android.com/reference/android/text/Html.html) et Html.ImageGetter (http://developer.android.com/reference/android/text/Html.ImageGetter.html)
- TextView (http://developer.android.com/reference/android/widget/TextView.html)
- TextWatcher (http://developer.android.com/reference/android/text/TextWatcher.html)
- TranslateAnimation (http://developer.android.com/reference/android/view/animation/TranslateAnimation.htm

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 suivante.



Le Logcast est ouvert

La première chose à faire, c'est de cliquer sur le troisième bouton en haut à droite (voir figure suivante).

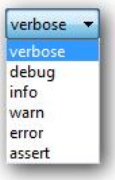
Cliquez sur le troisième bouton

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 suivante.



Cette liste déroulante permet d'afficher dans le Logcat les messages que vous souhaitez

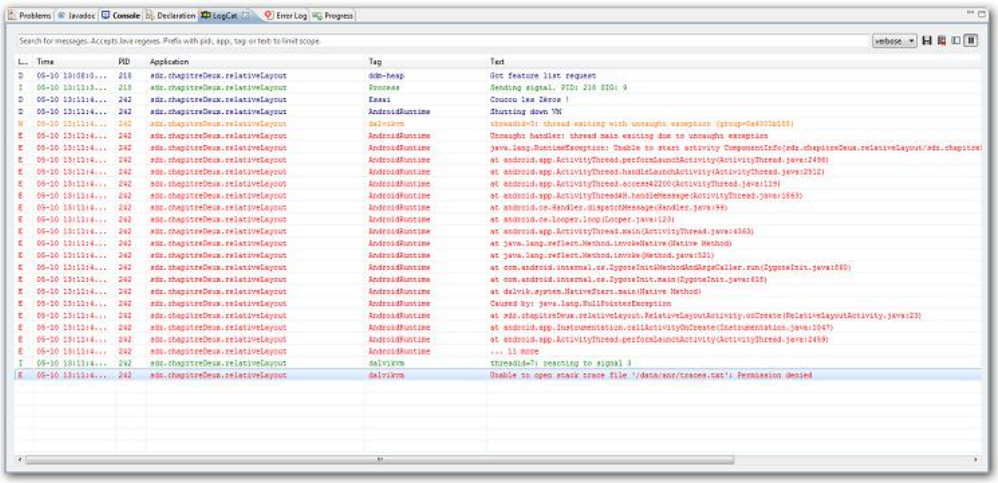
Vous voyez, la première lettre utilisée dans le code indique un type de message : `v` pour `Verbose`, `d` pour `Debug`, etc.

> 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.

Par exemple avec le code :

```
Log.d("Essai", "Coucou les Zéros !");
TextView x = null;
x.setText("Va planter");
```

On obtient la figure suivante.

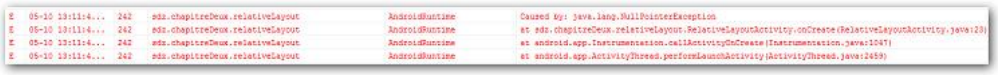


À la figure suivante, on peut voir le message que j'avais inséré.

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 suivante que mon étourderie a provoqué un plantage de l'application.



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 :

```
<resources>
  <color name="background">#99CCFF</color>
  <color name="black">#000000</color>
  <color name="translucide">#00000000</color>
</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 :

```
<resources>
  <style name="blueBackground">
    <item name="android:background">@color/background</item>
  </style>

  <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"
    android:padding="5dip"
    android:src="@drawable/smile"
    style="@style/translucide"
  />

  <ImageButton
    android:id="@+id/heureux"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@id/smile"
    android:layout_centerHorizontal="true"
    android:contentDescription="@string/heureux"
    android:padding="5dip"
    android:src="@drawable/heureux"
    style="@style/translucide"
```

```
/>

<ImageButton
  android:id="@+id/clin"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignTop="@id/smile"
  android:layout_alignLeft="@+id/underline"
  android:layout_alignRight="@+id/underline"
  android:contentDescription="@string/clin"
  android:padding="5dip"
  android:src="@drawable/clin"
  style="@style/translucide"
/>

<Button
  android:id="@+id/italic"
  style="@style/optionButton"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignParentTop="true"
  android:layout_centerHorizontal="true"
  android:text="@string/italic"
/>

<Button
  android:id="@+id/underline"
  style="@style/optionButton"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignParentTop="true"
  android:layout_alignParentRight="true"
  android:text="@string/underline"
/>

<RadioGroup
  android:id="@+id/colors"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignParentLeft="true"
  android:layout_alignParentRight="true"
  android:layout_below="@id/heureux"
  android:orientation="horizontal" >

  <RadioButton
    android:id="@+id/black"
    style="@style/blackText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:text="@string/black"
  />
```

```
      <RadioButton
        android:id="@+id/blue"
        style="@style/blackText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/blue"
      />
      <RadioButton
        android:id="@+id/red"
        style="@style/blackText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/red"
      />
    </RadioGroup>
  </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"`.

```
  <Button
    android:id="@+id/hideShow"
    style="@style/hideButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingBottom="5dip"
    android:layout_gravity="center_horizontal"
    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" >

    <TextView
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:text="@string/preview"
      style="@style/blackText" />

    <TextView
      android:id="@+id/text"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:textSize="8sp"
      android:text=""
      android:scrollbars="vertical"
      android:maxLines = "100"
      android:paddingLeft="5dip"
      android:paddingTop="5dip"
      style="@style/blackText" />

  </TableRow>

  <TableRow
```

```
      android:layout_width="fill_parent"
      android:layout_height="fill_parent" >
      <TextView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:text="" />

      <TextView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:text="                    " />

    </TableRow>

  </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 :

```
/**
 * Récupère une image depuis les ressources
 * pour les ajouter dans l'interpréteur HTML
*/
public class SmileyGetter implements ImageGetter {
  /* Context de notre activité */
  protected Context context = null;

  public SmileyGetter(Context c) {
    context = c;
  }

  public void setContext(Context context) {
    this.context = context;
  }

  @Override
  /**
   * Donne un smiley en fonction du paramètre d'entrée
   * @param smiley Le nom du smiley à afficher
   */
  public Drawable getDrawable(String smiley) {
    Drawable retour = null;

    // On récupère le gestionnaire de ressources
    Resources resources = context.getResources();

    // Si on désire le clin d'œil…
    if(smiley.compareTo("clin") == 0)
      // … alors on récupère le drawable correspondant
      retour = resources.getDrawable(R.drawable.clin);
    else if(smiley.compareTo("smile") == 0)
      retour = resources.getDrawable(R.drawable.smile);
    else
      retour = resources.getDrawable(R.drawable.heureux);
    // On délimite l'image (elle va de son coin en haut à gauche à son coin en bas à droite)
    retour.setBounds(0, 0, retour.getIntrinsicWidth(), retour.getIntrinsicHeight());
    return retour;
  }
}
```

L'activité

Enfin, le principal, le code de l'activité :

```
public class NotepadActivity extends Activity {
  /* Récupération des éléments du GUI */
  private Button hideShow = null;
  private Slider slider = null;
  private RelativeLayout toHide = null;
  private EditText editer = null;
  private TextView text = null;
  private RadioGroup colorChooser = null;

  private Button bold = null;
  private Button italic = null;
  private Button underline = null;

  private ImageButton smile = null;
  private ImageButton heureux = null;
  private ImageButton clin = null;

  /* Utilisé pour planter les smileys dans le texte */
  private SmileyGetter getter = null;

  /* Couleur actuelle du texte */
  private String currentColor = "#000000";

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

    getter = new SmileyGetter(this);

    // On récupère le bouton pour cacher/afficher le menu
    hideShow = (Button) findViewById(R.id.hideShow);
    // Puis on récupère la vue racine de l'application et on change sa couleur
    hideShow.getRootView().setBackgroundColor(R.color.background);
    // On rajoute un Listener sur le clic du bouton…
    hideShow.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View vue) {
        // … pour afficher ou cacher le menu
        if(slider.toggle())
        {
          // Si le Slider est ouvert…
          // … on change le texte en "Cacher"
          hideShow.setText(R.string.hide);
        }else
        {
          // Sinon on met "Afficher"
          hideShow.setText(R.string.show);
        }
      }
    });
```

```
    // On récupère le menu
    toHide = (RelativeLayout) findViewById(R.id.toHide);
    // On récupère le layout principal
    slider = (Slider) findViewById(R.id.slider);
    // On donne le menu au layout principal
    slider.setToHide(toHide);

    // On récupère le TextView qui affiche le texte final
    text = (TextView) findViewById(R.id.text);
    // On permet au TextView de défiler
    text.setMovementMethod(new ScrollingMovementMethod());

