Créez des applications pour Android

Par Frédéric Espiau (DakuTenshi)

www.siteduzero.com

Dernière mise à jour le 10/05/2012

Sommaire

Créez des applications pour Android

Le tutoriel que vous êtes en train de lire est en **bêta-test**. Son auteur souhaite que vous lui fassiez part de vos commentaires pour l'aider à l'améliorer avant sa publication officielle. Notez que le contenu n'a pas été validé par l'équipe éditoriale du Site du Zéro.

Par  Frédéric Espiau (DakuTenshi)

Mise à jour : 10/05/2012
Difficulté : Intermédiaire Durée d'étude : 2 mois

Bonjour à tous et bienvenue dans le monde merveilleux du développement d'applications 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'application de Google ? Hé bien figurez-vous que c'est justement le but de ce cours : vous apprendre à créer des applications pour Android !

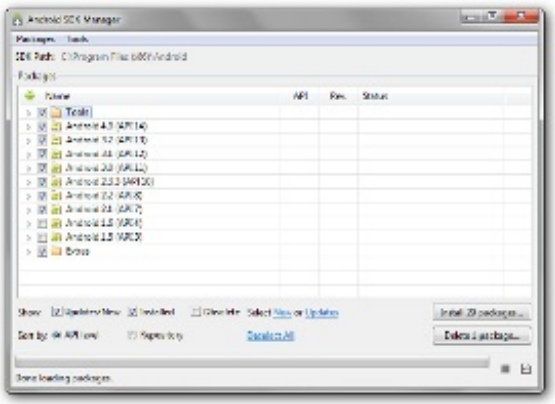
Cependant, pour suivre ce cours, il vous faudra quelques connaissances :

- Les applications Android étant presque essentiellement codées en **Java**, il vous faut connaître ce langage. Heureusement, le Site du Zéro propose un cours et même un livre sur le Java.
- Connaître un minimum de **SQL** pour les requêtes (ça tombe bien, le Site du Zéro propose un cours sur MySQL) ainsi qu'avoir des bases sur **OpenGL**.
- 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 ça, 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 de pouvoir 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 !


android
open handset alliance




Gras
Italique
Souligné
Smileys :
Noir
Bleu
Rouge
Cacher
Edition :
Prévisualisation :

Partie 1 : Les bases indispensables à toutes applications

L'univers Android

Dans ce tout premier chapitre, je vais vous présenter ce que j'appelle l'« univers Android » ! La genèse de ce système 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 des téléphones portables.
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 d'essayer de développer un système d'exploitation mobile plus intelligent, qui ne se contentait pas uniquement de permettre d'envoyer des SMS et transmettre des appels, mais qui devait permettre d'interagir avec la situation de l'utilisateur dans la nature (notamment avec sa position géographique). C'est pourquoi, contrairement à une croyance populaire, on peut affirmer qu'Android n'est pas une réponse de Google à l'iPhone d'Apple puisque l'existence de ce dernier n'a été révélée que 2 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, prévenant ainsi la possibilité de développer aisément une application qui s'adapterait à tous les téléphones, puisque la base était 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 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. Android est le logiciel vedette de cette alliance, mais il ne s'agit pas de leur seule activité.

Il existe à l'heure actuelle plus de 80 membres dans l'OHA.

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, hein ?

La philosophie et les avantages d'Android

Open-source

Le contrat de licence pour Android respecte l'idéologie 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-sources 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 (par contre bon courage pour tout le travail à fournir !). En revanche, pour poster vos applications sur le Play Store, il vous en coûtera la modique somme de 25$. Ces 25$ permettent de publier autant d'applications que vous le souhaitez, à vie !

Facile à développer

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

Une API, ou `Interface de Programmation` en Français, est un ensemble de règles à suivre pour pouvoir dialoguer avec d'autres applications. Dans le cas du 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 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 liée, 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 boutons. 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 version 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 de manière négative.

Bon ça parait 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 que vous devez connaître la programmation en Java pour être en mesure de suivre ce cours ; je ne fais ici que rappeler quelques notions de bases pour vous rafraîchir la mémoire ! 

La notion d'objet

La seule chose qu'un programme sait faire, c'est des calculs ou afficher des images. Pour faire des calculs, on a besoin de **variables**. Ces variables permettent de conserver des informations avec lesquelles on va pouvoir faire des opérations. En Java, il existe deux types de variables. Le premier type s'appelle les **primitives**. Ces primitives permettent de retenir des informations simples telles que des nombres sans virgules (auquel cas la variable est un entier, `int`), des chiffres à virgules (des réels, `float`) ou des booléens (variable qui ne peut être que vraie : `true` ou fausse : `false`, avec `boolean`).

Cette liste n'est bien sûr pas exhaustive !

Le second type, ce sont les variables objet. En effet, à l'opposé des primitives (les variables « simples »), il existe les **objets** : les variables « compliquées ».
En fait une primitive ne peut contenir qu'une information, par exemple la valeur d'un chiffre ; tandis qu'un objet est constitué d'une ou plusieurs autres variables. Ainsi, une variable objet peut elle-même contenir une variable objet ! Un objet peut représenter absolument ce qu'on veut : un concept philosophique, une formule mathématique, une chaise, une voiture... Par définition, je ne peux représenter une voiture dans une variable primitive. Je peux cependant concevoir un objet auquel je donnerais quatre roues, un volant et une vitesse ; ce qui ressemblerait déjà pas mal à une voiture !

Prenons donc un exemple, en créant un objet Voiture. Pour cela, il va falloir déclarer une **classe** voiture, comme ceci :

Code : Java

```
//Dans la classe Voiture
class Voiture {
    //Les attributs
    int nombre_de_roues = 4;
    float vitesse;
}
```

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 :

Code : Java

```
//Je déclare une méthode
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 :

Code : Java

```
//On dit ici que la méthode renvoie un float et qu'elle a besoin
```

```
    d'un float et d'un boolean pour s'exécuter
    float changer_vitesse(float facteur_de_vitesse, boolean
    acceleration)
        //S'il s'agit d'une 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 méthodes, 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. Construire un objet s'appelle l'**instanciation**.
Ainsi, tout à l'heure, en créant l'objet Voiture via l'instruction : `class Voiture {}`, nous avons en réalité **instancié l'objet Voiture**.

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, 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.

Code : Java

```
    //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.

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**.

Code : Java

```
//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 faire 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 pour exécuter votre programme.

Installation et configuration des outils

Avant de pouvoir rentrer 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 **2,11 Go** pour tout installer. Et si vous possédez un terminal sous Android, je vais vous montrer comment le configurer de façon à pouvoir travailler directement avec.

Encore un peu de patience, les choses sérieuses démarrerons 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).

Et comment savoir quelle version de Windows j'utilise ?

C'est simple, si vous utilisez Windows 7 ou Windows Vista, appuyez en même temps 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, utilisez une distribution `Ubuntu` plus récente que la `8.04`. 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.

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

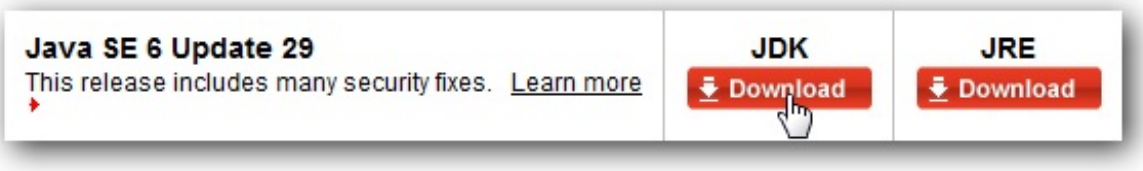
Le Java Development Kit

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

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

- le **JRE** (**J**ava **R**untime **E**nvironment), qui contient la `JVM` (**J**ava **V**irtual **M**achine, rappelez-vous j'ai expliqué le concept de machine virtuelle dans le premier chapitre), les bibliothèques de base du langage ainsi que tous les composants nécessaires au lancement d'applications ou d'applets Java. En gros, c'est l'ensemble d'outils qui vous permettra d'exécuter des applications Java.
- le **JDK** (**J**ava **D**evelopment **K**it), qui contient le JRE (afin d'exécuter les applications Java), mais aussi un ensemble d'outils pour compiler et déboguer votre code ! Vous trouverez un peu plus de détails sur la compilation dans l'annexe sur l'architecture d'Android.

Rendez-vous ici 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`.



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

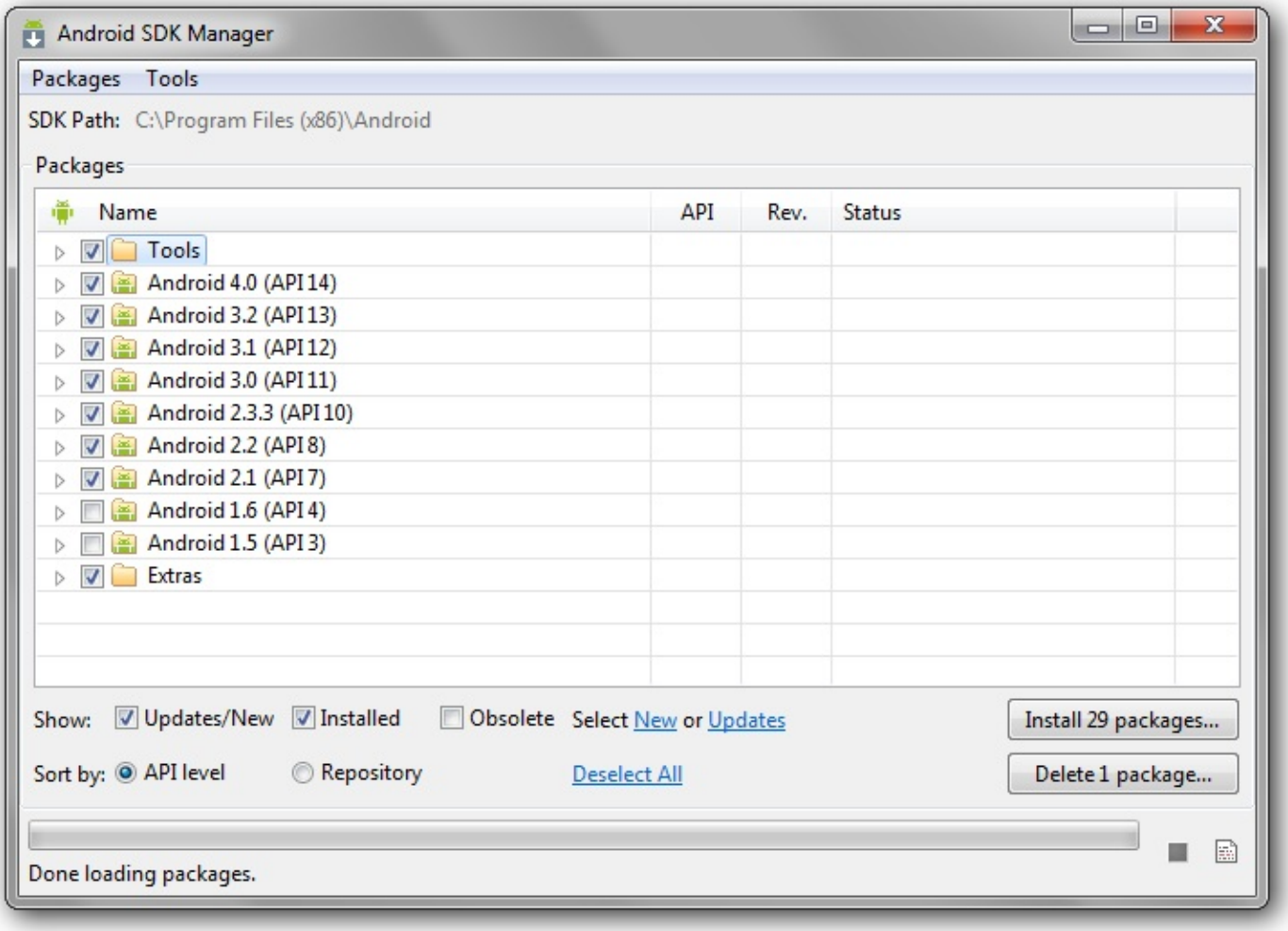
Choisissez ensuite la version adaptée à votre configuration. Une fois le téléchargement terminé, vous pouvez installer le tout là où vous le désirez. Vous aurez besoin de **200 Mo** de libre sur le disque ciblé.

Le SDK d'Android

C'est quoi un `SDK` ?

Un SDK, 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 et sélectionnez la version dont vous avez besoin. Au premier lancement du SDK, un écran de ce type s'affichera :



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 que tu as sélectionnés ?

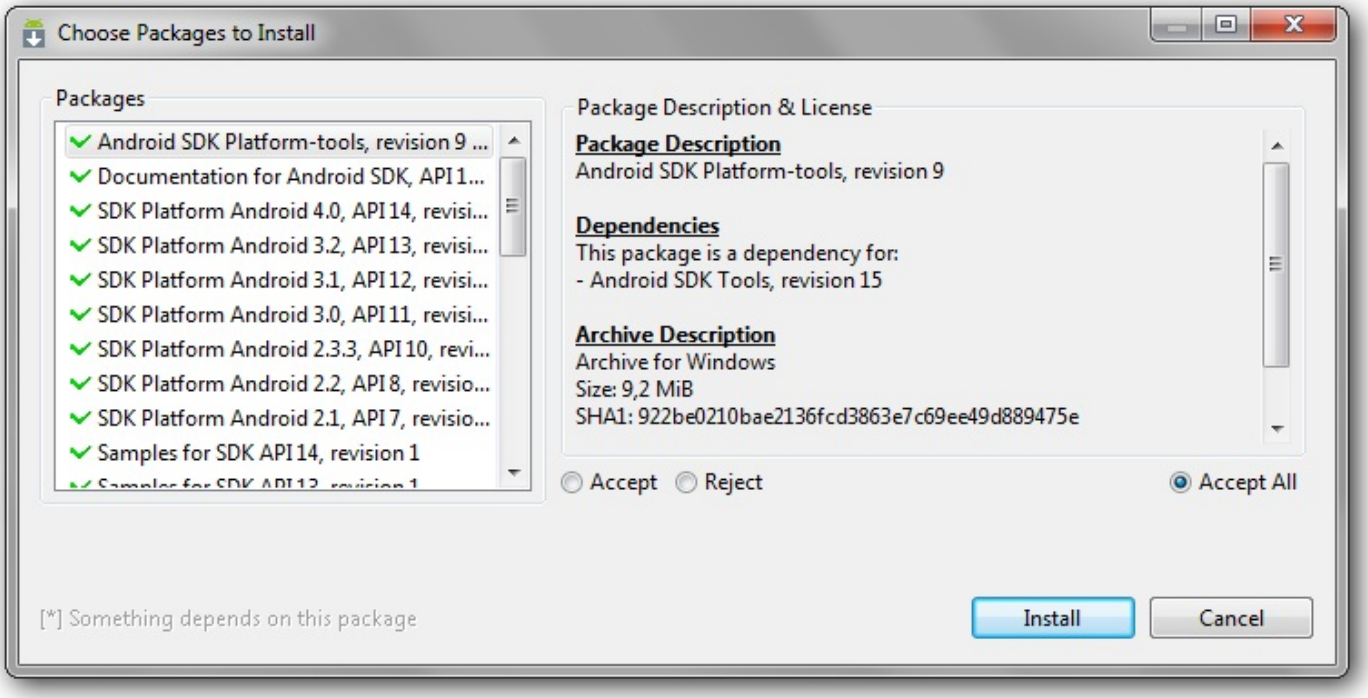
Regardez bien le nom des paquets, vous remarquerez qu'ils suivent tous un même motif. Il est écrit à chaque fois `Android [un nombre] (API [un autre nombre])`. La présence de ces nombres s'explique par le fait qu'il existe plusieurs versions de la plateforme Android qui ont été développées depuis ses 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 au moins un de ces numéros (l'autre sera toujours le même pour une même version), puisqu'une application développée pour une version précise d'Android ne fonctionnera pas pour les version 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 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 1.2% 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 serons 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.



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.

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. Vous pouvez aussi opter pour d'autres IDE compétents tels que IntelliJ IDEA, NetBeans avec une extension ou encore MoSync.

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

Cliquez ici 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 en APK (pour pouvoir publier vos applications).