    // On récupère l'éditeur de texte
    editer = (EditText) findViewById(R.id.edit);
    // On ajoute un Listener sur l'appui de touches
    editer.setOnKeyListener(new View.OnKeyListener() {
      @Override
      public boolean onKey(View v, int keyCode, KeyEvent event) {
        // On récupère la position du début de la sélection dans le texte
        int cursorIndex = editer.getSelectionStart();
        // Ne réagir qu'à l'appui sur une touche (et pas au relâchement)
        if(event.getAction() == 0)
          // S'il s'agit d'un appui sur la touche « entrée »
          if(keyCode == 66)
            // On insère une balise de retour à la ligne
            editer.getText().insert(cursorIndex, "<br />");
        return true;
      }
    });
    // On ajoute un autre Listener sur le changement, dans le texte cette fois
    editer.addTextChangedListener(new TextWatcher() {
      @Override
      public void onTextChanged(CharSequence s, int start, int before, int count) {
        // Le Textview interprète le texte dans l'éditeur en une certaine couleur
        text.setText(Html.fromHtml("<font color=\"" + currentColor + "\">" + editer.getText().toStrin
g() + "</font>", getter, null));
      }

      @Override
      public void beforeTextChanged(CharSequence s, int start, int count, int after) {

      }

      @Override
      public void afterTextChanged(Editable s) {

      }
    });



    // On récupère le RadioGroup qui gère la couleur du texte
```

```
colorChooser = (RadioGroup) findViewById(R.id.colors);
// On rajoute un Listener sur le changement de RadioButton sélectionné
colorChooser.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
  @Override
  public void onCheckedChanged(RadioGroup group, int checkedId) {
    // En fonction de l'identifiant du RadioButton sélectionné…
    switch(checkedId)
    {
      // On change la couleur actuelle pour noir
      case R.id.black:
        currentColor = "#000000";
        break;
      // On change la couleur actuelle pour bleu
      case R.id.blue:
        currentColor = "#0022FF";
        break;
      // On change la couleur actuelle pour rouge
      case R.id.red:
        currentColor = "#FF0000";
    }
    /*
     * On met dans l'éditeur son texte actuel
     * pour activer le Listener de changement de texte
    */
    editer.setText(editer.getText().toString());
  }
});

smile = (ImageButton) findViewById(R.id.smile);
smile.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    // On récupère la position du début de la sélection dans le texte
    int selectionStart = editer.getSelectionStart();
    // Et on insère à cette position une balise pour afficher l'image du smiley
    editer.getText().insert(selectionStart, "<img src=\"smile\" >");
  }
});

heureux =(ImageButton) findViewById(R.id.heureux);
heureux.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    // On récupère la position du début de la sélection
    int selectionStart = editer.getSelectionStart();
    editer.getText().insert(selectionStart, "<img src=\"heureux\" >");
  }
});

clin = (ImageButton) findViewById(R.id.clin);
clin.setOnClickListener(new View.OnClickListener() {
  @Override
```

```
  public void onClick(View v) {
    //On récupère la position du début de la sélection
    int selectionStart = editer.getSelectionStart();
    editer.getText().insert(selectionStart, "<img src=\"clin\" >");
  }
});

bold = (Button) findViewById(R.id.bold);
bold.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View vue) {
    // On récupère la position du début de la sélection
    int selectionStart = editer.getSelectionStart();
    // On récupère la position de la fin de la sélection
    int selectionEnd = editer.getSelectionEnd();

    Editable editable = editer.getText();

    // Si les deux positions sont identiques (pas de sélection de plusieurs caractères)
    if(selectionStart == selectionEnd)
      //On insère les balises ouvrante et fermante avec rien dedans
      editable.insert(selectionStart, "<b></b>");
    else
    {
      // On met la balise avant la sélection
      editable.insert(selectionStart, "<b>");
      // On rajoute la balise après la sélection (et après les 3 caractères de la balise <b>)
      editable.insert(selectionEnd + 3, "</b>");
    }
  }
});

italic = (Button) findViewById(R.id.italic);
italic.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View vue) {
    // On récupère la position du début de la sélection
    int selectionStart = editer.getSelectionStart();
    // On récupère la position de la fin de la sélection
    int selectionEnd = editer.getSelectionEnd();

    Editable editable = editer.getText();

    // Si les deux positions sont identiques (pas de sélection de plusieurs caractères)
    if(selectionStart == selectionEnd)
      //On insère les balises ouvrante et fermante avec rien dedans
      editable.insert(selectionStart, "<i></i>");
    else
    {
      // On met la balise avant la sélection
      editable.insert(selectionStart, "<i>");
      // On rajoute la balise après la sélection (et après les 3 caractères de la balise <b>)
```

```
            editable.insert(selectionEnd + 3, "</i>");
          }
        }
      });

      underline = (Button) findViewById(R.id.underline);
      underline.setOnClickListener(new View.OnClickListener() {
       @Override
       public void onClick(View vue) {
         // On récupère la position du début de la sélection
         int selectionStart = editer.getSelectionStart();
         // On récupère la position de la fin de la sélection
         int selectionEnd = editer.getSelectionEnd();

         Editable editable = editer.getText();

         // Si les deux positions sont identiques (pas de sélection de plusieurs caractères)
         if(selectionStart == selectionEnd)
           // On insère les balises ouvrante et fermante avec rien dedans
           editable.insert(selectionStart, "<u></u>");
         else
         {
           // On met la balise avant la sélection
           editable.insert(selectionStart, "<u>");
           // On rajoute la balise après la sélection (et après les 3 caractères de la balise <b>)
           editable.insert(selectionEnd + 3, "</u>");
         }
       }
     });
   }
 }
```

Télécharger le projet (http://www.sdz-files.com/cours/android/notepad.zip)

Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

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 suivante.

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 :

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
  <item android:state_pressed="true"
    android:drawable="@drawable/pressed" />
  <item android:drawable="@drawable/number" />
</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 ? :)

Des widgets plus avancés et des boîtes de dialogue

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 suivante. Dans ce dernier, l'adaptateur va créer un widget pour chaque élément en fonction des informations fournies en amont.

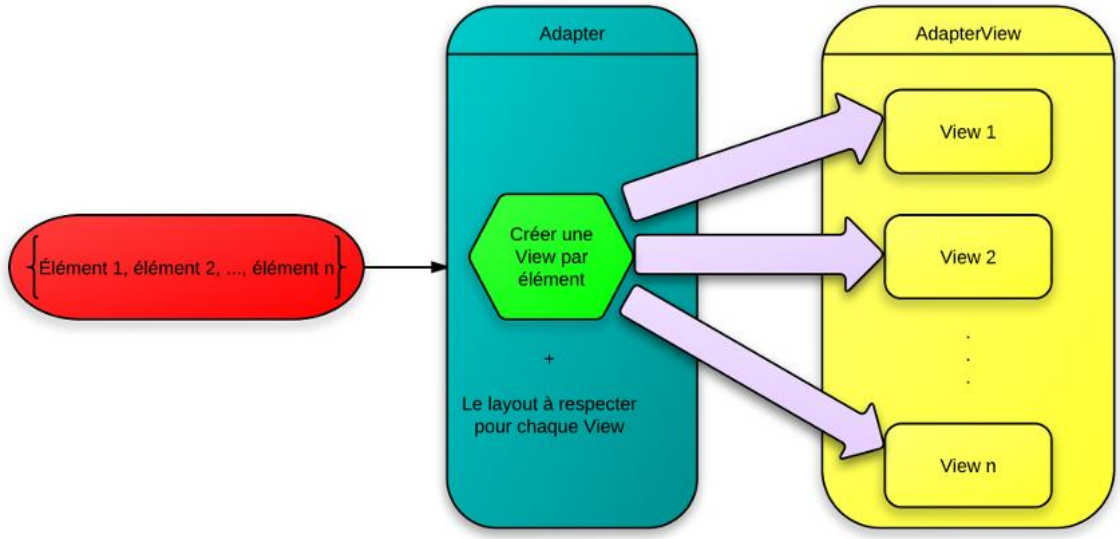


Schéma du fonctionnement des « Adapter » et « AdapterView »

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 (http://developer.android.com/reference/android/widget/Adapter.html) 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 (http://developer.android.com/reference/android/widget/BaseAdapter.html).

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 (http://developer.android.com/reference/android/widget/ArrayAdapter.html) 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 (http://developer.android.com/reference/android/widget/SimpleAdapter.html) 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 :

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.SimpleAdapter;

public class ListesActivity extends Activity {
  ListView vue;

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

    //On récupère une ListView de notre layout en XML, c'est la vue qui représente la liste
    vue = (ListView) findViewById(R.id.listView);

    /*
     * On entrepose nos données dans un tableau qui contient deux colonnes :
     *  - la première contiendra le nom de l'utilisateur
     *  - la seconde contiendra le numéro de téléphone de l'utilisateur
    */
    String[][] repertoire = new String[][]{
      {"Bill Gates", "06 06 06 06 06"},
      {"Niels Bohr", "05 05 05 05 05"},
      {"Alexandre III de Macédoine", "04 04 04 04 04"}};

    /*
     * On doit donner à notre adaptateur une liste du type « List<Map<String, ?> » :
     *  - la clé doit forcément être une chaîne de caractères
     *  - en revanche, la valeur peut être n'importe quoi, un objet ou un entier par exemple,
     *  si c'est un objet, on affichera son contenu avec la méthode « toString() »
     *
     * Dans notre cas, la valeur sera une chaîne de caractères, puisque le nom et le numéro de téléph
one
     * sont entreposés dans des chaînes de caractères
    */
    List<HashMap<String, String>> liste = new ArrayList<HashMap<String, String>>();

    HashMap<String, String> element;
    //Pour chaque personne dans notre répertoire…
    for(int i = 0 ; i < repertoire.length ; i++) {
      //… on crée un élément pour la liste…
      element = new HashMap<String, String>();
      /*
       * … on déclare que la clé est « text1 » (j'ai choisi ce mot au hasard, sans sens technique par
ticulier)
       * pour le nom de la personne (première dimension du tableau de valeurs)…
```

```
        */
        element.put("text1", repertoire[i][0]);
        /*
         * … on déclare que la clé est « text2 »
         * pour le numéro de cette personne (seconde dimension du tableau de valeurs)
        */
        element.put("text2", repertoire[i][1]);
        liste.add(element);
      }

      ListAdapter adapter = new SimpleAdapter(this,
        //Valeurs à insérer
        liste,
        /*
         * Layout de chaque élément (là, il s'agit d'un layout par défaut
         * pour avoir deux textes l'un au-dessus de l'autre, c'est pourquoi on
         * n'affiche que le nom et le numéro d'une personne)
        */
        android.R.layout.simple_list_item_2,
        /*
         * 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 suivante.

Le résultat en image

On a utilisé le constructeur

```
public SimpleAdapter(Context context, List<? extends Map<String, ?>> data, int ressource, String[] from, int[
```

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 :

```
// On crée un adaptateur qui fonctionne avec des chaînes de caractères
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1);
// On rajoute la chaîne de caractères "Pommes"
adapter.add("Pommes");
// 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
int position = adapter.getPosition("Pommes");
// On affiche la valeur et la position de la chaîne de caractères
Toast.makeText(this, "Les " + adapter.getItem(position) + " se trouvent à la position " + position +
 ".", Toast.LENGTH_LONG).show();
// Puis on la supprime, n'en n'ayant plus besoin
adapter.remove("Pommes");
```

Les vues responsables de l'affichage des listes : les `AdapterView`

On trouve la classe AdapterView (http://developer.android.com/reference/android/widget/AdapterView.html) 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 suivante. Le layout de base est `android.R.layout.simple_list_item_1`.

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)` pour un clic simple sur un élément de la liste. La fonction de *callback* associée est `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.
- `void setOnItemLongClickListener (AdapterView.OnItemLongClickListener listener)` 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 :

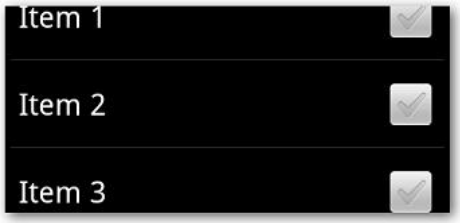
```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
  @Override
  public void onItemClick(AdapterView<?> adapterView,
    View view,
    int position,
    long id) {
      // Que faire quand on clique sur un élément de la liste ?
  }
});
```

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 suivante.

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 suivante.



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 suivante.

À gauche, au démarrage de l'application ; à droite, après avoir appuyé sur le bouton « Envoyer »

Solution

Le layout :

```
<?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
      android:id="@+id/listProg"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:choiceMode="multipleChoice" >
    </ListView>

    <Button
      android:id="@+id/send"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center"
      android:text="Envoyer" />

</LinearLayout>
```

Et le code :

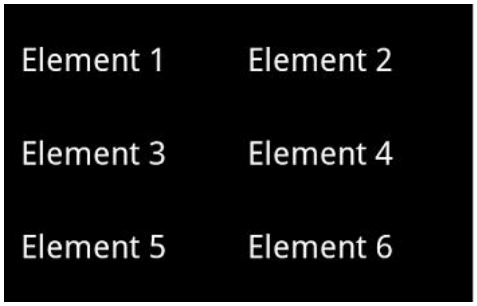
```
package sdz.exemple.selectionMultiple;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.Toast;

public class SelectionMultipleActivity extends Activity {
  /** Affichage de la liste des sexes **/
  private ListView mListSexe = null;
  /** Affichage de la liste des langages connus **/
  private ListView mListProg = null;
  /** Bouton pour envoyer le sondage **/
  private Button mSend = null;

  /** Contient les deux sexes **/
  private String[] mSexes = {"Masculin", "Feminin"};
  /** Contient différents langages de programmation **/
  private String[] mLangages = null;

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

    //On récupère les trois vues définies dans notre layout
    mListSexe = (ListView) findViewById(R.id.listSexe);
    mListProg = (ListView) findViewById(R.id.listProg);
    mSend = (Button) findViewById(R.id.send);

    //Une autre manière de créer un tableau de chaînes de caractères
    mLangages = new String[]{"C", "Java", "COBOL", "Perl"};

    //On ajoute un adaptateur qui affiche des boutons radio (c'est l'affichage à considérer quand on
 ne peut
    //sélectionner qu'un élément d'une liste)
    mListSexe.setAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_single_choi
ce, mSexes));
    //On déclare qu'on sélectionne de base le premier élément (Masculin)
    mListSexe.setItemChecked(0, true);

    //On ajoute un adaptateur qui affiche des cases à cocher (c'est l'affichage à considérer quand on
 peut sélectionner
    //autant d'éléments qu'on veut dans une liste)
    mListProg.setAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_multiple_ch
oice, mLangages));
    //On déclare qu'on sélectionne de base le second élément (Féminin)
    mListProg.setItemChecked(1, true);
```

```
    //Que se passe-t-il dès qu'on clique sur le bouton ?
    mSend.setOnClickListener(new View.OnClickListener() {

      @Override
      public void onClick(View v) {
        Toast.makeText(SelectionMultipleActivity.this, "Merci ! Les données ont été envoyées !", Toas
t.LENGTH_LONG).show();

        //On déclare qu'on ne peut plus sélectionner d'élément
        mListSexe.setChoiceMode(ListView.CHOICE_MODE_NONE);
        //On affiche un layout qui ne permet pas de sélection
        mListSexe.setAdapter(new ArrayAdapter<String>(SelectionMultipleActivity.this, android.R.layou
t.simple_list_item_1,
                        mSexes));

        //On déclare qu'on ne peut plus sélectionner d'élément
        mListProg.setChoiceMode(ListView.CHOICE_MODE_NONE);
        //On affiche un layout qui ne permet pas de sélection
        mListProg.setAdapter(new ArrayAdapter<String>(SelectionMultipleActivity.this, android.R.layou
t.simple_list_item_1, mLangages));

        //On désactive le bouton
        mSend.setEnabled(false);
      }
    });
  }
}
```

Dans un tableau : `GridView`

On peut utiliser la classe `GridView (http://developer.android.com/reference/android/widget/GridView.html)` 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 suivante.



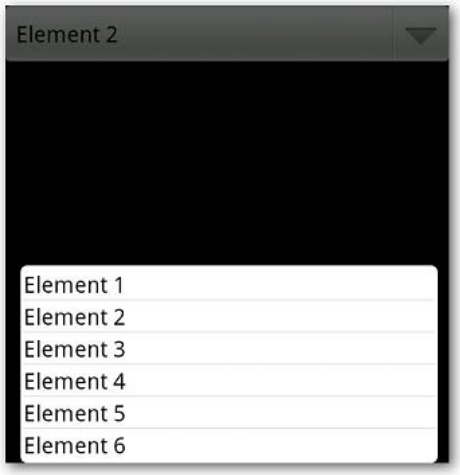
Les éléments sont placés sur une grille

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 (http://developer.android.com/reference/android/widget/Spinner.html)` 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 suivante montre ce qui arrive si on ne définit pas de mise en page pour la liste d'éléments.



Aucune mise en page pour la liste d'éléments n'a été définie

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 :

```
import java.util.ArrayList;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.Spinner;

public class TutoListesActivity extends Activity {
  private Spinner liste = null;

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

    liste = (Spinner) findViewById(R.id.spinner1);
    List<String> exemple = new ArrayList<String>();
    exemple.add("Element 1");
    exemple.add("Element 2");
    exemple.add("Element 3");
    exemple.add("Element 4");
    exemple.add("Element 5");
    exemple.add("Element 6");

    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_ite
m, exemple);
    //Le layout par défaut est android.R.layout.simple_spinner_dropdown_item
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    liste.setAdapter(adapter);
  }
}
```

Ce code donnera la figure suivante.

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. :D

> 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 :

```
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 suivante.