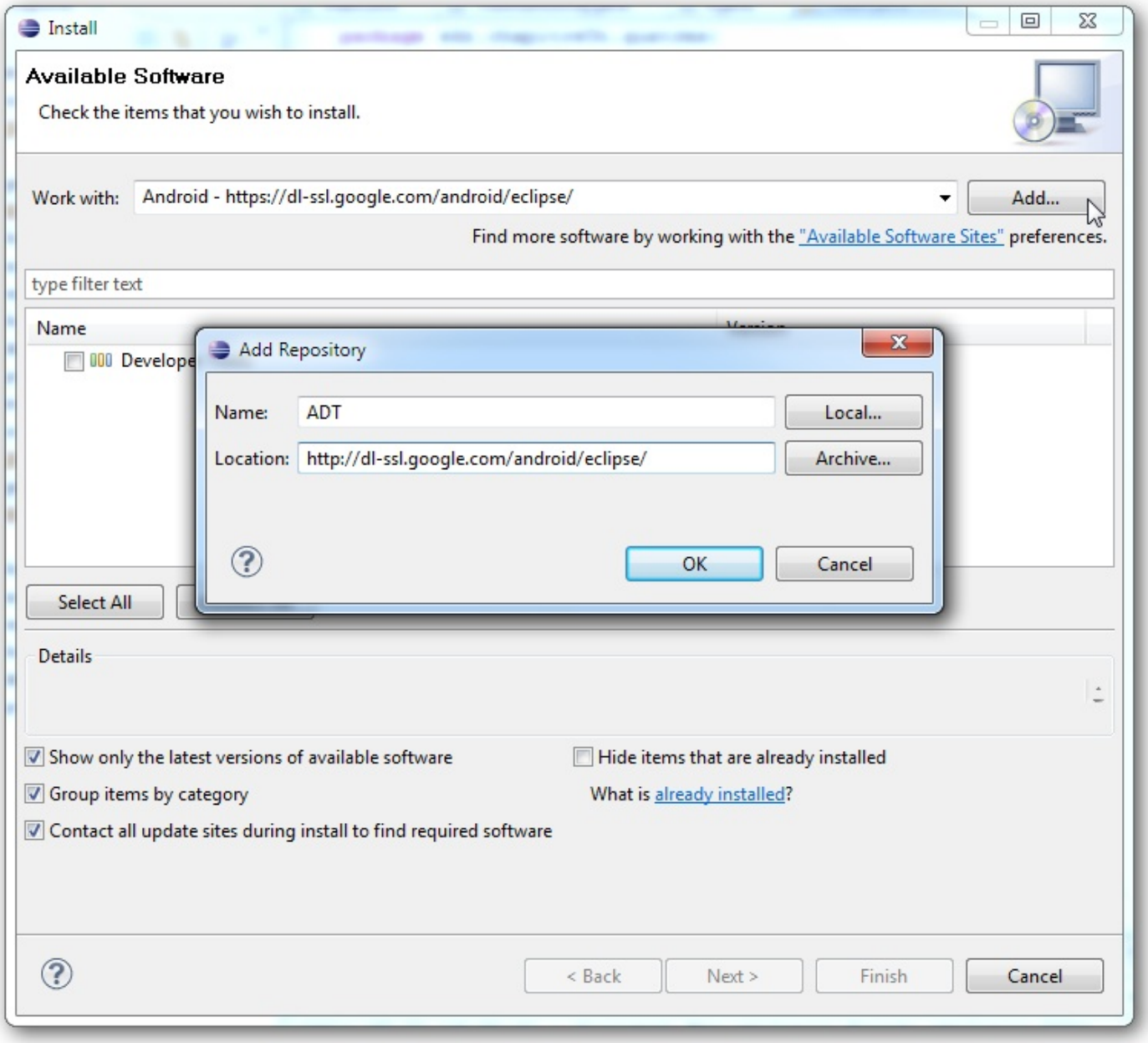
ADT n'est pas le seul add-on qui permette de paramétrer Eclipse pour le développement Android, le MOTODEV Studio For Android est aussi très évolué si vous le désirez.

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, copiez l'adresse ci-dessous :

Citation

https://dl-ssl.google.com/android/eclipse/

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 sur `OK`.

Si cette manipulation ne fonctionne pas, essayez avec l'adresse suivante :

Citation

> http://dl-ssl.google.com/android/eclipse/

(même chose mais sans le « s » à **http**)

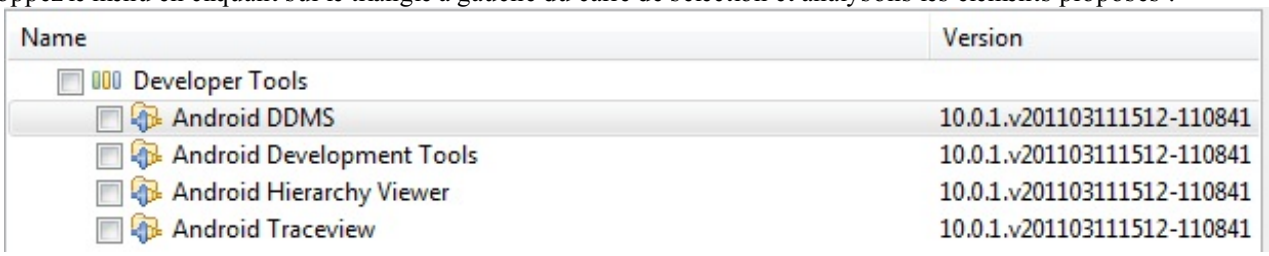
Si vous rencontrez toujours une erreur, alors il va falloir télécharger l'ADT manuellement. Cliquez sur ce lien puis cliquez sur le lien qui se trouve dans le tableau afin de télécharger une archive qui contient l'ADT :

Name	Package	Size	MD5 Checksum
ADT 16.0.1	ADT-16.0.1.zip	7000078 bytes	03a2a23650ddac128c8b9e8aaf0aa433

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 :

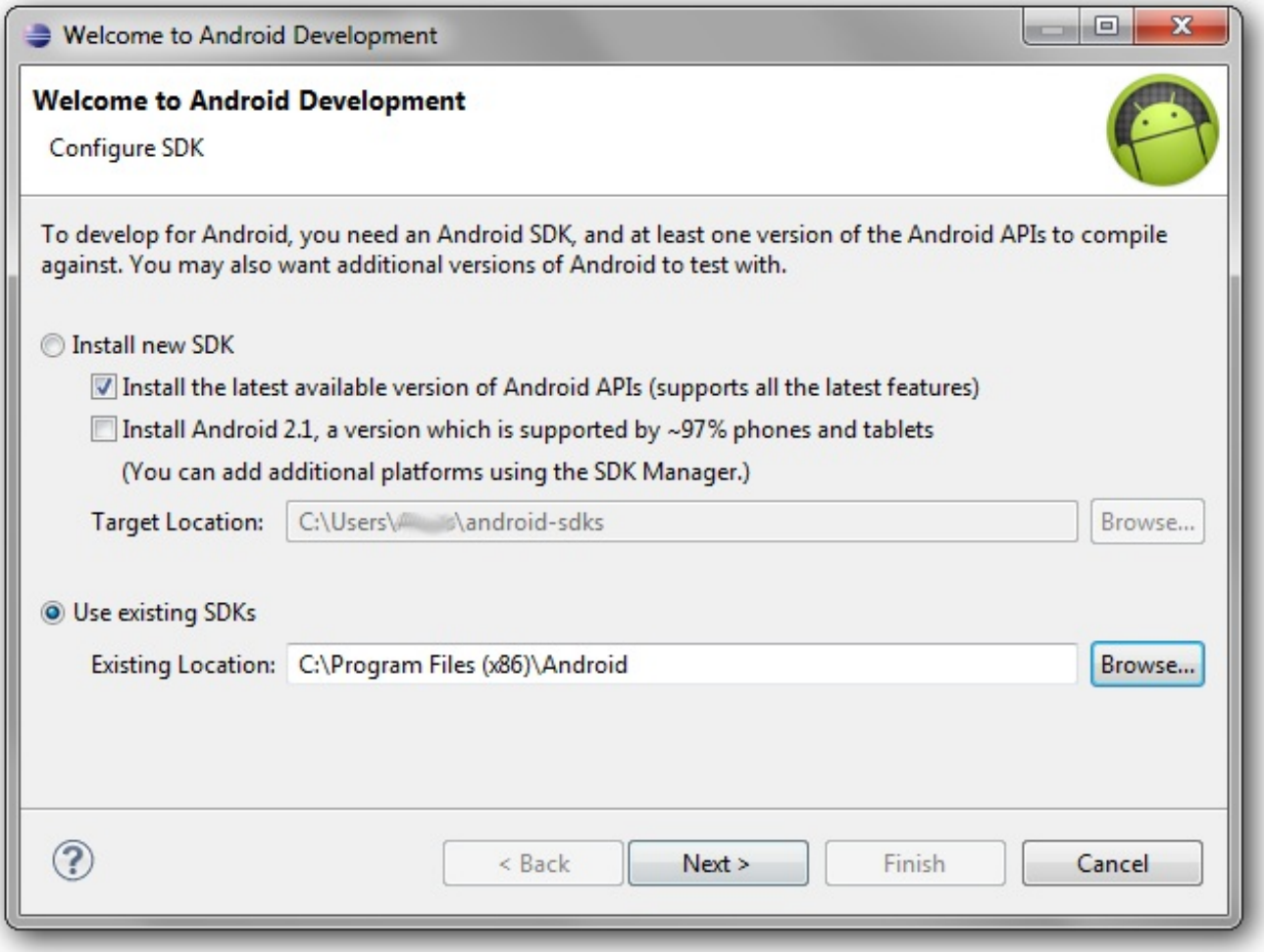


- `Android DDMS` est l'*Android Dalvik Debug Monitor Server*, il permet d'exécuter quelques fonctions pour vous aider à déboguer votre application (simuler un appel ou une position géographique par exemple) et d'avoir accès à d'autres informations utiles.
- L'ADT.
- `Android Hierarchy Viewer` qui permet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

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. Faites donc en cliquant sur `Restart Now`. Au démarrage, Eclipse vous demandera d'indiquer où se situe le SDK :



Sélectionnez `Use existing SDKs` puisqu'on a déjà téléchargé un SDK, puis cliquez sur `Browse...` pour sélectionner l'emplacement du SDK.

C'est fait, Eclipse sait désormais où trouver le SDK. On n'est pas encore tout à fait prêts, il nous reste 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 qu'il en simule le comportement. C'est la raison pour laquelle vous n'avez pas besoin d'un périphérique sous Android pour tester votre application !

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.

Vous voyez ce couple d'icônes ?

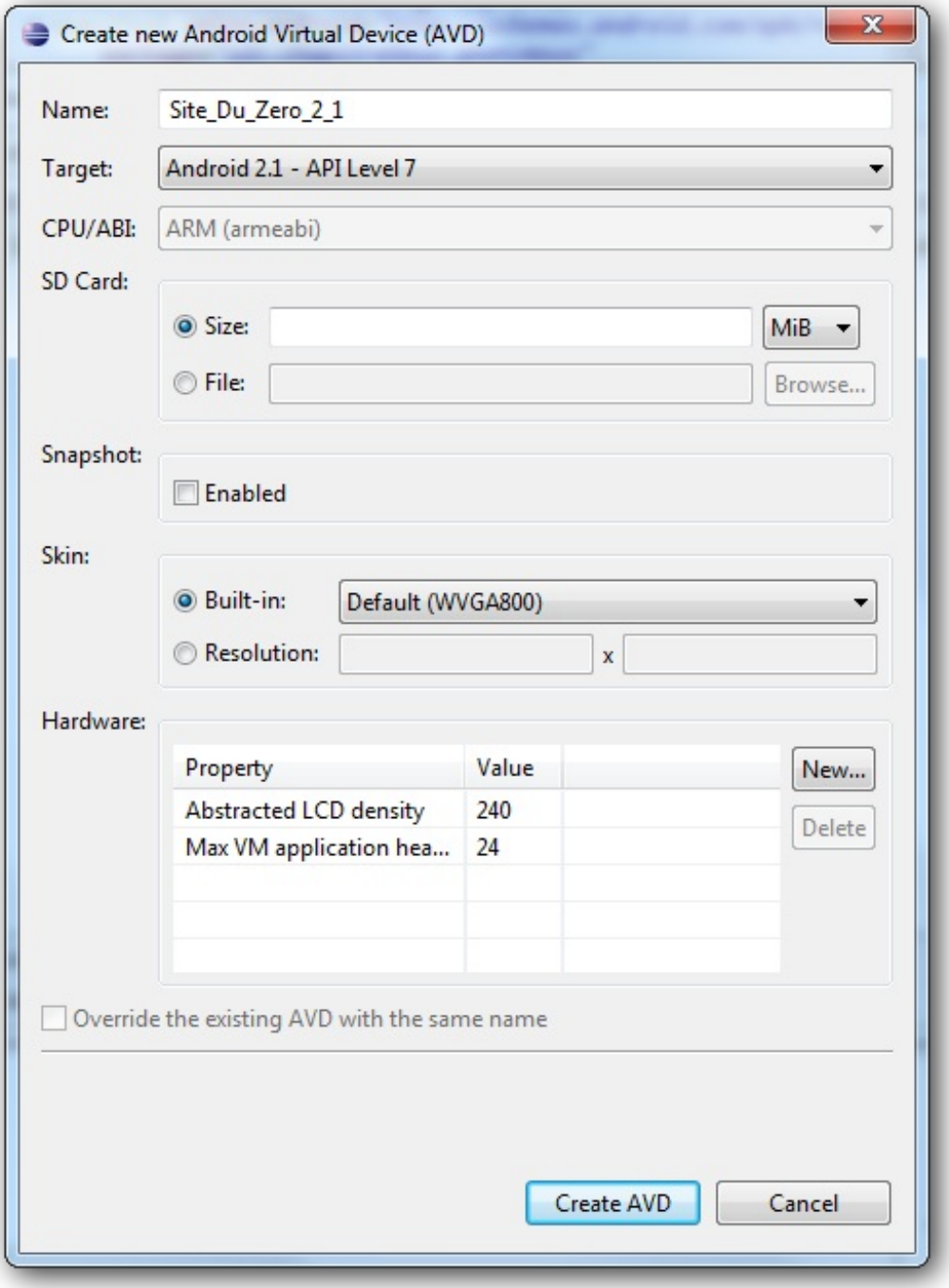
Celle de gauche permet d'ouvrir les outils du SDK et celle de droite permet d'ouvrir l'interface de gestion d'AVD. Cliquez dessus puis sur `New...` pour ajouter un nouvel AVD.

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, mais il se peut aussi qu'Eclipse pipote un peu, auquel cas ré-appuyez sur le bouton jusqu'à ce qu'il abdique. 

Une fenêtre s'ouvre, 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 accents et les espaces ne sont pas autorisés.
Dans `Target`, choisissez `Android 2.1 - API Level 7` puisque j'ai décidé que nous ferons notre première application sans le Google API et sans les fonctionnalités qu'apportent les versions suivantes d'Android.

Laissez les autres options à leur valeur par défaut, nous y reviendrons plus tard quand nous confectionnerons d'autres AVD. Cliquez enfin sur `Create AVD` et vous aurez une machine prête à l'emploi !



Si vous utilisez Windows et que votre nom de session possède un caractère spécial - dont les accents, alors Eclipse vous enverra paître en déclarant qu'il ne trouve pas le fichier de configuration de l'AVD. Je pense à quelqu'un dont la session s'appellerait "Jérémie" par exemple. Heureusement, il existe une solution à ce problème.

Si vous utilisez Windows 7 ou Windows Vista, appuyez en même temps 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 `Entrée`. 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 présent 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`.