Dans cet exemple, la méthode « getView » a été appelée sur les sept lignes visibles, mais pas sur les autres lignes de la liste

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 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 :

```
LayoutInflater mInflater;
String[] mListe;

public View getView(int position, View convertView, ViewGroup parent) {
  TextView vue = (TextView) mInflater.inflate(R.layout.ligne, null);

  vue.setText(mListe[position]);

  return vue;
}
```

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 :

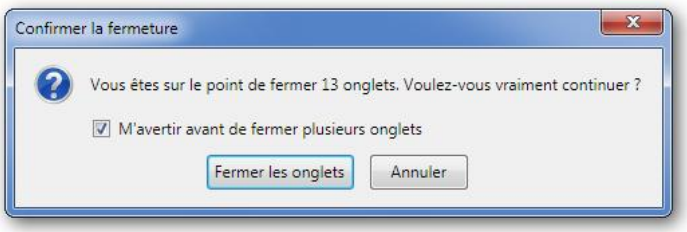
```
static class ViewHolder {
  public TextView mNom;
  public TextView mNumero;
  public ImageView mPhoto;
}
```

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 suivante.



Firefox demande confirmation avant de se fermer si plusieurs onglets sont ouverts

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.

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 (http://www.siteduzero.com/tutoriel-3-494425-1-premiere-application-celle-qui-vous-rapportera-des-millions.html#ss_part_2), une activité qu'on ne voit plus que partiellement est suspendue.

> Les boîtes de dialogue héritent de la classe `Dialog (http://developer.android.com/reference/android/app/Dialog.html)` 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;

    case IDENTIFIANT_BOITE_DEUX :
      // On construit la seconde boîte de dialogue, que l'on insère dans « box »
      break;
  }
  return box;
}
```

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 suivante.

Après le cinquième clic

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

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

public class StringExampleActivity extends Activity {
  private Button bouton;
  //Variable globale, au-dessus de cette valeur c'est l'autre boîte de dialogue qui s'exprime
  private final static int ENERVEMENT = 4;
  private int compteur = 0;

  private final static int ID_NORMAL_DIALOG = 0;
  private final static int ID_ENERVEE_DIALOG = 1;

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

    bouton = (Button) findViewById(R.id.bouton);
    bouton.setOnClickListener(boutonClik);
  }

  private OnClickListener boutonClik = new OnClickListener() {
    @Override
    public void onClick(View v) {
      // Tant qu'on n'a pas invoqué la première boîte de dialogue 5 fois
      if(compteur < ENERVEMENT) {
        //on appelle la boîte normale
        compteur ++;
        showDialog(ID_NORMAL_DIALOG);
      } else
        showDialog(ID_ENERVEE_DIALOG);
    }
  };

  /*
   * Appelée qu'à la première création d'une boîte de dialogue
   * Les fois suivantes, on se contente de récupérer la boîte de dialogue déjà créée…
   * Sauf si la méthode « onPrepareDialog » modifie la boîte de dialogue.
  */
  @Override
  public Dialog onCreateDialog (int id) {
    Dialog box = null;
    switch(id) {
    // Quand on appelle avec l'identifiant de la boîte normale
    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 text
e 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 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"
      android:text="Je suis une jolie boîte de dialogue !"/>
  </LinearLayout>
</LinearLayout>
```

On peut associer ce fichier XML à une boîte de dialogue comme on le fait pour une activité :

```
Dialog box = new Dialog(this);
box.setContentView(R.layout.dialog);
box.setTitle("Belle ! On dirait un mot inventé pour moiiii !");
```

Sur le résultat, visible à la figure suivante, 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.

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.OnClickListener 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.OnClickListener listener)`. Ce bouton se trouvera entre les deux autres boutons.

- `AlertDialog.Builder setNegativeButton (text, DialogInterface.OnClickListener 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 :

Méthode

`AlertDialog.Builder setItems (CharSequence[] items, DialogInterface.OnClickListener listener)`

Méthode

```
AlertDialog.Builder setSingleChoiceItems (CharSequence[] items, int checkedItem, DialogInterface.OnClickList
```

Méthode

```
AlertDialog.Builder setMultipleChoiceItems (CharSequence[] items, boolean[] checkedItems, DialogInterface.On
```


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 (http://developer.android.com/reference/java/util/Calendar.html) 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 !
- `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 (http://developer.android.com/reference/android/widget/DatePicker.html)`. 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_cha`
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 suivante.

Insertion d'horaires

Pour choisir un horaire, on utilise `TimePicker (http://developer.android.com/reference/android/widget/TimePicker.html)`, classe pas très contraignante puisqu'elle fonctionne comme `DatePicker` ! Alors qu'il n'est pas possible de définir un horaire 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(hourOfDa
y) + ":" + String.valueOf(minute), Toast.LENGTH_SHORT).show();
  }
});
```

Ce qui donne la figure suivante.

Sachez enfin que vous pouvez utiliser de manière équivalente des boîtes de dialogue qui contiennent ces widgets. Ces boîtes s'appellent `DatePickerDialog` et `TimePickerDialog`.

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 (http://developer.android.com/reference/android/widget/AnalogClock.html), soit avec une horloge numérique (comme sur une montre sans aiguilles) qui s'appelle `DigitalClock`, les deux visibles à la figure suivante.

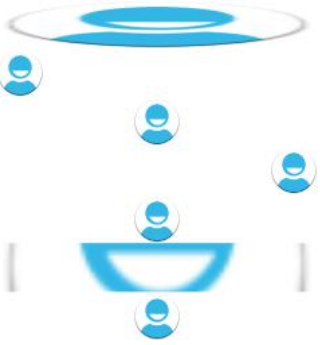
À gauche une « AnalogClock » et à droite une « DigitalClock »

Afficher des images

Le widget de base pour afficher une image est `ImageView (http://developer.android.com/reference/android/widget/ImageView.html)`. 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 suivante.



- `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`.
- `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 (http://developer.android.com/reference/android/graphics/drawable/Drawable.html)` et un `Bitmap (http://developer.android.com/reference/android/graphics/Bitmap.html)` ?*

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 (http://developer.android.com/reference/android/widget/ImageButton.html)`, 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` :

```
<?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" >

  <AutoCompleteTextView
    android:id="@+id/complete"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

</LinearLayout>
```

Ensuite, je déclare l'activité `AutoCompletionActivity` suivante :

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;

public class AutoCompletionActivity extends Activity {
  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_it
em_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 suivante.

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.CommaTokenizer`, 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).

- 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`.

Gestion des menus de l’application

À une époque pas si lointaine, tous les terminaux sous Android possédaient un bouton physique pour afficher le menu. Cependant, cette pratique est devenue un peu 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 du système d'exploitation, qui permet d'ouvrir un menu. En sorte, on peut dire que tous les utilisateurs sont touchés par la présence d'un menu.

En tout cas, un menu est un endroit privilégié pour placer certaines fonctions tout en économisant notre précieux espace dans la fenêtre. Vous pouvez par exemple faire en sorte que ce menu ouvre la page des options, ou au contraire vous ramène à la page d'accueil.

Il existe deux sortes de menu dans Android :

- Le menu d'options, celui 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.
- Les menus contextuels, vous savez, ces menus 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.

Et ces deux menus peuvent bien entendu contenir des sous-menus, qui peuvent contenir des sous-menus, etc. Encore une fois, on va devoir manipuler des fichiers XML mais, franchement, vous êtes devenus des experts maintenant, non ? :p

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 ? :euh:), et on ne peut pas vraiment le personnaliser avec des attributs, ce qui donne la majorité du temps :

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <!-- Code -->
</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 es\` 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.

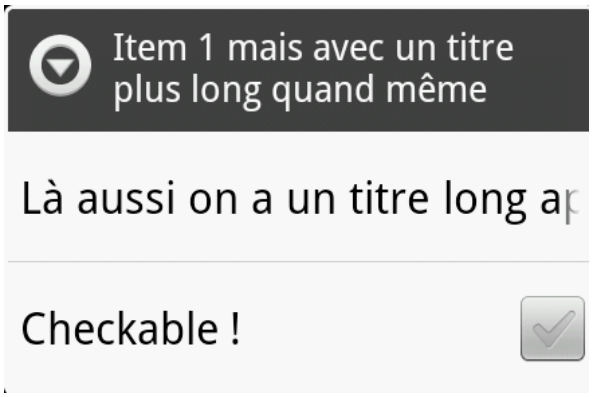
Le problème est que l'espace consacré à un menu est assez réduit, comme toujours sur un périphérique portable, remarquez. Afin de gagner un peu de place, il est possible d'avoir un `<item>` qui ouvre un sous-menu, et ce sous-menu sera à traiter comme tout autre menu. On lui mettra donc des items aussi. En d'autres termes, la syntaxe est celle-ci :

```
<item>
  <menu>
    <item />
    <!-- d'autres items-->
    <item />
  </menu>
</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 deux figures suivantes : la première utilise `android:titleCondensed="Item 1"`, la deuxième `android:title="Item 1 mais avec un titre plus long quand même"`.

Le titre est condensé



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 :

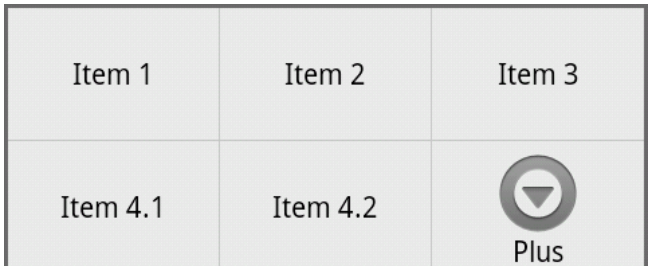
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
  <item android:id="@+id/item1" android:title="Item 1"></item>
  <item android:id="@+id/item2" android:titleCondensed="Item 2" android:title="Item 2 mais avec un no
m assez long quand même">
    <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;
}
```

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 suivante.



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` :

```
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 :

```
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. :o 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, comme celle-ci qui est officielle (http://developer.android.com/resources/samples/ActionBarCompat/index.html) ou celle-là qui est puissante (http://actionbarsherlock.com/index.html). Je vous invite à les découvrir par vous-mêmes. ;)

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.

- 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.

Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

Création de vues personnalisées

Vous savez désormais l'essentiel pour développer de belles interfaces graphiques fonctionnelles, et en théorie vous devriez être capables de faire tout ce que vous désirez. Cependant, il vous manque encore l'outil ultime qui vous permettra de donner vie à tous vos fantasmes les plus extravagants : être capables de produire vos propres vues et ainsi avoir le contrôle total sur 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 pour le faire coïncider avec vos besoins.
- 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, il est toujours possible de 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.

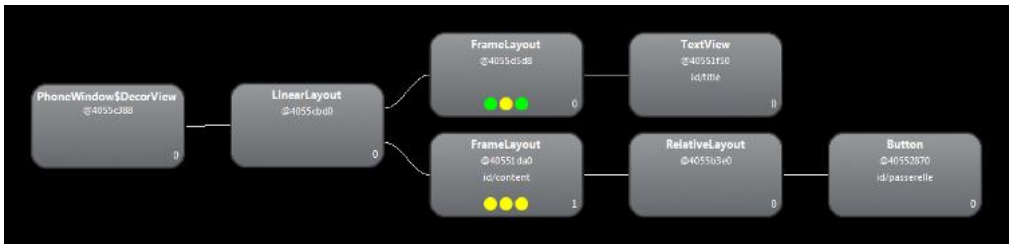
Par exemple, si vous dites que vous disposez d'une vue qui occupe toute seule son layout parent et que vous lui donnez l'instruction `FILL_PARENT`, alors les dimensions réelles seront identiques aux dimensions demandées puisque la vue peut occuper tout le parent. En revanche, s'il y a plusieurs vues 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`.

```
<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" >

  <Button
    android:id="@+id/passerelle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true" />

</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 suivante.



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. :waw: 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 :

```
int taille = MeasureSpec.makeMeasureSpec(10, MeasureSpec.AT_MOST);
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 :

```
vue.getWidth();
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 :p). 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 suivante.

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 suivante. 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.

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 à cette adresse (http://developer.android.com/reference/android/view/View.html) une liste intégrale des méthodes que vous pouvez implémenter.

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) {
    super(context, attrs, defStyle);
        init();
  }

  /**
   * Constructeur utilisé pour inflater sans style
   */
  public ColorButton(Context context, AttributeSet attrs) {
        super(context, attrs);
        init();
  }

  /**
   * Constructeur utilisé pour construire dans le code
   */
  public ColorButton(Context context) {
        super(context);
        init();
  }

  private void init() {
        // Je récupère mes ressources
        Resources res = getResources();
        // Et dans ces ressources je récupère mon tableau de couleurs
        mCouleurs = res.obtainTypedArray(R.array.colors);

        setText("Changer de couleur");
```

```
    }

    /* … */

}
```

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.

```
/** Rectangle qui délimite le dessin */
private Rect mRect = null;

@Override
protected void onLayout (boolean changed, int left, int top, int right, int bottom)
{
  //Si le layout a changé
  if(changed)
    //On redessine un nouveau carré en fonction des nouvelles dimensions
      mRect = new Rect(Math.round(0.5f * getWidth() - 50),
                       Math.round(0.75f * getHeight() - 50),
                       Math.round(0.5f * getWidth() + 50),
                       Math.round(0.75f * getHeight() + 50));
  //Ne pas oublier
  super.onLayout(changed, left, top, right, bottom);
}
```

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.

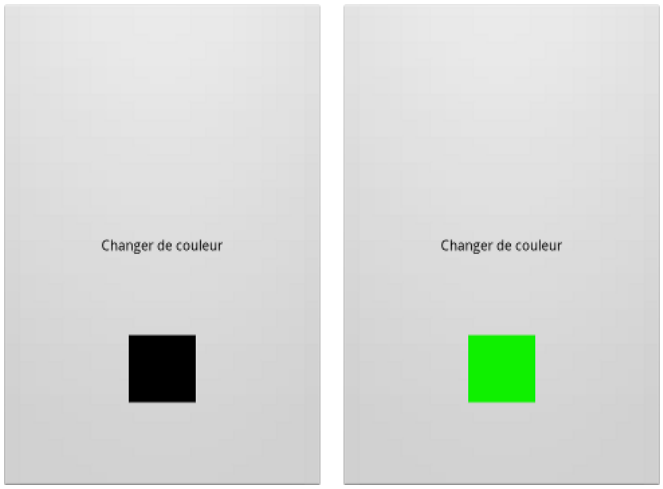
```
/** Outil pour peindre */
private Paint mPainter = new Paint(Paint.ANTI_ALIAS_FLAG);

@Override
public boolean onTouchEvent(MotionEvent event) {
  // Uniquement si on appuie sur le bouton
  if(event.getAction() == MotionEvent.ACTION_DOWN) {
    // On passe à la couleur suivante
    position ++;
    // Évite de dépasser la taille du tableau
    // (dès qu'on arrive à la longueur du tableau, on repasse à 0)
    position %= mCouleurs.length();

    // Change la couleur du pinceau
    mPainter.setColor(mCouleurs.getColor(position, -1));

    // Redessine la vue
    invalidate();
  }
  // Ne pas oublier
  return super.onTouchEvent(event);
}
```

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.

```
@Override
protected void onDraw(Canvas canvas) {
  // Dessine le rectangle à l'endroit voulu avec la couleur voulue
  canvas.drawRect(mRect, mPainter);
  // Ne pas oublier
  super.onDraw(canvas);
}
```

> 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 suivante.



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)

Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

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 !

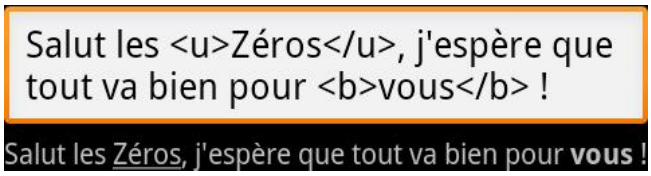
Logiquement, cette vue sera un assemblage d'autres vues et par conséquent ne sera pas un widget — qui ne peut pas contenir d'autres vues — mais bien un layout, elle devra donc dériver de `ViewGroup` ou d'une sous-classe de `ViewGroup`.

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
    mEdit = new EditText(getContext());
    // Le texte sera de type web et peut être long
    mEdit.setInputType(InputType.TYPE_TEXT_VARIATION_WEB_EDIT_TEXT | InputType.TYPE_TEXT_FLAG_MULTI_L
INE);
```

```
        // Il fera au maximum dix lignes
        mEdit.setMaxLines(10);
        // On interdit le scrolling horizontal pour des questions de confort
        mEdit.setHorizontallyScrolling(false);

        // Listener qui se déclenche dès que le texte dans l'EditText change
        mEdit.addTextChangedListener(new TextWatcher() {

          // À chaque fois que le texte est édité
          @Override
          public void onTextChanged(CharSequence s, int start, int before, int count) {
            // On change le texte en Spanned pour que les balises soient interprétées
            mText.setText(Html.fromHtml(s.toString()));
          }

          // Après que le texte a été édité
          @Override
          public void beforeTextChanged(CharSequence s, int start, int count, int after) {

          }

          // Après que le texte a été édité
          @Override
          public void afterTextChanged(Editable s) {

          }
        });

        mText = new TextView(getContext());
        mText.setText("");

        // Puis on rajoute les deux widgets à notre vue
        addView(mEdit, new LinearLayout.LayoutParams(fill, wrap));
        addView(mText, new LinearLayout.LayoutParams(fill, wrap));
    }
}
```

Ce qui donne, une fois intégré, la figure suivante.