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 (avec un vrai éditeur de texte, c'est-à-dire pas le bloc-note de Windows. Essayez plutôt Notepad++).
Il devrait ressembler à :

Code : Ini

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

S'il n'y as 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 :

Code : Ini

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

en

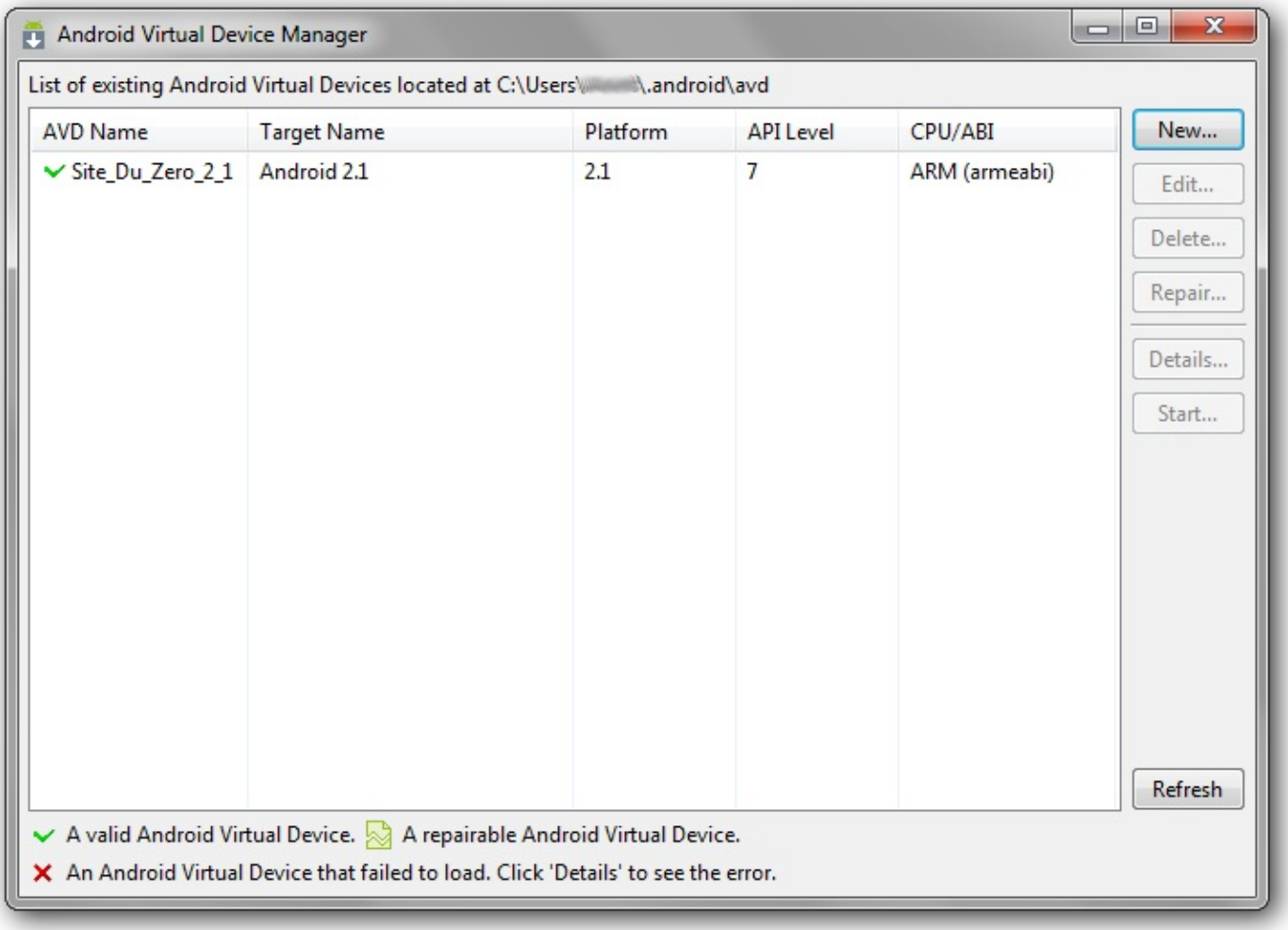
Code : Ini

```
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.

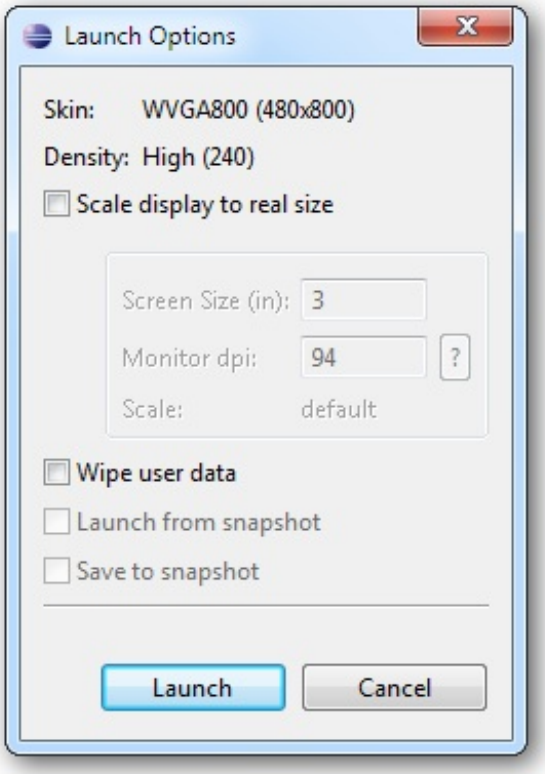
Si vous êtes sortis du gestionnaire Android, retournez-y en cliquant sur l'icône `Bugdroid`, comme nous l'avons fait auparavant. Vous aurez quelque chose de plus ou moins similaire à ça :



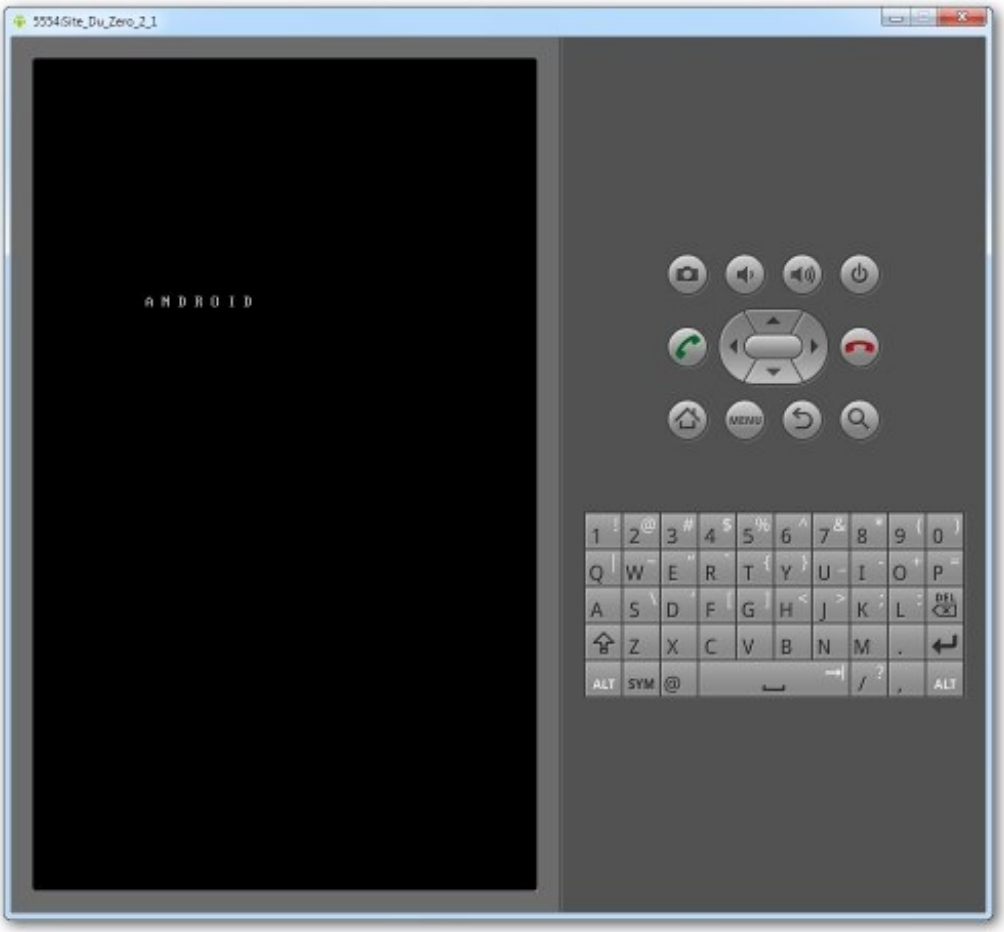
Vous y voyez l'AVD que nous venons tout juste de créer. Cliquez dessus pour déverrouiller le menu de droite. Comme je n'ai pas l'intention de vraiment détailler ces options moi-même, je vais rapidement vous expliquer à quoi elles correspondent pour que vous sachiez les utiliser en cas de besoin. Les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

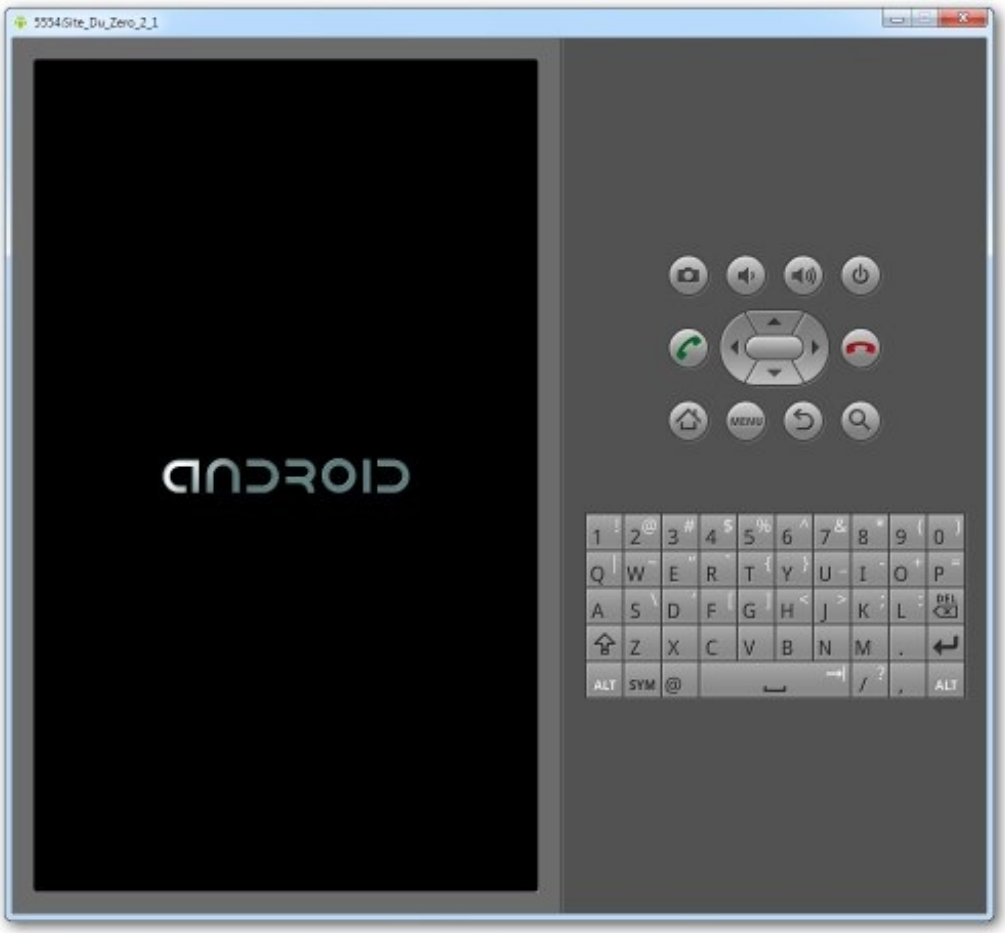
Cliquons donc sur le bouton `Start...` et une nouvelle fenêtre se lance, qui devrait ressembler peu ou prou à ceci :



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 :



puis :



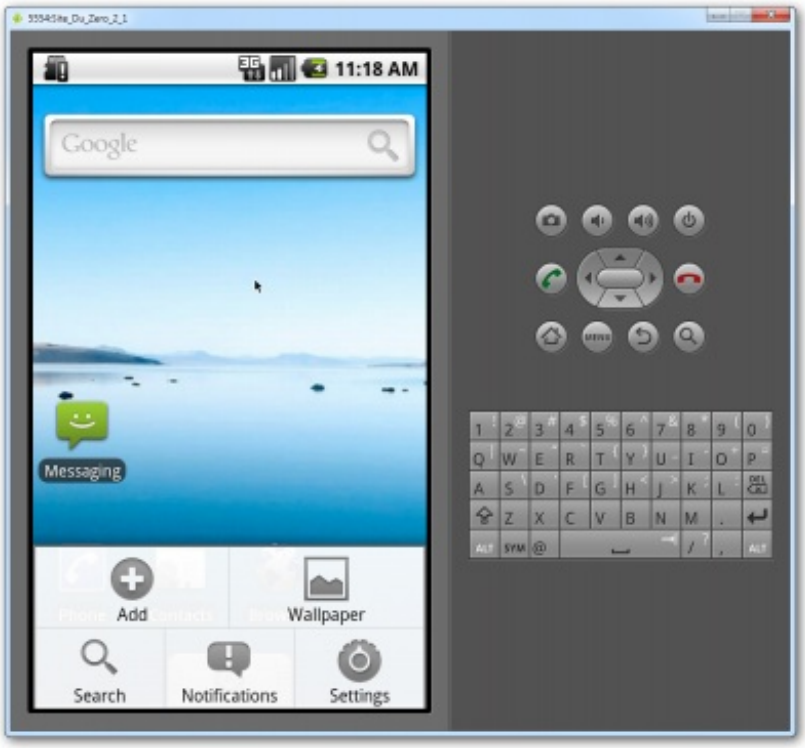
Enfin, votre terminal se lancera. Voici la liste des boutons qui se trouvent dans le menu à droite et ce à quoi ils servent :

Icône	Fonction
	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, et moi et l'anglais ça fait... ben zéro. 

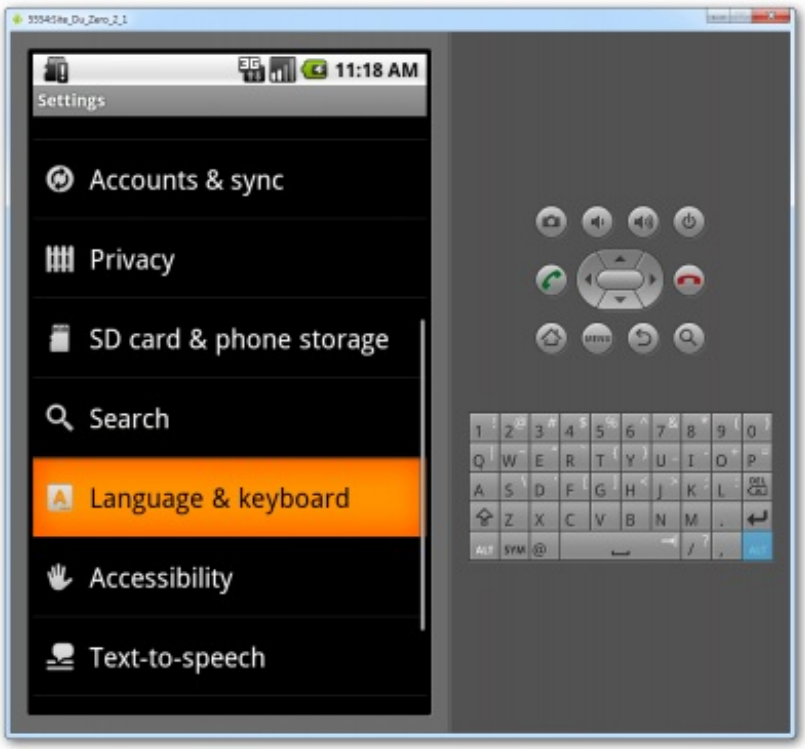
Je ne vais pas vous faire la morale, mais la maîtrise de l'anglais devient vite indispensable dans le monde de l'informatique... ! Ensuite, les machines que vous achetez dans le commerce sont déjà configurées pour le pays dans lequel vous les avez acquises, et comme ce n'est pas une machine réelle ici, alors 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.



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` ; 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).



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-même. 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. Le but est bien entendu de vous permettre de découvrir le système par vous-même et de vérifier que vous avez bien compris comment vous déplacer dans l'environnement Android.

Secret (cliquez pour afficher)

> Cliquez d'abord sur `Date & heure`, dé-sélectionnez `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 suit :

Vous l'aurez remarqué par vous-même, 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  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 Windows

Tout d'abord, vous devez télécharger les drivers adaptés à votre terminal. Je peux vous donner la démarche 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 l'emplacement suivant :

Citation

> \android-sdk\extras\google\usb_driver

Pour les terminaux HTC, les drivers sont fournis dans le logiciel **HTC Sync**. Vous pouvez le télécharger ici.
Pour les autres marques, vous trouverez l'emplacement des pilotes à télécharger dans le tableau qui se trouve sur cette page.

Pour les utilisateurs Mac

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

Pour les utilisateurs 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. Je vais d'abord la décrypter avec vous :
« 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 dans ce tableau. Par exemple pour mon HTC Desire, j'indique la ligne suivante `SUBSYSTEM=="usb", ATTR{idVendor}=="0bb4", MODE="0666", GROUP="plugdev"`, ce qui donne que je tape dans la console
```
echo "SUBSYSTEM==\"usb\", ATTR{idVendor}==\"0bb4\", MODE=\"0666\", GROUP=\"plugdev\"" >> /etc/udev/rules.d/51-android.rules
```

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

Et après ?

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

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 ?

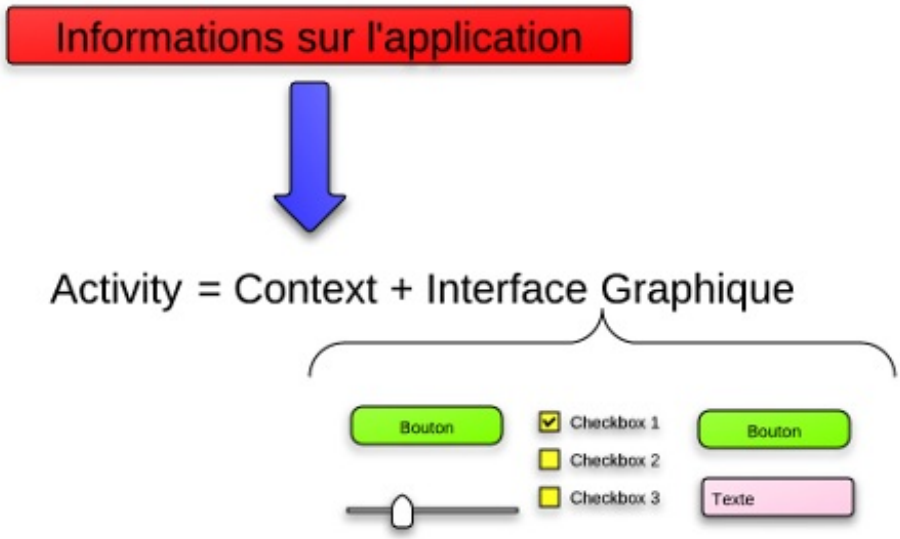
Si vous observez un peu l'architecture de la majorité des applications Android, vous remarquerez une construction toujours à peu près similaire. Prenons par exemple l'application du Play Store. Vous avez plusieurs fenêtres à l'intérieur même de cette application : si vous effectuez une recherche, une 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 remplie tout l'écran, alors votre application ne peut en afficher qu'une à la fois.

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 prodiguent des informations. Pour rentrer dans les détails, 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 `contexte`. Le contexte constitue un lien avec le système Android ainsi que les autres activités de l'application.



Une activité est constituée du contexte de l'application et d'une seul et unique interface graphique

Comme il est plus aisé de comprendre à l'aide d'exemples, imaginez que vous naviguez 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.

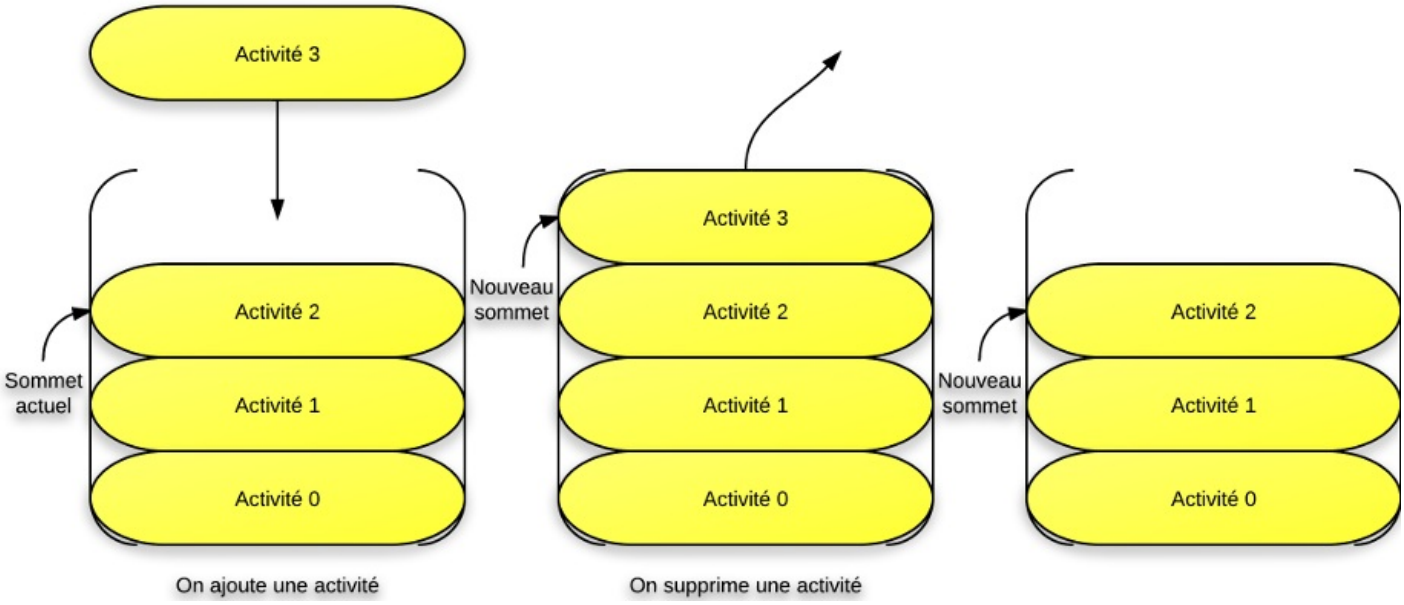
États d'une activité

Comme je vous l'ai déjà dit, si un utilisateur reçoit un appel il devient plus important qu'il puisse y répondre que d'écouter la chanson que votre application diffuse. Pour pouvoir toujours répondre à ce besoin, les développeurs d'Android ont pris deux décisions :

- À tout moment votre application peut laisser place à d'autres priorités. Si votre application utilise trop de ressources système, alors elle empêchera le système de fonctionner correctement et Android pourra décider de l'arrêter sans prévenir.