Le rendu du code

Mais j'aurais très bien pu passer par un fichier XML aussi ! Voici comment j'aurais pu faire :

```
<?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.

Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

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>
  <declare-styleable name="ChessBoardView">
    <!-- L'attribut d'identifiant "colorOne" est de type "color" -->
    <attr name="colorOne" format="color"/>
    <attr name="colorTwo" format="color"/>
  </declare-styleable>
</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 :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:sdzName="http://schemas.android.com/apk/res/sdz.chapitreDeux.chessBoard"

  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <sdz.chapitreDeux.chessBoard.ChessBoardView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    sdzName:colorOne="#FF0000"
    sdzName:colorTwo="#00FF00" />
</LinearLayout>
```

Il me suffit maintenant de récupérer les attributs comme nous l'avions fait précédemment :

```
// attrs est le paramètre qui contient les attributs de notre objet en XML
public ChessBoardView(Context context, AttributeSet attrs, int defStyle) {
  super(context, attrs, defStyle);
  init(attrs);
}

// idem
public ChessBoardView(Context context, AttributeSet attrs) {
  super(context, attrs);
  init(attrs);
}

private void init(AttributeSet attrs) {
  // Je récupère mon tableau d'attributs depuis le paramètre que m'a donné le constructeur
  TypedArray attr = getContext().obtainStyledAttributes(attrs, R.styleable.ChessBoardView);
  // Il s'agit d'un TypedArray qu'on sait déjà utiliser, je récupère la valeur de la couleur, 1 ou -1
 si on ne la trouve pas
  int tmpOne = attr.getColor(R.styleable.ChessBoardView_colorOne, -1);
  // Je récupère la valeur de la couleur, 2 ou -1 si on ne la trouve pas
  int tmpTwo = attr.getColor(R.styleable.ChessBoardView_colorTwo, -1);
  init(tmpOne, tmpTwo);
}
```

`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.

```
/**
 * Calcule la bonne mesure sur un axe uniquement
 * @param spec - Mesure sur un axe
 * @param screenDim - Dimension de l'écran sur cet axe
 * @return la bonne taille sur cet axe
 */
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);
}
```

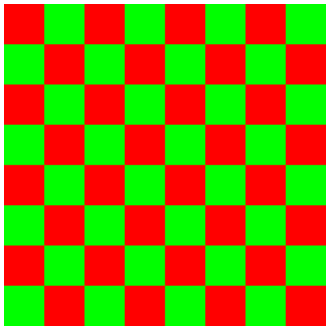
Toujours avoir dans son implémentation de `onMeasure` un appel à la méthode `void setMeasuredDimension (int measuredWidth, int measuredHeight)`, sinon votre vue vous renverra une exception.

`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) {
  // Largeur de la vue
  int width = getWidth();
  // Hauteur de la vue
  int height = getHeight();

  int step = 0, min = 0;
  // La taille minimale entre la largeur et la hauteur
  min = Math.min(width, height);

  // Comme on ne veut que 8 cases par ligne et 8 lignes, on divise la taille par 8
  step = min / 8;

  // Détermine quand on doit changer la couleur entre deux cases
  boolean switchColor = true;
  for(int i = 0 ; i < min ; i += step) {
    for(int j = 0 ; j < min ; j += step) {
      if(switchColor)
        canvas.drawRect(i, j, i + step, j + step, mPaintTwo);
      else
        canvas.drawRect(i, j, i + step, j + step, mPaintOne);
      // On change de couleur à chaque ligne…
      switchColor = !switchColor;
    }
    // … et à chaque case
    switchColor = !switchColor;
  }
}
```

Ce qui peut donner la figure suivante.



Le choix des couleurs est discutable

- 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.

Préambule : quelques concepts avancés

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 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` :

```
xmlns:android="http://schemas.android.com/apk/res/android"
```

Puis, on déclare dans quel package se trouve notre application :

```
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 sur cette page (http://developer.android.com/guide/topics/manifest/uses-feature-element.html#features-reference) la liste exhaustive des valeurs que peut prendre `android:name`.

`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 :

```
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 :

```
<activity android:theme="@android:style/Theme.NoTitleBar">
```

Vous en retrouverez d'autres sur la documentation (http://developer.android.com/reference/android/R.style.html).

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 :

```
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
</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 suivante). 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 ».

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 cette page (http://developer.android.com/reference/android/Manifest.permission.html) une liste des permissions qui existent déjà.

Ainsi, pour demander un accès à internet, on indiquera la ligne :

```
<uses-permission android:name="android.permission.INTERNET" />
```

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 :

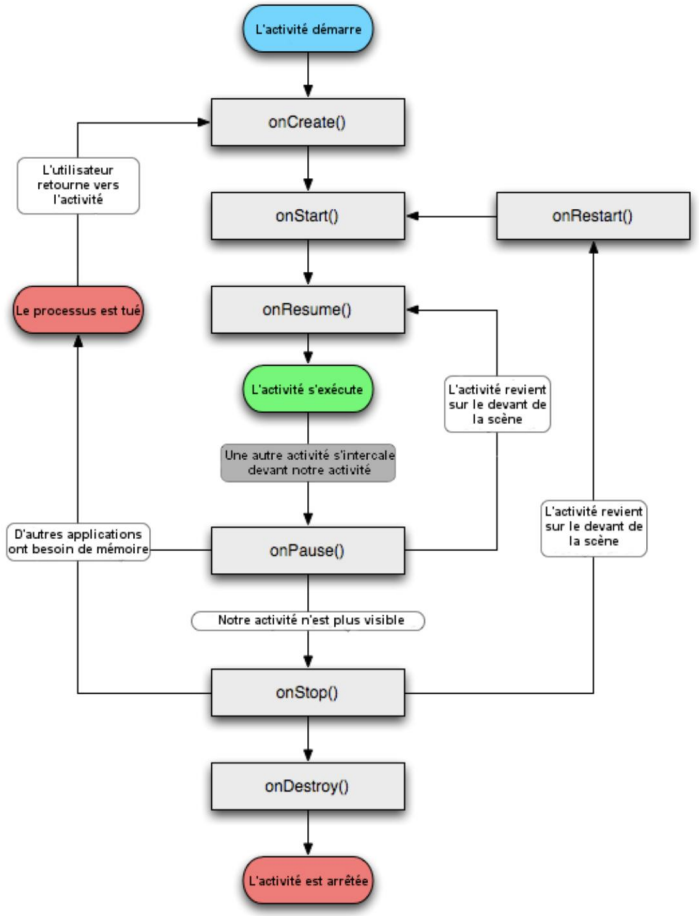
```
<permission android:name="string"
            android:label="string resource"
            android:description="string resource"
            android:icon="drawable resource"
            android:protectionLevel=XXX />
```

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 suivante.



Le cycle de vie d'une activité

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**.

> 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'application 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 (http://developer.android.com/reference/android/os/Bundle.html)` 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 :

```
private final static RESULTAT_DU_CALCUL = 0;

@Override
protected void onSaveInstanceState (Bundle bundle)
{
  super.onSaveInstanceState(bundle);
  bundle.putInt(RESULTAT_DU_CALCUL, 10);
  /*
   * On pourra le récupérer plus tard avec
   * int resultat = bundle.getInt(RESULTAT_DU_CALCUL);
  */
}
```

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 ?

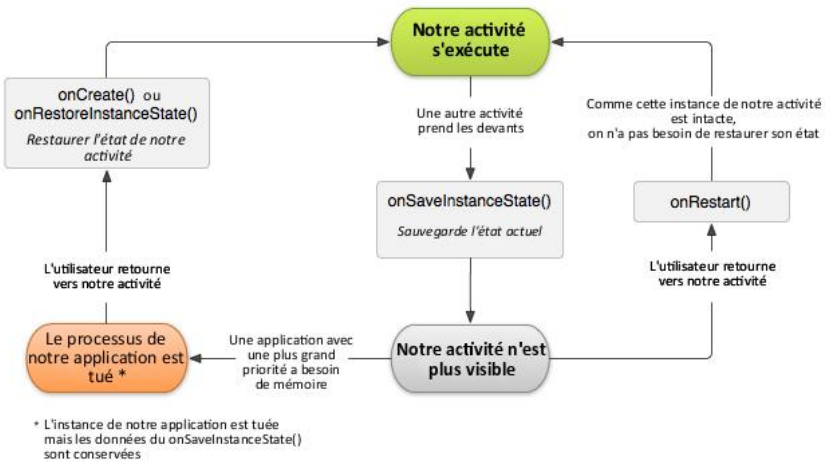
En ce qui concerne Android, on n'utilise pas la sérialisation standard de Java, avec l'interface `Java.io.Serializable`, parce que ce processus est trop lent. Or, quand nous essayons de faire communiquer des composants, il faut que l'opération se fasse de manière rapide. C'est pourquoi on utilise un 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.



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 :

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

  <EditText
    android:id="@+id/editText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >

  </EditText>

  <EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

</LinearLayout>
```

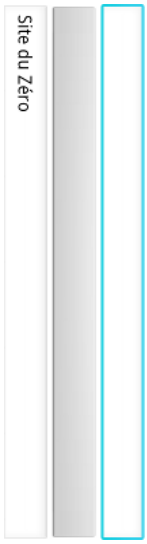
Seul un de ces `EditText` possède un identifiant. Ensuite, on fait un layout presque similaire, mais avec un quantificateur pour qu'il ne s'affiche qu'en mode paysage :

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

  <EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >

  </EditText>

  <Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />

  <EditText
    android:id="@+id/editText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

</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 suivante.

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 suivante.



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 `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 :

```
@Override
public Object onRetainNonConfigurationInstance() {
  // La classe « DonneesConservees » permet de contenir tous les objets voulus
  // Et la méthode "constituerDonnees" va construire un objet
  // En fonction de ce que devra savoir la nouvelle instance de l'activité
  DonneesConservees data = constituerDonnees();
  return data;
}
```

Enfin, il est possible de récupérer cet objet dans le `onCreate(Bundle)` à l'aide de la méthode `public Object getLastNonConfigurationInstance()` :

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

  DonneesConservees data = (DonneesConservees) getLastNonConfigurationInstance();
  // S'il ne s'agit pas d'un retour depuis un changement de configuration, alors data est null
  if(data == null)
    …
}
```

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 :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="fr.sdz.configuration.change"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="7"
    android:targetSdkVersion="15" />

  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name=".MainActivity"
      android:label="@string/title_activity_main"
      android:configChanges="orientation"
      android:screenOrientation="portrait" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

Voilà, maintenant vous aurez beau tourner le terminal dans tous les sens, l'application restera toujours orientée de la même manière.

- 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 par l'élément `uses-permission` ou que vous en créez par 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.

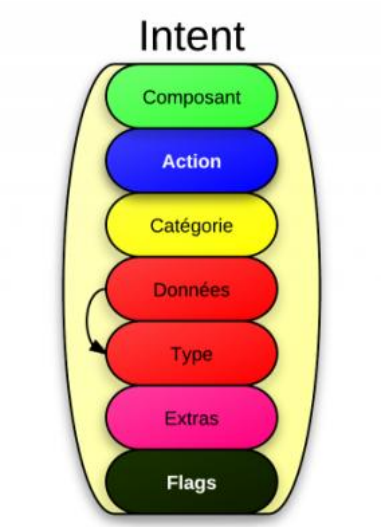
La communication entre composants

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`. Par exemple, si l'utilisateur clique sur un numéro de téléphone dans votre application, peut-être souhaiteriez-vous que le téléphone appelle le numéro demandé. Avec un intent, vous allez dire à tout le système que vous avez un numéro qu'il faut appeler, et c'est le système qui fera en sorte de trouver les applications qui peuvent le prendre en charge. 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 suivante.



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.

```
// On déclare une constante dans la classe FirstClass
public final static String NOMS = "sdz.chapitreTrois.intent.examples.NOMS";

…

// Autre part dans le code
Intent i = new Intent();
String[] noms = new String[] {"Dupont", "Dupond"};
i.putExtra(FirstClass.NOMS, noms);

// Encore autre part
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 (http://developer.android.com/reference/android/os/Parcelable.html)`. Si on devait entrer dans les détails, sachez qu'un `Parcelable` est un objet qui sera transmis à un `Parcel (http://developer.android.com/reference/android/os/Parcel.html)`, 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 (http://developer.android.com/reference/java/io/FileDescriptor.html)`. 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 :

```
import android.os.Parcel;
import android.os.Parcelable;

public class Contact implements Parcelable{
  private String mNom;
  private String mPrenom;
  private int mNumero;

  public Contact(String pNom, String pPrenom, int pNumero) {
    mNom = pNom;
    mPrenom = pPrenom;
    mNumero = pNumero;
  }

  @Override
  public int describeContents() {
    //On renvoie 0, car notre classe ne contient pas de FileDescriptor
    return 0;
  }

  @Override
  public void writeToParcel(Parcel dest, int flags) {
    // On ajoute les objets dans l'ordre dans lequel on les a déclarés
    dest.writeString(mNom);
    dest.writeString(mPrenom);
    dest.writeInt(mNumero);
  }
}
```

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` :

```
public static final Parcelable.Creator<Contact> CREATOR = new Parcelable.Creator<Contact>() {
  @Override
  public Contact createFromParcel(Parcel source) {
    return new Contact(source);
  }

  @Override
  public Contact[] newArray(int size) {
    return new Contact[size];
  }
};

public Contact(Parcel in) {
  mNom = in.readString();
  mPrenom = in.readString();
  mNumero = in.readInt();
}
```

Enfin, comme n'importe quel autre objet, on peut l'ajouter dans un intent avec `putExtra` et on peut le récupérer avec `getParcelableExtra`.

```
Intent i = new Intent();
Contact c = new Contact("Dupont", "Dupond", 06);
i.putExtra("sdz.chapitreTrois.intent.examples.CONTACT", c);

// Autre part dans le code

Contact c = i.getParcelableExtra("sdz.chapitreTrois.intent.examples.CONTACT");
```

Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

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
        Intent secondeActivite = new Intent(MainActivity.this, IntentExample.class);

        // On rajoute un extra
        secondeActivite.putExtra(AGE, 31);

        // Puis on lance l'intent !
        startActivity(secondeActivite);
      }
    });
  }
}
```

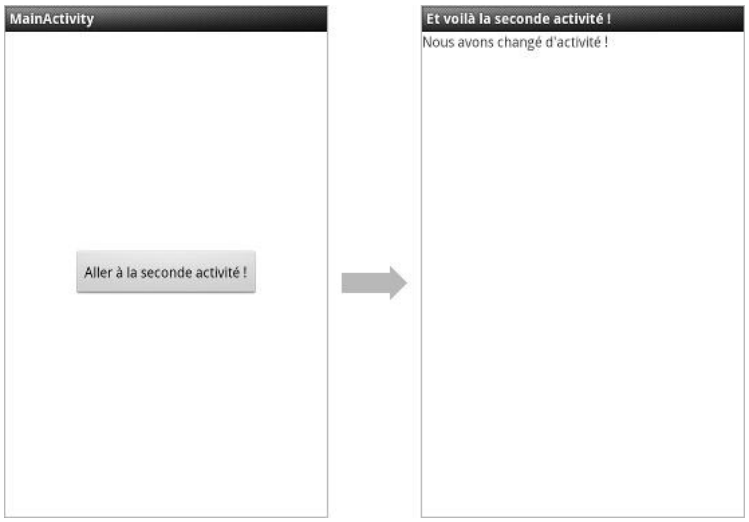
La seconde activité ne fera rien de particulier, si ce n'est afficher un layout différent :

```
package sdz.chapitreTrois.intent.example;

import android.app.Activity;
import android.os.Bundle;

public class IntentExample extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.layout_example);

    // On récupère l'intent qui a lancé cette activité
    Intent i = getIntent();

    // Puis on récupère l'âge donné dans l'autre activité, ou 0 si cet extra n'est pas dans l'intent
    int age = i.getIntExtra(MainActivity.AGE, 0);

    // S'il ne s'agit pas de l'âge par défaut
    if(age != 0)
      // Traiter l'âge
      age = 2;
  }
}
```

Enfin, n'oubliez pas de préciser dans le Manifest que vous avez désormais deux activités au sein de votre application :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sdz.chapitreTrois.intent.example"
  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=".MainActivity"
      android:label="@string/title_activity_main" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <activity
      android:name=".IntentExample"
      android:label="@string/title_example" >
    </activity>
  </application>

</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 suivante.



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 :

```
package sdz.chapitreTrois.intent.example;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {
  private Button mPasserelle = null;
  // L'identifiant de notre requête
  public final static int CHOOSE_BUTTON_REQUEST = 0;
  // L'identifiant de la chaîne de caractères qui contient le résultat de l'intent
  public final static String BUTTONS = "sdz.chapitreTrois.intent.example.Boutons";

  @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) {
        Intent secondeActivite = new Intent(MainActivity.this, IntentExample.class);
        // On associe l'identifiant à notre intent
        startActivityForResult(secondeActivite, CHOOSE_BUTTON_REQUEST);
      }
    });
  }

  @Override
  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // On vérifie tout d'abord à quel intent on fait référence ici à l'aide de notre identifiant
    if (requestCode == CHOOSE_BUTTON_REQUEST) {
      // On vérifie aussi que l'opération s'est bien déroulée
      if (resultCode == RESULT_OK) {
        // On affiche le bouton qui a été choisi
        Toast.makeText(this, "Vous avez choisi le bouton " + data.getStringExtra(BUTTONS), Toast.LENG
TH_SHORT).show();
      }
    }
  }
}
```

Alors que la seconde activité devient :

```
package sdz.chapitreTrois.intent.example;

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

public class IntentExample extends Activity {
  private Button mButton1 = null;
  private Button mButton2 = null;

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

    mButton1 = (Button) findViewById(R.id.button1);
    mButton1.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Intent result = new Intent();
        result.putExtra(MainActivity.BUTTONS, "1");
        setResult(RESULT_OK, result);
        finish();
      }
    });

    mButton2 = (Button) findViewById(R.id.button2);
    mButton2.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Intent result = new Intent();
        result.putExtra(MainActivity.BUTTONS, "2");
        setResult(RESULT_OK, result);
        finish();
      }
    });
  }
}
```

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 suivante.

Un Toast affiche quel bouton a été pressé

Fatigué(e) de lire sur un écran ? Découvrez ce cours en livre (http://62.4.17.167/informatique/book/creez-des-applications-pour-android).

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.