- 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.

En fait, quand une application se lance, elle se met tout en haut de ce qu'on appelle la pile d'activité.

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 et l'objet en seconde place deviendra le nouveau sommet :

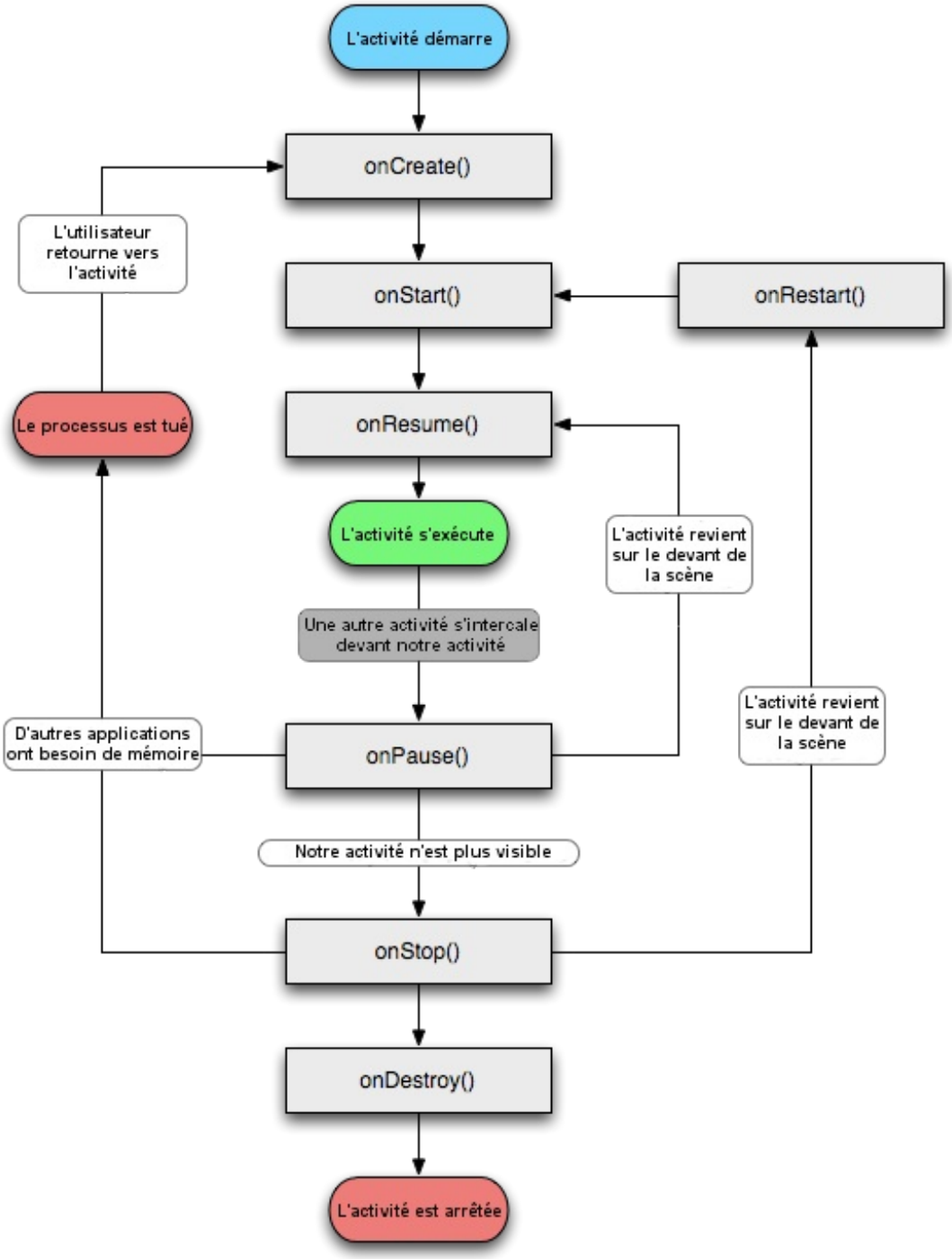


L'activité que voit l'utilisateur est celle qui se trouve au-dessus de la pile. Ainsi, lorsqu'un appel arrive, il se place au sommet de la pile et c'est lui qui s'affiche à la place de votre application, qui n'est plus qu'à la seconde place. Votre activité ne reviendra qu'à partir du moment où toutes les activités qui se trouvent au-dessus d'elle seront arrêtées et sorties de la pile. Une activité peut se trouver dans 3 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, 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.

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 - Schéma traduit à partir d'un contenu fourni par Google

Les activités héritent de la classe `Activity`. Or, la classe `Activity` hérite de l'interface `Context` dont le but est de représenter tout ce qui « peut être 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 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 désagréments, 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`

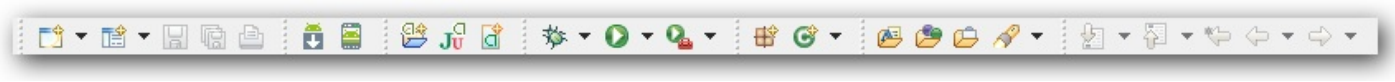
pour la classe `X` qui se trouve dans le package `Z`. Dans le cas un peu farfelu du code source d'un navigateur internet, on pourrait trouver les packages *Web.Affichage.Image*, *Web.Affichage.Video* et *Web.Telechargement*.

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 avec Swing, il est possible de mettre en page les vues à l'aide de conteneurs.

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

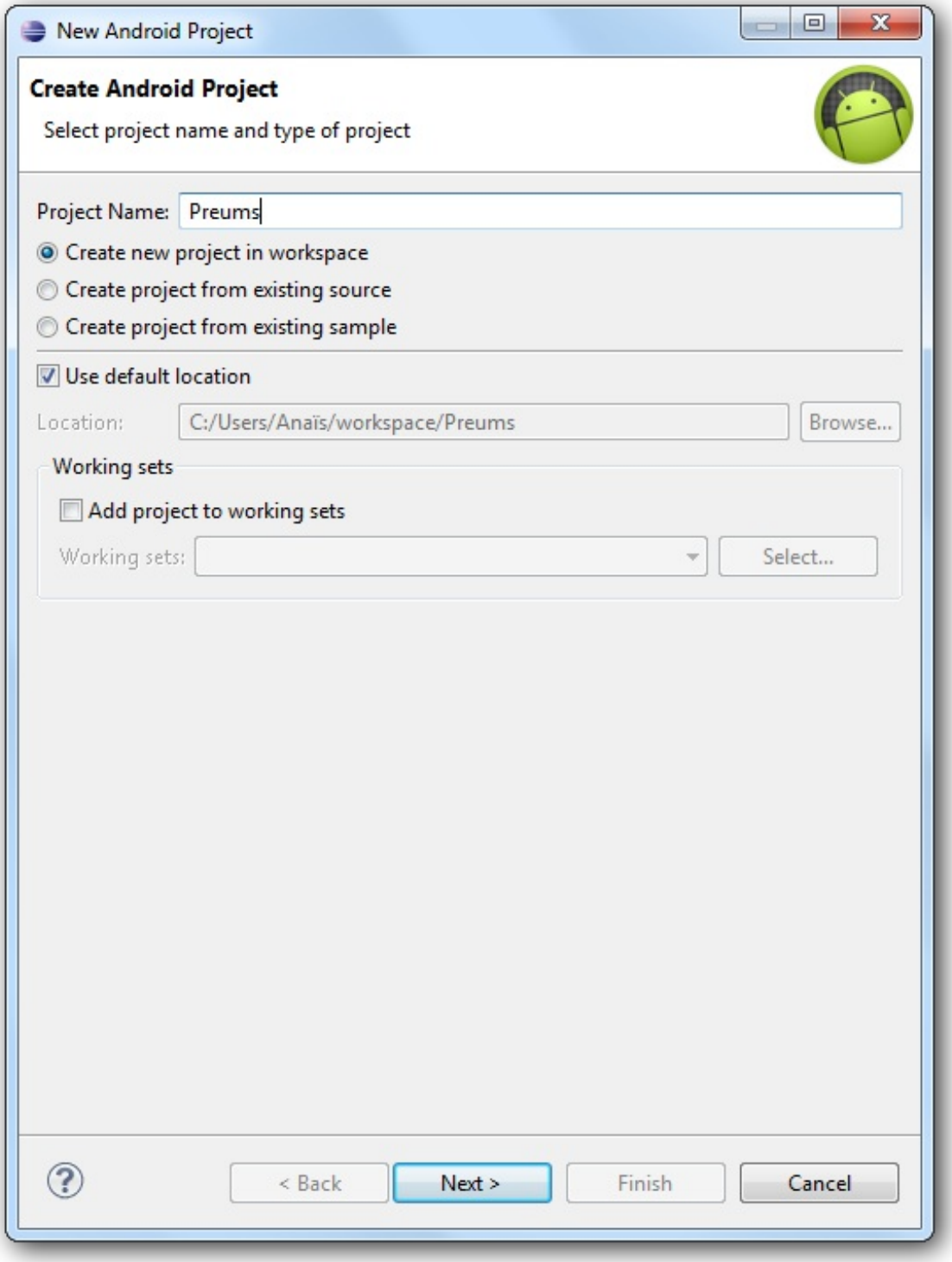
Création d'un projet

Une fois Eclipse démarré, localisez l'emplacement de sa barre d'outils.



Pour lancer la création d'un projet à l'aide de l'assistant de création, cliquez sur le bouton de gauche de la section consacrée à Android :

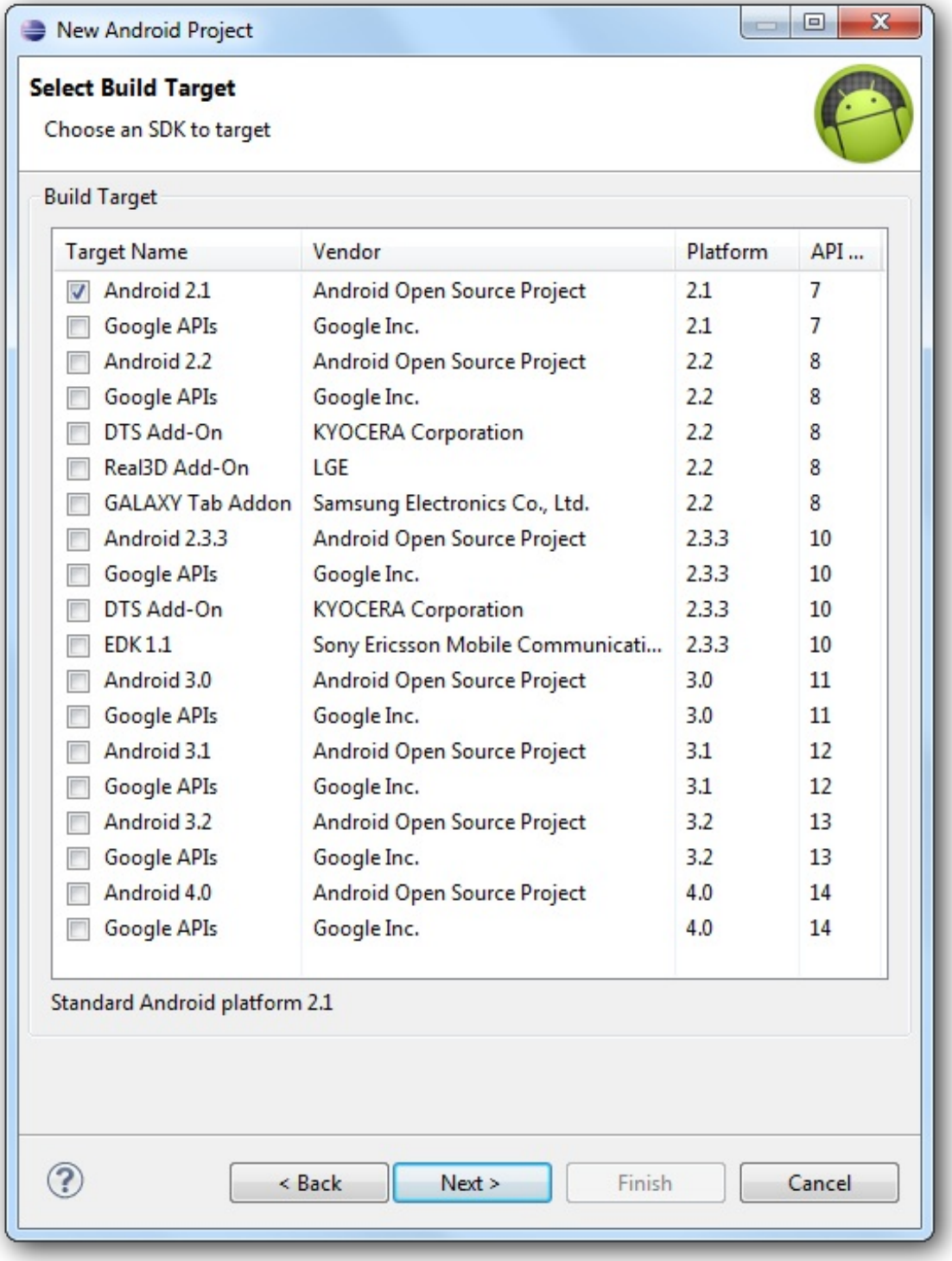
Une fenêtre s'ouvre ; voyons ensemble ce qu'elle contient :



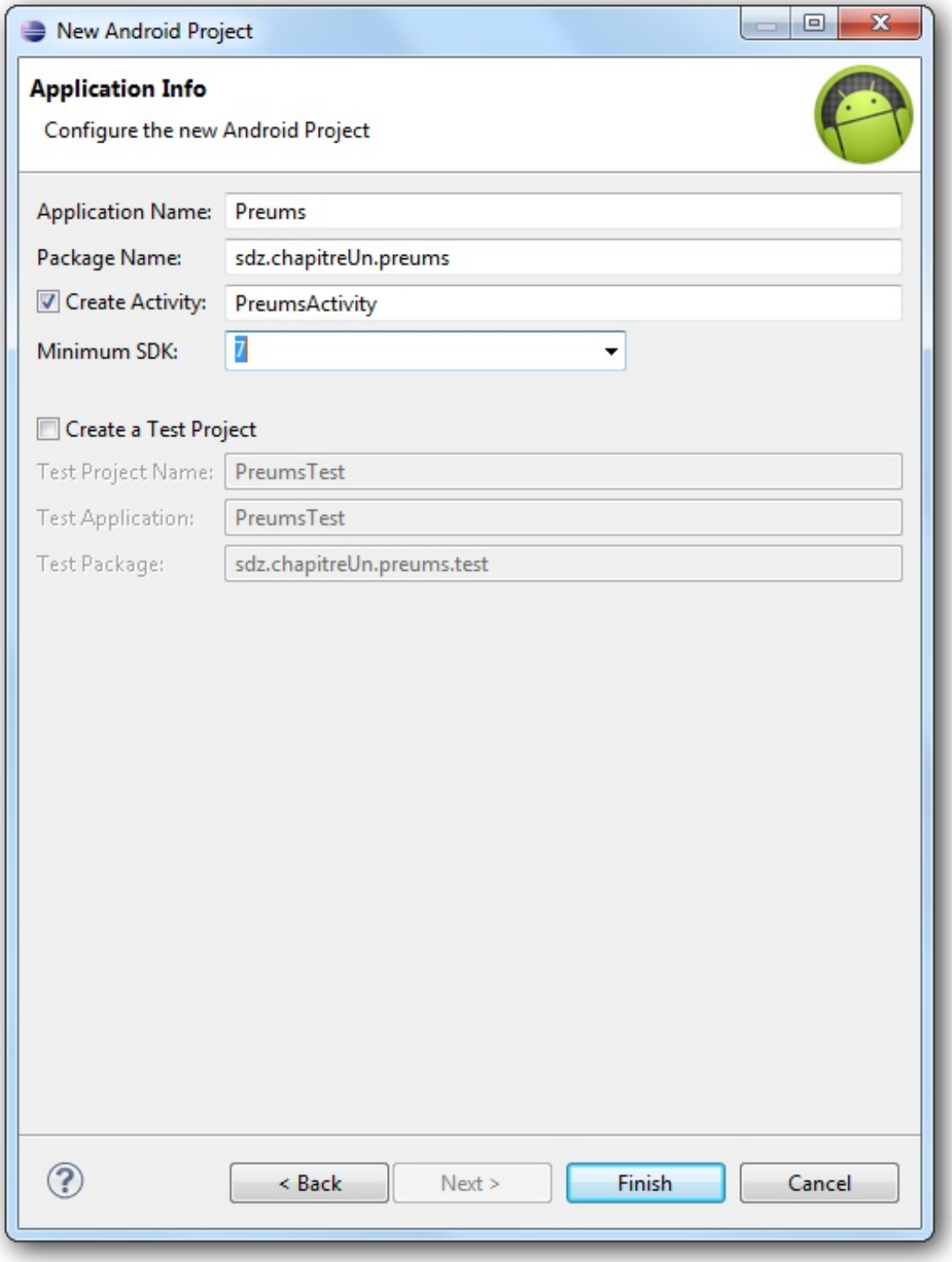
Tous ces champs nous permettent de définir certaines caractéristiques de notre projet :

- **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 reconnaitra Android, peut très bien n'avoir aucune similitude avec ce que vous mettrez dans ce champ. On trouve ensuite trois options. La première vous permet de créer un projet à partir de zéro ; la seconde d'utiliser des fichiers source qui existent déjà (par exemple si une personne a codé une application et qu'elle vous donne les sources, vous pourrez les importer dans un nouveau projet avec cette option) et la troisième et dernière option créera un projet à partir des échantillons de code que vous pouvez télécharger avec le SDK. Utile pour les disséquer et en étudier le comportement ! Comme nous allons créer un tout nouveau projet, il faut sélectionner la première option.
- Vous trouverez ensuite une case cochée par défaut. Si vous la laissez ainsi, le projet sera créé à l'endroit que l'on a défini comme `Workspace` lors du premier lancement d'Eclipse - d'ailleurs cet emplacement est rappelé juste en-dessous. Si vous décochez cette case, les fichiers seront créés à un endroit que vous devrez spécifier.
- Vous pouvez oublier la troisième option, elle ne vous est utile que sur les gros projets, et encore pas dans tous les cas.

Cliquez sur `Next` pour valider ces informations et passer à l'écran suivant :



Cet écran permet simplement de choisir la version d'Android pour laquelle sera compilée votre application. J'avais décrété que je voulais que mon application utilise l'API 7 sans les `Google API`, c'est pourquoi j'ai choisi la ligne `Android 2.1`. Cliquez sur `Next` pour valider ces informations et passer à l'écran suivant :



Tout d'abord, vous pouvez choisir le nom de votre application avec `Application name`. Cette fois-ci, il s'agit du nom qui apparaîtra sur l'appareil et... dans le Play Store pour vos futures applications ! Choisissez donc un nom qui semble correct, original et judicieux. J'ai choisi Preums. Oui bon, ce n'est qu'un exemple.

Il faudra ensuite choisir dans quel package ira notre application. J'ai choisi de rassembler tous mes codes relatifs au Site du Zéro dans le package `sdz` et tout ce qui est relatif au chapitre un dans `chapitreUn`. Ce package doit être unique et ne pas déjà exister sur le Play Store si vous souhaitez y mettre votre application.

Le prochain encart intitulé `Create Activity` permet de choisir si vous voulez créer automatiquement une activité, ce qui est pratique. L'activité peut s'appeler un peu comme vous le désirez (donc dans notre exemple, rien ne nous oblige à mettre `Preums` dans le nom, ni de le terminer par `Activity` comme moi je l'ai fait).

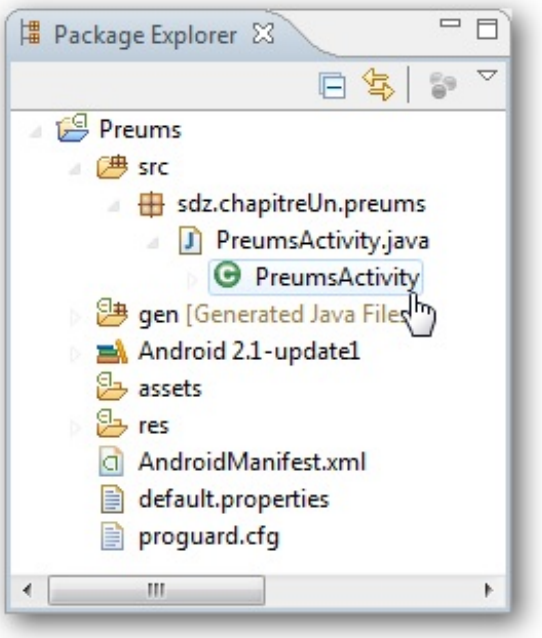
Dans `Min SDK Version` on choisit quelle est la version minimum du SDK qui est acceptée. Si l'utilisateur possède une

version d'Android en-dessous de cette indication, il ne pourra pas la télécharger depuis le Play Store, ni la lancer.

Pour finaliser la création, cliquez sur `Finish`.

Un non-Hello world!

Vous trouverez les fichiers créés dans le package `Explorer` :



On y trouve notre premier grand répertoire `src/`, celui qui contiendra tous les fichiers sources `.java`. Ouvrez le fichier `PreumsActivity.java`. Les autres fichiers ne nous intéressent pas pour l'instant. Vous devriez avoir un contenu similaire à celui-ci :

Code : Java