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. 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`
- `/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) :

```
<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 :

```
Uri sms = Uri.parse("sms:0606060606");
```

Mais je peux aussi indiquer plusieurs destinataires et un corps pour ce message :

```
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 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. Vous trouverez une liste non exhaustive des types MIME sur cette page Wikipédia (http://fr.wikipedia.org/wiki/Type_MIME).

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` :

```
public final static String ACTION_PERSO = "sdz.chapitreTrois.intent.action.PERSO";
```

Voici quelques actions natives parmi les plus usitées :

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	/

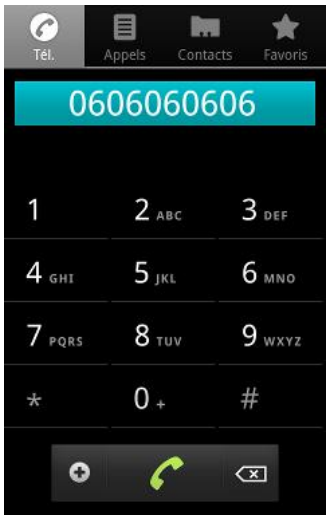
Intitulé	Action	Entrée attendue	Sortie attendue
`ACTION_VIEW`	Permet de visionner une donnée	Un peu tout. Une adresse e-mail sera visionnée dans l'application pour les e-mails, un numéro de téléphone dans le composeur, une adresse internet dans le navigateur, etc.	/
ACTION_WEB_SEARCH	Effectuer une recherche 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	/

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 :

```
mPasserelle.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    Uri telephone = Uri.parse("tel:0606060606");
    Intent secondeActivite = new Intent(Intent.ACTION_DIAL, telephone);
    startActivity(secondeActivite);
  }
});
```

Ce qui donne, une fois que j'appuie dessus, la figure suivante.



Le composeur téléphonique est lancé avec le numéro souhaité

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 ?

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.

```
<activity>
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.SENDTO" />
  </intent-filter>
</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. Par exemple, pour l'action `DESEMBROUILLER`, on pourrait utiliser `sdz.chapitreQuatre.action.DESEMBROUILLER`.

```
<activity>
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.SEARCH" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="com.sdz.intent.category.DESEMBROUILLEUR" />
  </intent-filter>
</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`.

Voici les principales catégories par défaut fournies par Android :

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).

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 :

```
<data android:mimeType="text/plain" android:scheme="http" />
<data android:mimeType="text/plain" android:scheme="https" />
```

> *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 (http://developer.android.com/reference/android/content/pm/PackageManager.html). 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)` :

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("tel:0606060606"));

PackageManager manager = getPackageManager();

ComponentName component = intent.resolveActivity(manager);
// On vérifie que component n'est pas null
if(component != null)
  //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.

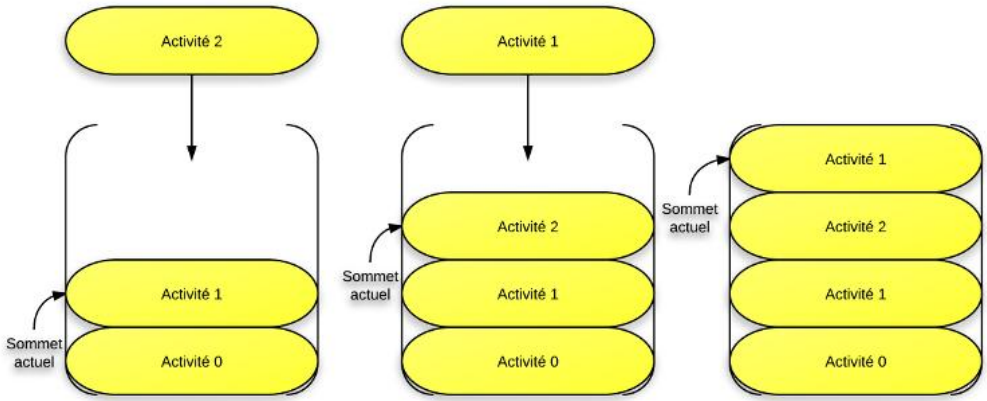
Je vous avais présenté au tout début du tutoriel la pile des activités. En effet, comme on ne peut avoir qu'une activité visible à la fois, les activités étaient présentées dans une pile où il était possible d'ajouter ou d'enlever un élément au sommet afin de changer l'activité consultée actuellement. 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 suivante.



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 :

```
<activity android:taskAffinity="sdz.chapitreTrois.intent.exemple.tacheUn" />
<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 (http://developer.android.com/reference/android/content/BroadcastReceiver.html). De plus, quand vous allez déclarer ce composant dans votre Manifest, il faudra que vous annonciez qu'il s'agit d'un broadcast receiver :

```
<receiver android:name="CoucouReceiver">
  <intent-filter>
    <action android:name="sdz.chapitreTrois.intent.action.coucou" />
  </intent-filter>
</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 :

```
public class CoucouReceiver extends BroadcastReceiver {
  private static final String NOM_USER = "sdz.chapitreTrois.intent.extra.NOM";

  // Déclenché dès qu'on reçoit un broadcast intent qui réponde aux filtres déclarés dans le Manifest
  @Override
  public void onReceive(Context context, Intent intent) {
    // On vérifie qu'il s'agit du bon intent
    if(intent.getAction().equals("ACTION_COUCOU")) {
      // On récupère le nom de l'utilisateur
      String nom = intent.getExtra(NOM_USER);
      Toast.makeText(context, "Coucou " + nom + " !", Toast.LENGTH_LONG).show();
    }
  }
}
```

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 :

```
import android.app.Activity;
import android.content.IntentFilter;
import android.os.Bundle;

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 on
Pause **/
  @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"/>
```

- 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>`.

Le stockage de données

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.

Ici, on ne parlera pas des bases de données. Mais bientôt, promis.

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 (http://developer.android.com/reference/android/content/SharedPreferences.html)`. 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 (http://developer.android.com/reference/android/content/SharedPreferences.Editor.ht
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 :

```
public final static String FAVORITE_COLOR = "fav color";

…

SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(this);
SharedPreferences.Editor editor = preferences.edit();
editor.putString(FAVORITE_COLOR, "FFABB4");
editor.commit();
```

De manière naturelle, 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` :

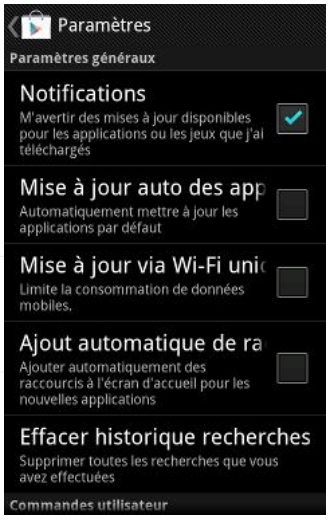
```
// On veut la chaîne de caractères d'identifiant FAVORITE_COLOR
// Si on ne trouve pas cette valeur, on veut rendre "FFFFFF"
String couleur = preferences.getString(FAVORITE_COLOR, "FFFFFF");
```

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 suivante). 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.



L'activité permettant de choisir des paramètres pour le Play Store

Ce type d'activités s'appelle les « `PreferenceActivity (http://developer.android.com/reference/android/preference/PreferenceActivity.html)` ». 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.

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 :

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >
  <PreferenceCategory android:title="Réseau">

  </PreferenceCategory>
  <PreferenceCategory android:title="Luminosité">

  </PreferenceCategory>
  <PreferenceCategory android:title="Couleurs">

  </PreferenceCategory>
</PreferenceScreen>
```

… donne le résultat visible à la figure suivante.

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`.

```
<CheckBoxPreference
  android:key="checkBoxPref"
  android:title="Titre"
  android:summaryOn="Résumé quand sélectionné"
  android:summaryOff="Résumé quand pas sélectionné"
  android:defaultValue="true"/>
```

Ce qui donne la figure suivante.

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 rappellerons 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.

```
<EditTextPreference
  android:key="editTextPref"
  android:dialogTitle="Titre de la boîte"
  android:positiveButtonText="Je valide !"
  android:negativeButtonText="Je valide pas !"
  android:title="Titre"
  android:summary="Résumé"
  android:dependency="checkBoxPref" />
```

Ce qui donne la figure suivante.

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 suivante.

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 ! Encore une fois, 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 :

```
<resources>
  <array name="liste_couleurs_fr">
    <item>Bleu</item>
    <item>Rouge</item>
    <item>Vert</item>
  </array>
  <array name="liste_couleurs">
    <item>blue</item>
    <item>red</item>
    <item>green</item>
  </array>
</resources>
```

Qu'on peut ensuite fournir aux attributs susnommés :

```
<ListPreference
  android:key="listPref"
  android:dialogTitle="Choisissez une couleur"
  android:entries="@array/liste_couleurs_fr"
  android:entryValues="@array/liste_couleurs"
  android:title="Choisir couleur" />
```

Ce qui donne la figure suivante.

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 :

```
<activity
  android:name=".PreferenceActivityExample"
  android:label="@string/title_activity_preference_activity_example" >
</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 :

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  addPreferencesFromResource(R.xml.preference);
}
```