```
package sdz.chapitreUn.preums;

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

public class PreumsActivity extends Activity {

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

Ah ! On reconnait des 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.

Code : Java

```
package sdz.chapitreUn.preums;
```

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

Code : Java

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

On importe deux classes qui se trouvent dans deux packages différents : les classes `Activity` et `Bundle` afin de pouvoir les utiliser dans notre code.

Code : Java

```
public class PreumsActivity extends Activity {
    //...
}
```

Comme indiqué dans la section précédente, une activité dérive de la classe `Activity`.

Code : Java

```
@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`.

Cette méthode est la première qui est lancée au démarrage d'une application, mais elle est aussi appelée après qu'une application se soit faite tuer par le système en manque de mémoire ! C'est à cela que sert l'attribut 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 ait perdu le focus et redémarré, alors il pourra contenir un état sauvegardé de l'application que vous aurez pris soin de constituer. Par exemple, l'utilisateur sera content si la chanson qu'il écoutait reprenait exactement à l'endroit où elle s'était arrêtée avant d'être sauvagement interrompue par un appel.

Dans cette méthode, vous devez définir ce qui doit être recréé à chaque fois que l'application est lancée après que l'utilisateur en soit sorti (volontairement ou non), donc l'interface graphique et toutes les variables à initialiser par exemple.

Code : Java

```
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 `PreumsActivity`. Il gère bien entendu le cas où le `Bundle` est **`null`**.

Cette instruction est obligatoire.

En revanche l'instruction suivante :

Code : Java

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

sera expliquée dans le prochain chapitre. Pour l'instant, remplacez toute la classe par celle-ci :

Code : Java

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

public class PreumsActivity extends Activity {
    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);
    }
}
```

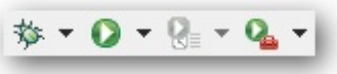
Nous avons ajouté un attribut de classe que j'ai appelé `coucou`. Cet attribut est de type `TextView`, j'imagine que le nom est déjà assez explicite. Il s'agit d'une vue (`View`)… qui représente un texte (`Text`). La méthode `void setContentView(View vue)` permet de faire en sorte que la seule chose qu'affichera notre interface graphique soit la vue passée en paramètre.

Lancement de l'application

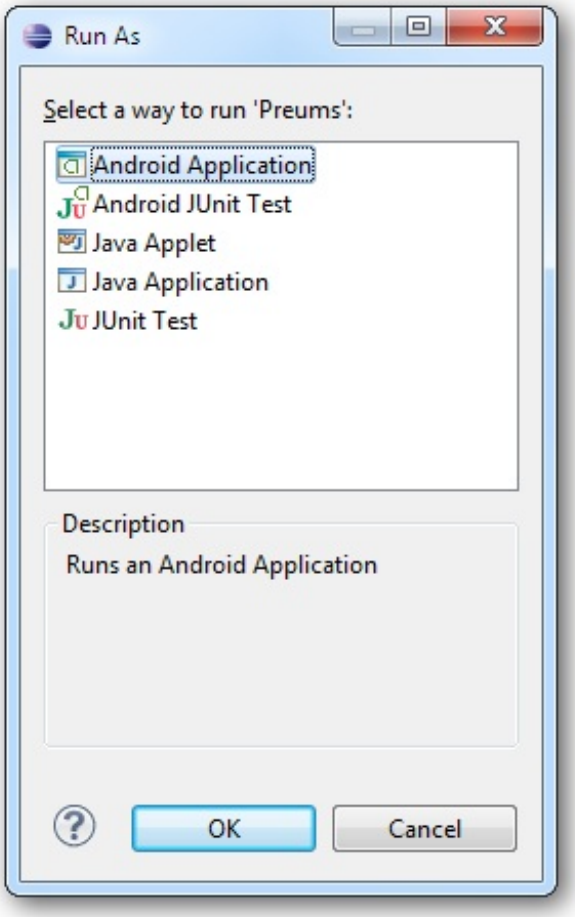
Souvenez-vous, je vous ai dit précédemment qu'il était préférable de ne pas fermer l'AVD, celui-ci étant long à se lancer. Si vous l'avez fermé ce n'est pas grave, il s'ouvrira tout seul. Mais retenez bien par la suite de ne pas le fermer.

Pour lancer notre application, regardez la barre d'outils d'Eclipse :

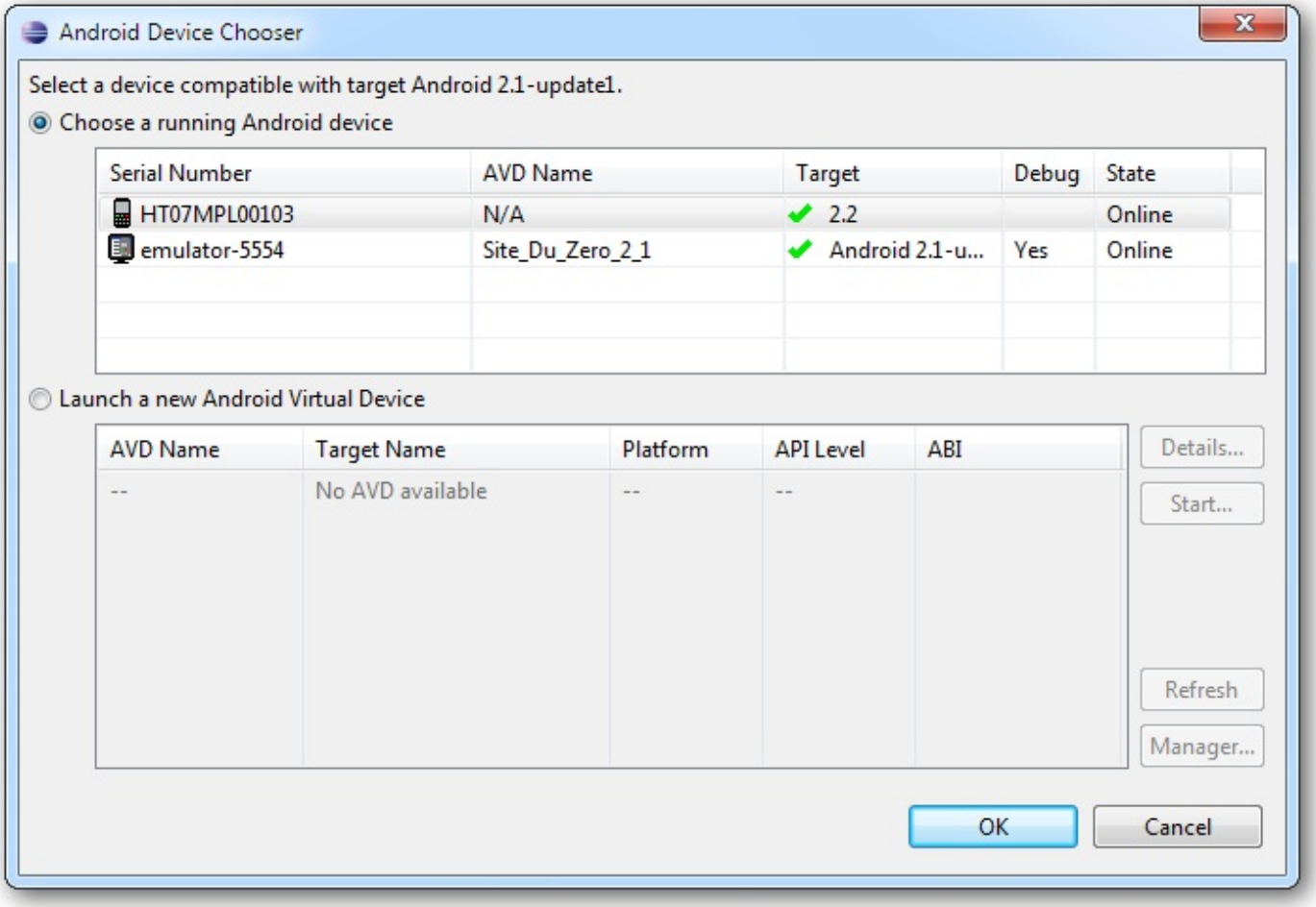
Et cherchez cet encart-là :



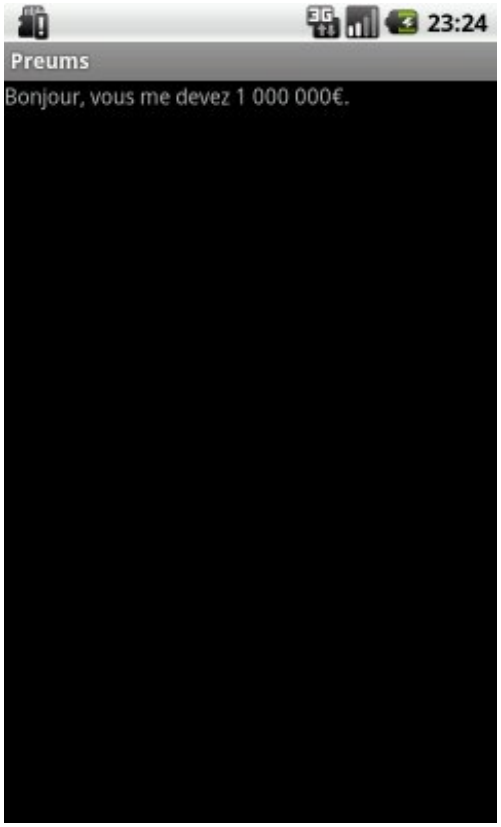
Il vous suffit de cliquer sur le deuxième bouton (celui qui ressemble au symbole « play »). Une fenêtre s'ouvre pour vous demander comment exécuter l'application. Sélectionnez `Android Application` :



Si vous avez plusieurs terminaux, l'écran suivant s'affichera (sauf si vous n'avez pas de terminal de connecté) :



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. Génial ! L'utilisateur (naïf) vous doit 1 000 000 € !



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 qui influenceront ce qui s'affichera à l'écran.

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.

Le format XML

Un des moyens d'adapter nos applications à tous les terminaux est d'utiliser les **ressources**, c'est-à-dire des objets qui seront déclarés dans le format XML et qui nous permettront d'anticiper les besoins de nos utilisateurs en fonction de leur configuration matérielle.

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, des objets de différentes natures qui seront définis dans la langage de balisage XML, c'est pourquoi un point sur le XML s'impose.

Si vous maîtrisez déjà ce concept, vous pouvez passer directement à la suite.

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 connait 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) et effectuera des calculs et des affichages en fonction des informations fournies.

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.

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.

Code : 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 :

Code : 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>`.

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 « bibliothèque », deux nœuds appelés « livre », etc.

Le format XML n'a pas de sens en lui-même. Dans notre exemple, notre nœud s'appelle « bibliothèque », on en déduit, nous humains, 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 `bibliothèque`, 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*. 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 ».

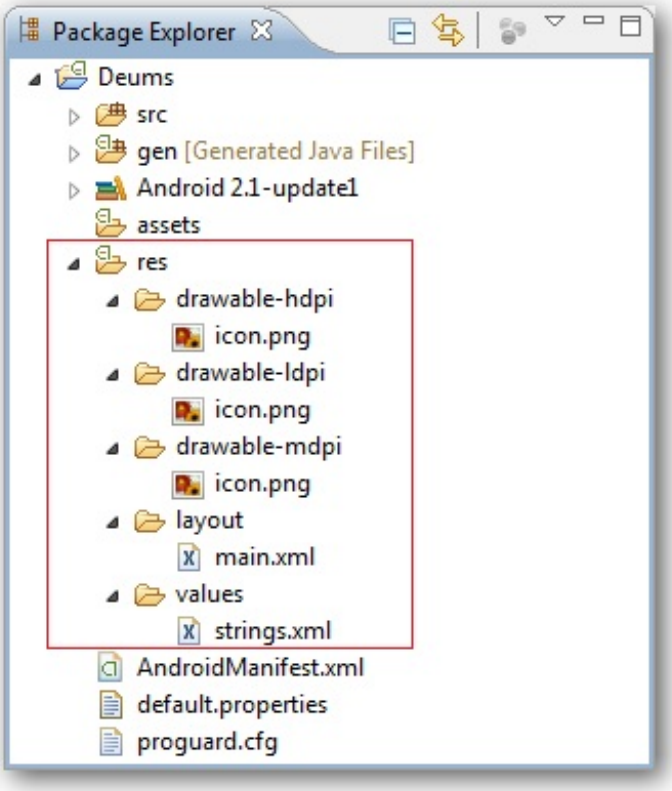
Les nœuds peuvent avoir des *attributs* pour indiquer des informations. Dans notre exemple, le nœud `prix` a l'attribut `devise` afin de préciser en quelle devise est exprimé ce prix : `<prix devise="euro">9.40</prix>` pour « La Horde Du Contrevent », qui vaut donc 9€40. Vous remarquerez que pour « A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

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.



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). On y trouve aussi des fichiers XML dont le contenu décrit des formes ou des dessins.	Oui
Mise en page (`res/layout`)	Les fichiers XML qui représentent la disposition que doivent adopter les vues (on abordera cet aspect, qui est très vaste, dans le prochain chapitre).	Exclusivement
Menu (`res/menu`)	Les fichiers XML pour pouvoir constituer des menus.	Exclusivement
Donnée brute (`res/raw`)	Données diverses au format brut. Ces données n'ont pas de méthodes spécifiques dans Android pour les traiter. On peut imaginer y mettre de la musique ou des fichiers HTML par exemple.	**Le moins possible**
Donnée (`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 chaines 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 ressources sont tout le temps des fichiers XML.
- « Oui » si les fichiers peuvent être d'un autre type qu'XML, en fonction de ce qu'on veut faire.
- « Le moins possible » si les fichiers doivent de préférence ne pas être de type XML. Un fichier XML se trouvera dans ce répertoire uniquement s'il n'a 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 en fait tout à fait normal.

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, mais il est possible de définir d'autres emplacements pour préciser le matériel de destination afin de se conformer au matériel de l'utilisateur. Il y a une syntaxe très précise à respecter pour qu'Android sache dans quel répertoire il doit fouiller.

En partant du nom du répertoire par défaut, on va créer d'autres répertoires qui permettent de préciser à quels types de matériels les ressources de ce répertoire sont destinées. Les restrictions sont représentées par des *quantificateurs* et ce sont ces quantificateurs qui vous permettront de préciser le matériel de destination. 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 14 en tout, dont un bon paquet qu'on utilise rarement, j'ai donc décidé de les ignorer).

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
- normal
- large
- xlarge

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.

Version d'Android

Priorité : 14
Il s'agit du niveau de l'API (v3, v5, v7, etc.).

Dans les répertoires vus précédemment, que se passe-t-il si l'écran du terminal de l'utilisateur a une grande résolution ? Android ira chercher dans `res/drawable-hpdi` ! 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 ? Et bien… il ira chercher dans `res/drawable-hpdi` 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/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, *il faut mettre les quantificateurs par ordre croissant de priorité.*
- `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 faut 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 chaque projet.

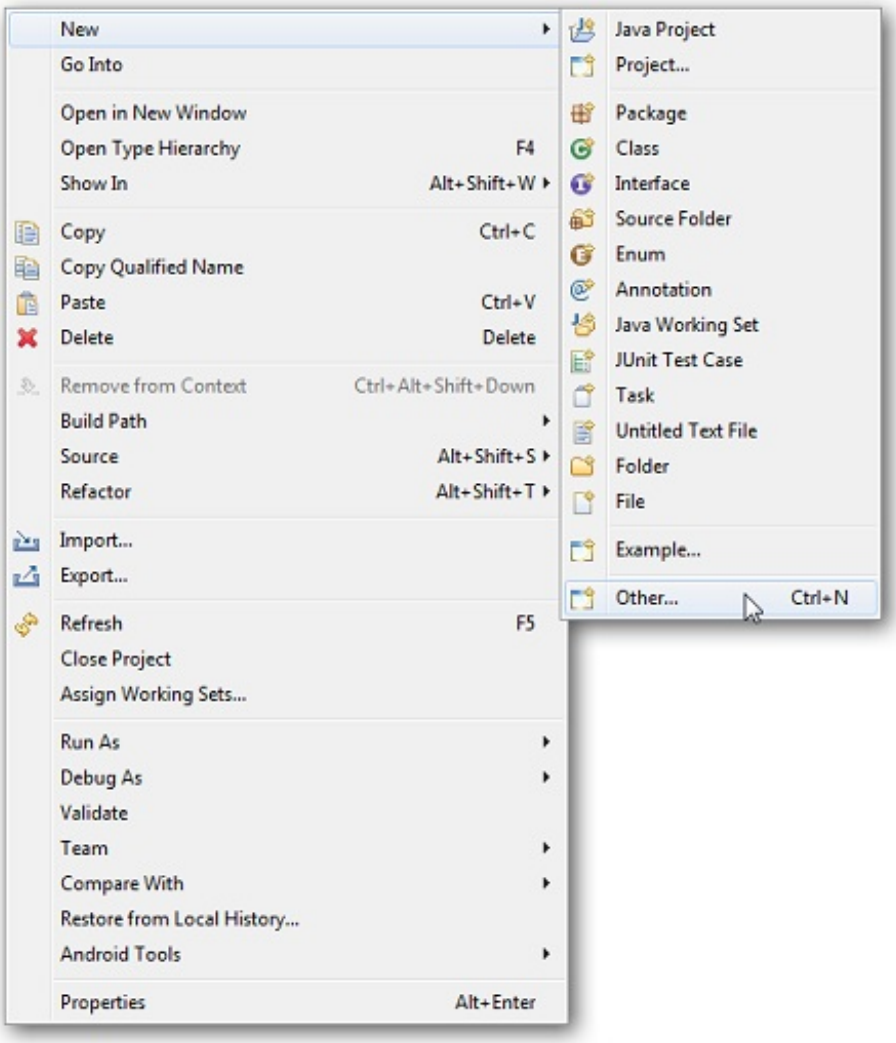
- `res/drawable-hdpi`
- `res/drawable-ldpi`
- `res/drawable-mdpi`
- Pas de `res/drawable`
- `res/layout-land`
- `res/layout`

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

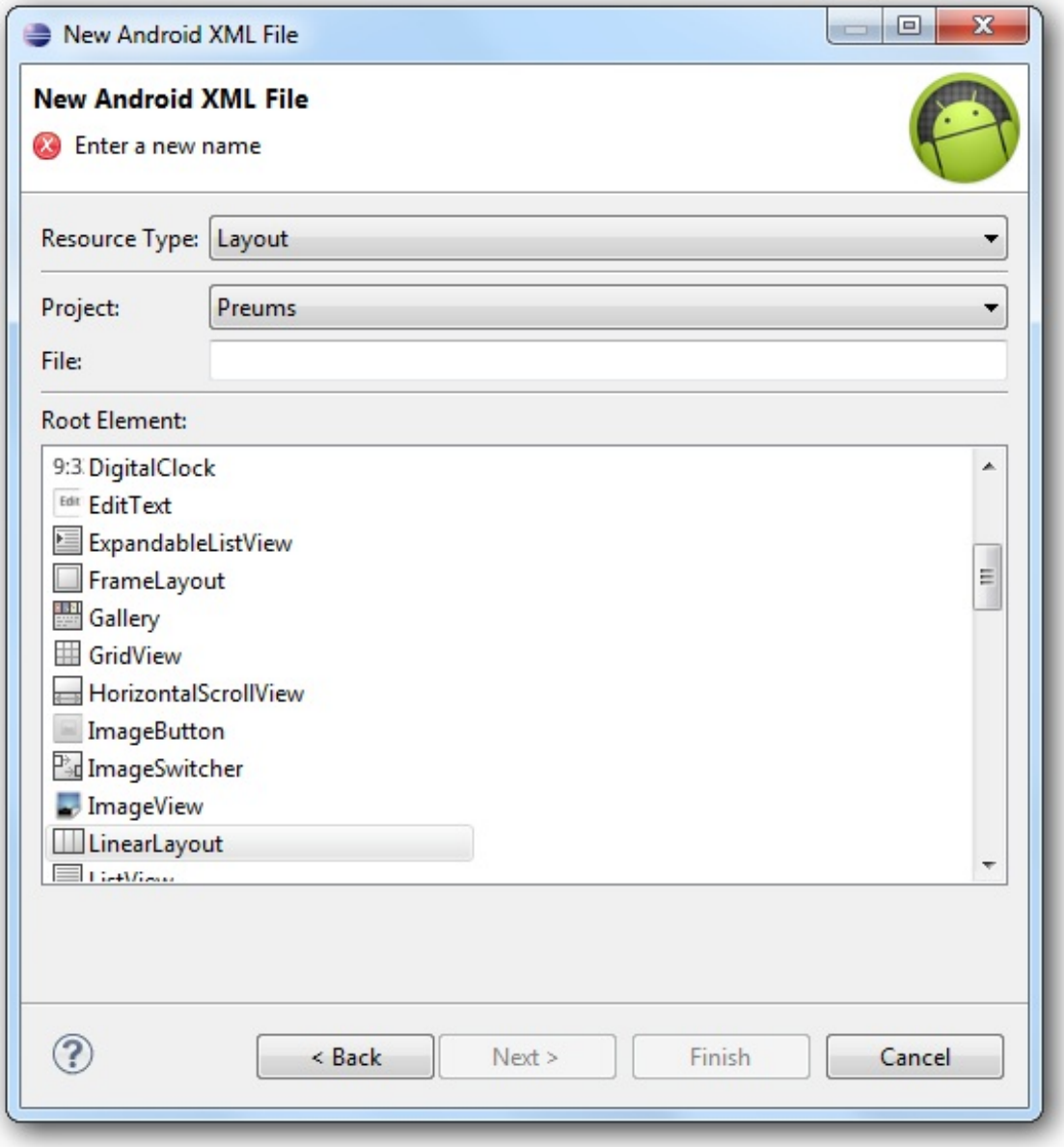
Pour finir, sachez que les écrans de taille `small` se font rares.

Ajouter un fichier avec Eclipse

Heureusement, les développeurs de l'ADT ont pensé à nous en créant un petit menu qui vous aidera à créer des répertoires de manière simple, sans avoir à retenir de syntaxe. Par contre 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:

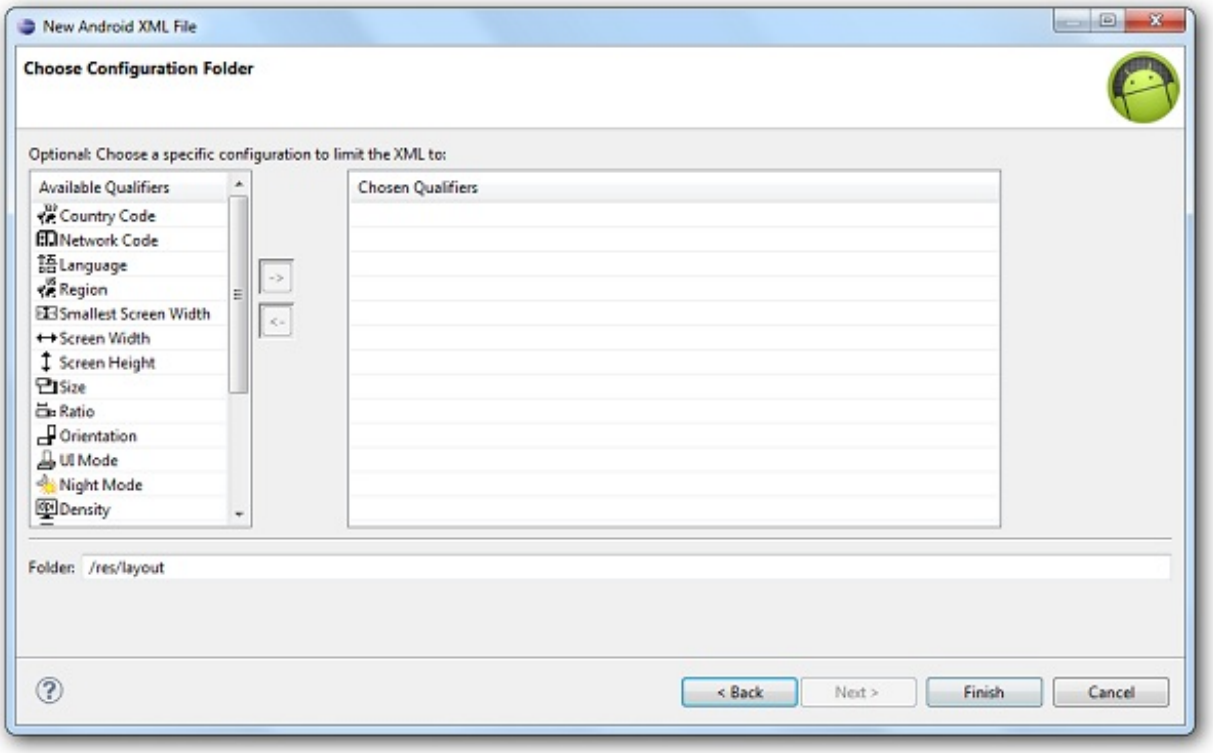


Dans le sous-menu `New`, cliquez sur `Other...` puis cherchez `Android XML File` dans le répertoire `Android`. Cette opération ouvrira un assistant de création de fichiers XML.



Le premier champ vous permet de sélectionner quel type de ressources vous voulez. 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`. Logique, nous allons créer un fichier XML et les fichiers XML devraient en théorie ne pas se situer dans ce répertoire. 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 qui vous permettra de choisir des quantificateurs pour votre ressource ou `Finish` pour que le fichier soit créé dans un répertoire sans quantificateurs.



La section suivante 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. 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 choisir une langue qui n'existe pas, le quantificateur ne s'ajoutera pas dans le champ du répertoire. Si ce champ ne vous semble pas correct vis-à-vis des quantificateurs sélectionnés, c'est que vous avez fait une faute d'orthographe. Si vous écrivez directement un répertoire dans `Folder`, les quantificateurs indiqués s'ajouteront dans la liste correspondante.

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

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

Petit exercice

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

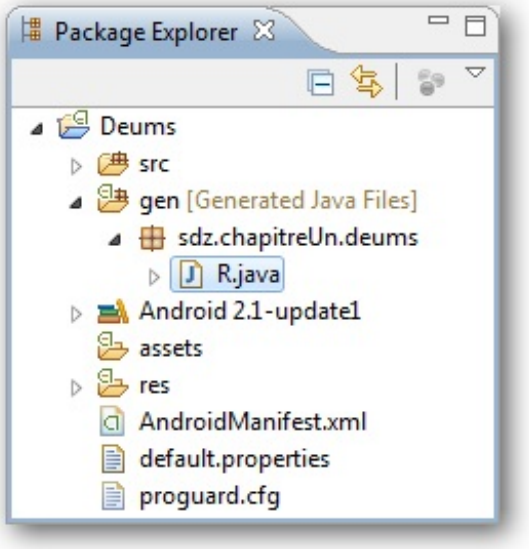
Le `Folder` doit contenir **exactement** `/res/layout-fr-rBE-port-mdpi-v8.</secret>`

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

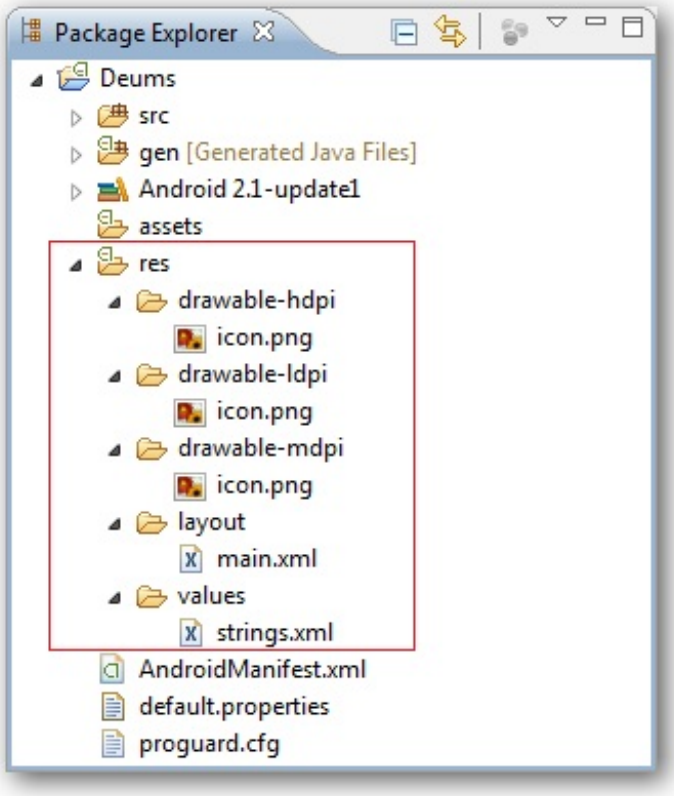
Récupérer une ressource

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) :



Ouvrez donc ce fichier. Ça vous rappelle quelque chose ? Comparons avec l'ensemble des ressources que comporte notre projet.



Code : Java

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
*
* This class was automatically generated by the
* aapt tool from the resource data it found. It
* should not be modified by hand.
*/

package sdz.chapitreUn.deums;
```

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

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

La classe `Layout`

Code : Java

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

Il s'agit d'une classe déclarée dans une autre classe : c'est ce qui s'appelle une classe interne. Une classe interne est une classe déclarée à l'intérieur d'une autre classe, sans grande particularité. 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 sont 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.
- 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.

Regardez les exemples suivants :

Code : Java

```
int deuxNorm = 2; // Valide !
int deuxHexa = 0x00000002; // Valide, et vaut la même chose que « deuxNom »
int deuxRed = 0x2; // Valide, et vaut la même chose que « deuxNom » et « deuxHexa » (évidemment, 00000002 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 les autres lettres !
int caVaPasLaTete = 0x124!AB57; // Alors là c'est carrément
n'importe quoi !
```

Cet entier a le même nom qu'un fichier de ressources, tout simplement parce qu'il représente ce fichier. 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 `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`

Code : Java

```
public static final class drawable {
    public static final int icon=0x7f020000;
}
```

Contrairement au cas précédent, on a un seul entier pour 3 fichiers ! On a vu dans la section précédente qu'il fallait nommer de façon identique ces 3 fichiers qui ont la même fonction, pour une même ressource, mais avec des quantificateurs différents. Et bien quand vous ferez appel à l'identificateur, Android saura qu'il lui faut le fichier `icon` 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`

Code : Java

```
public static final class string {
    public static final int app_name=0x7f040001;
    public static final int hello=0x7f040000;
}
```

Cette fois, si on a deux entiers, c'est tout simplement parce qu'on a deux chaînes de caractères dans le répertoire `res/values` qui contient les chaînes de caractères (qui sont des données). Comme on a qu'un fichier `strings.xml` dans ce répertoire, on en déduit que c'est lui qui contient la déclaration de ces deux chaînes de caractères (on verra comment dans un prochain chapitre).

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

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` (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 identifiant_de_la_ressource)`** de la classe `TextView`.

Solution

Code : Java

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

public class Main extends Activity {
    TextView text = null;

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

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

        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` qui se trouve dans la classe `String` puisqu'il s'agit d'une chaîne de caractères, donc `R.string.hello`.

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`.

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. 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 caractère `hello` grâce à la méthode `String getString(int identifiant_de_la_ressource)`.
- Et on modifie la chaîne récupérée.

Solution

Code : Java

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

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

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

        hello = getResources().getString(R.string.hello);
        hello = hello.replace(", ", ", poil dans la ");

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

        setContentView(text);
    }
}
```

Partie 2 : Création d'interfaces graphiques

Constitution des interfaces graphiques

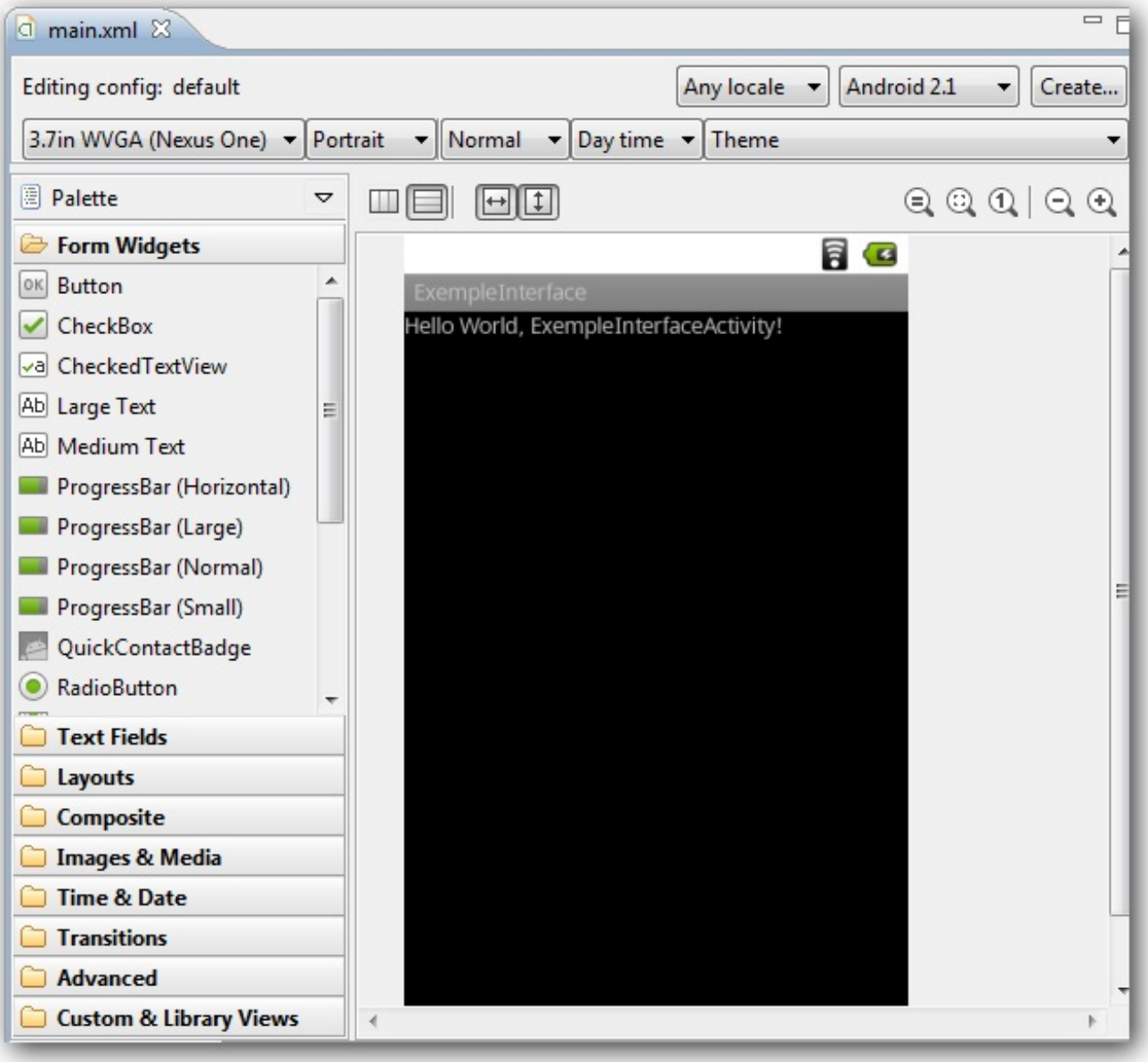
Bien, maintenant que vous avez compris le principe et l'utilité des ressources, voyons comment appliquer nos nouvelles connaissances aux interfaces graphiques. Avec la diversité des machines sous lesquelles fonctionne Android, il faut vraiment exploiter toutes les opportunités offertes par les ressources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

Une application Android polyvalente possède un fichier XML pour chaque type d'écrans, 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. Pratique non ? Voyons voir un peu comment ça fonctionne.

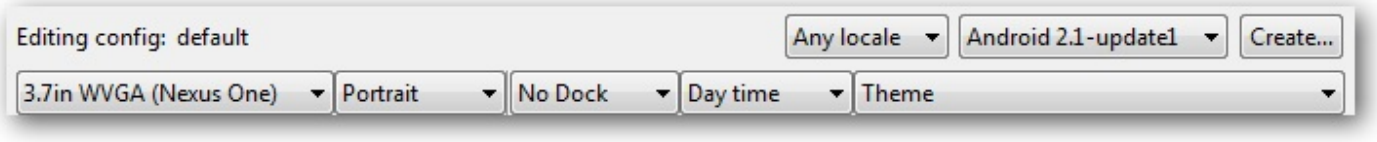
Ouvrez le seul fichier qui se trouve dans le répertoire `res/layout`. Il s'agit normalement du fichier `main.xml`. Une fois ouvert, vous devriez avoir quelque chose qui ressemble à ceci :



Cet outil vous aide à mettre en place les vues directement dans le layout de l'application, représenté par le rectangle noir à droite. Cependant, il ne peut remplacer la manipulation de fichiers XML, c'est pourquoi 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 que vous pourrez observer le résultat selon différentes options :



On va pour l'instant se concentrer sur la seconde partie du menu, celle avec les 5 listes déroulantes :

- La première liste déroulante permet de sélectionner un type d'écran. Le chiffre indique la taille de la diagonale en pouces (sachant qu'un pouce fait 2,54 centimètres, la diagonale du **Nexus One** fait $3,7 * 2,54 = 9,4$ 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.
- La seconde permet d'observer le comportement en mode portrait ou en mode paysage.
- La troisième permet d'observer le comportement quand le périphérique est associé à un matériel d'amarrage. Dans la majorité des cas il ne nous servira pas, d'ailleurs je n'ai même pas présenté le quantificateur adéquat.
- La quatrième permet d'observer le comportement en fonction de la période de la journée dans laquelle on se trouve. Encore une fois, peu d'applications changent de mise en page en fonction de l'heure.
- La dernière liste permet de choisir un thème. Nous aborderons plus tard les thèmes et les styles.

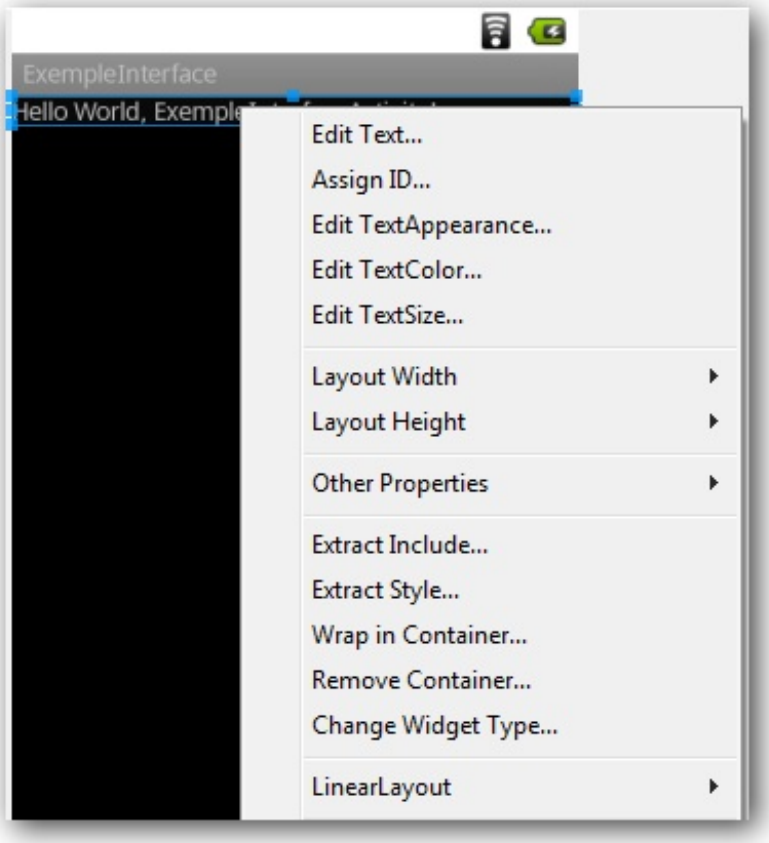
Occupons-nous maintenant de la première partie :

- La première liste déroulante reste vide si vous n'avez pas précisé de mise en page particulière pour certaines langues. Il vous faudra par exemple créer un répertoire `res/layout-en` pour pouvoir choisir cette option.
- La seconde liste vous permet d'avoir un aperçu de votre interface en fonction de la version de l'API qui sera utilisée.
- La dernière vous permet de créer un nouveau répertoire en fonction de certains critères, mais je préfère utiliser l'outil présenté dans le chapitre précédent.

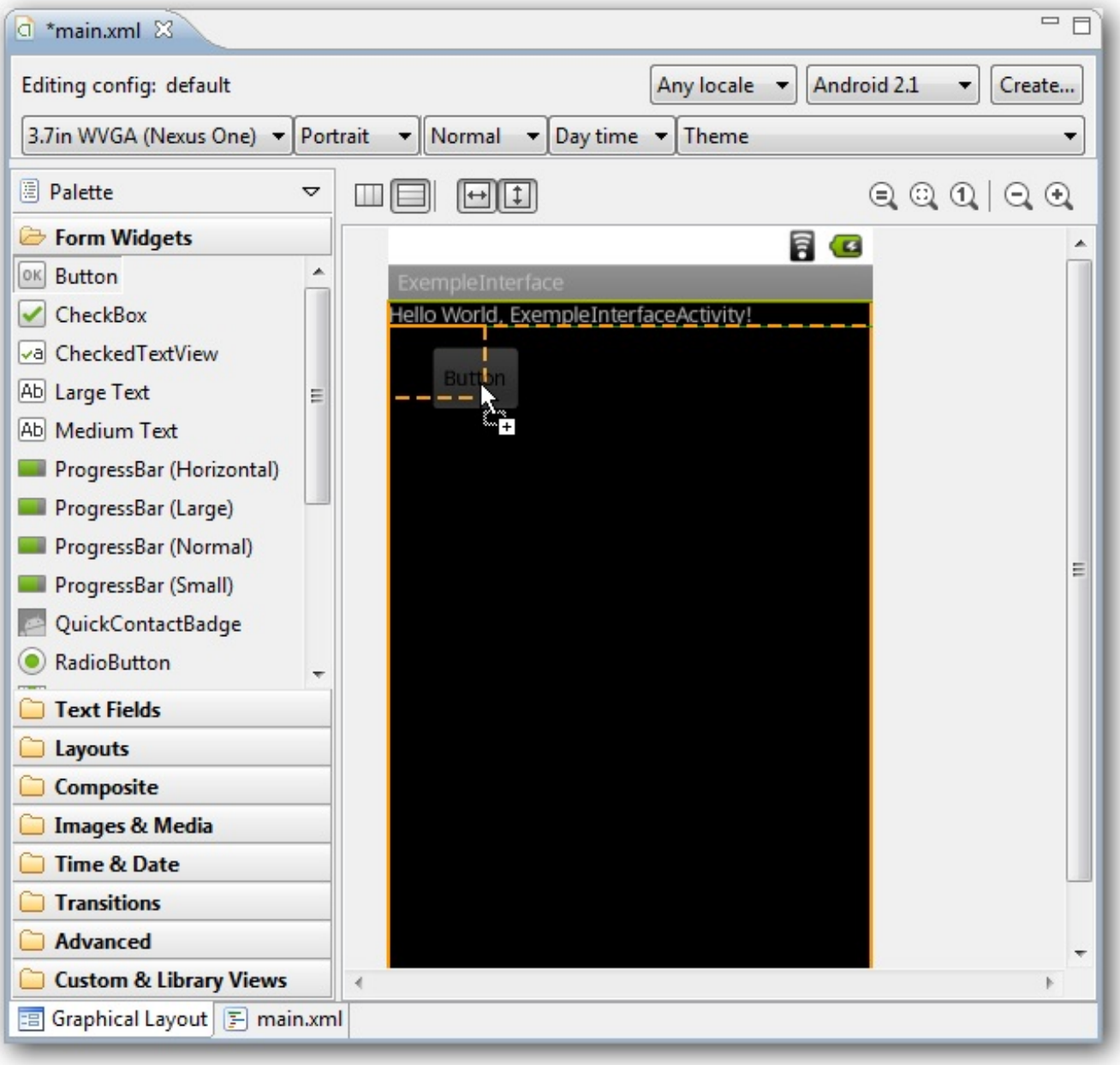
Rien, jamais rien ne remplacera un test sur un vrai terminal. Ne pensez pas que parce votre interface graphique est esthétique dans cet outil elle le sera aussi en vrai. Si vous n'avez pas de terminal, l'émulateur vous donnera déjà un meilleur aperçu de la situation.

Utilisation

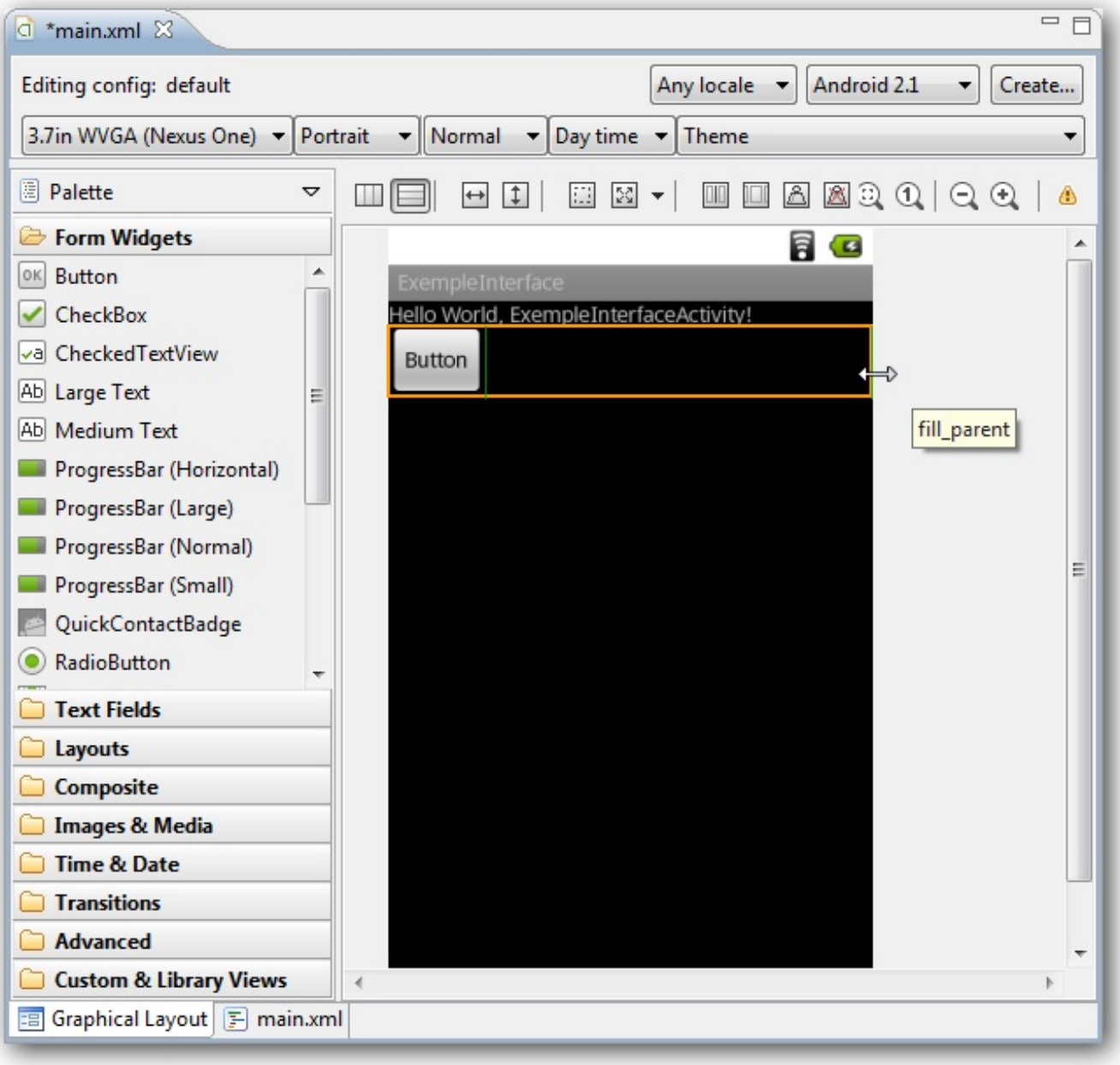
Autant cet outil n'est pas aussi précis, pratique et surtout dénué de bugs que le XML, autant il peut s'avérer pratique pour certaines manipulations de base. Il permet par exemple de modifier les attributs d'une vue à la volée. Sur la figure suivante, vous voyez au centre de la fenêtre une activité qui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



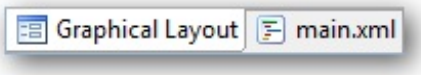
Nous reviendrons plus tard sur ces options, mais retenez bien qu'il est possible de modifier les attributs *via* un clic droit. De plus, vous pouvez placer différentes vues en cliquant dessus depuis le menu de gauche puis en les déposant sur l'activité.



Il vous est ensuite possible de les agrandir ou de les rapetisser en fonction de vos besoins.



Pour accéder au fichier XML correspondant, cliquez sur le second onglet `main.xml`.



Règles générales sur les vues
Différenciation entre un layout et un widget

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

Code : XML

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

```
            android:text="@string/hello"
            />
    </LinearLayout>
```

La racine possède un attribut qui sera toujours `xmlns:android="http://schemas.android.com/apk/res/android"`. Cette ligne permet de préciser que ce document contiendra des attributs spécifiques à Android. Si vous ne la mettez pas, Eclipse refusera de compiler.

On trouve ensuite une racine qui s'appelle `LinearLayout`. 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 :

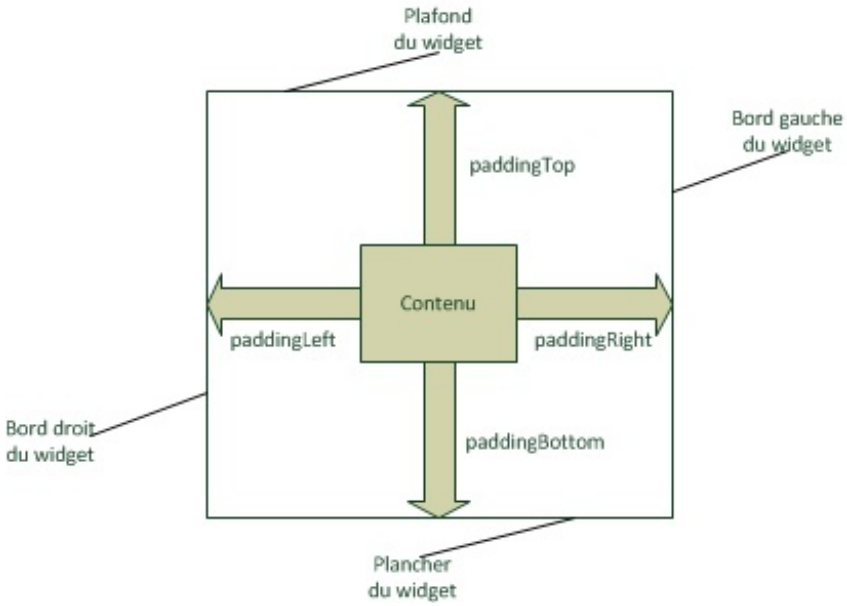
- `dp` ou `dip` : il s'agit d'une unité qui est indépendante de la résolution de l'écran. En effet, il existe d'autres unités comme le pixel (`px`) ou le millimètre (`mm`), mais celles-ci varient d'un écran à l'autre… Par exemple si vous mettez une taille de 500 dp pour un widget, il aura toujours la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



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.

Code : XML

```
<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 espacement_de_gauche, int espacement_du_dessus, int espacement_de_droite, int espacement_du_dessous)`.

Code : Java

```
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. Éh 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. Si on reprend notre classe créée par défaut :

Code : XML

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

```
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
android:text="@string/hello"
    />
</LinearLayout>
```

On devine d'après la ligne 10 que le `TextView` affichera le texte de la ressource qui se trouve dans la classe `String` de `R.java` et qui s'appelle `hello`. 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 `main.xml`. Eh bien c'est tout simplement parce qu'il faut créer cet identifiant nous-même (dans le fichier XML, 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 plait. 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 lui donner un *padding* de 25.7dp, ce qui nous donne :

Code : XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
android:id="@+id/text"
android:padding="25.7dp"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

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

Code : Java

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
        public static final int truc=0x7f020001;
    }
public static final class id {
public static final int text=0x7f050000;
```

```
    }
        public static final class layout {
            public static final int main=0x7f030000;
        }
        public static final class string {
            public static final int app_name=0x7f040002;
            public static final int hello=0x7f040000;
            public static final int histoire=0x7f040001;
        }
    }
```

Instanciation des objets XML

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

On cast ? Aucune idée de ce que ça 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 `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 **cast** `Classe_2` en `Classe_1`, et on le fait avec la syntaxe suivante :

Code : Java

```
//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.main`, il s'agit donc du layout d'identifiant `main`, autrement dit celui que nous avons manipulé un peu plus tôt.

Code : Java

```
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.main);

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

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

Code : Java

```
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.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 **dé-sérialisation**. C'est à ça que sert la fonction `View findViewById (int identifiant)`. Le problème est que cette méthode va aller chercher dans l'arbre des vues que l'activité va créer. 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 notre_activite, int ressource_a_recuperer, ViewGroup parent_de_la_ressource)`. Cette méthode va dé-sérialiser l'arbre XML au lieu de l'arbre de vues qui est créé par l'activité.

On aurait aussi pu procéder comme dans le chapitre précédent en utilisant le gestionnaire de ressources pour récupérer le fichier XML du layout puis le parcourir pour récupérer la bonne vue… Mais sincèrement, ça aurait été beaucoup d'investissement pour pas grand chose. 

Code : Java

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

public class Main extends Activity {
    LinearLayout 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 layout, c'est-à-dire
LinearLayout
layout = (LinearLayout) LinearLayout.inflate(this, R.layout.main,
null);
        // … puis on récupère TextView grâce à son identifiant
```

```
        text = (TextView) layout.findViewById(R.id.ganbare);
                text.setText("Et cette fois, ça fonctionne !");
                setContentView(layout);
                // On aurait très bien pu utiliser «
        setContentView(R.layout.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 ça vous voulez dire ?

Code : Java

```
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.main);

        monTexte = (TextView)findViewById(R.id.text);
        monTexte.setPadding(50, 60, 70, 90);

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

Eh oui ! 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 ! De toute façon celle d'avant n'aura absolument aucun effet, si ce n'est nous faire perdre du temps pour rien.

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 (ça 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

Code : 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 caractère 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

Code : Java

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

Vous remarquerez que l'équivalent de <minicode">#112233</minicode> est `0x112233` (il suffit de remplacer le # par « 0x »).

Rendu

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

Code : XML

```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 email, alors l'arobase sera proposé tout de suite à l'utilisateur sur le clavier. Vous trouverez une liste de tous les `inputTypes` possible ici.
- Enfin, on peut préciser la taille en lignes que doit occuper l'EditText avec `android:lines`.

Exemple en Java

Code : Java

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

Rendu

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

Code : XML

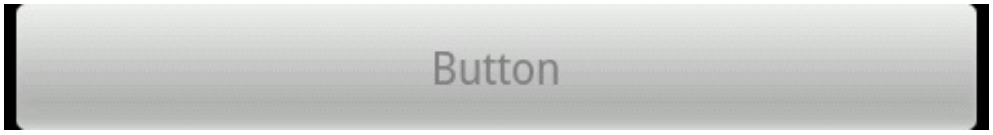
```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/button" />
```

Exemple en Java

Code : Java

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

Rendu



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

Code : 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

Code : 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

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 à plusieurs 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

Code : 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

Code : 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

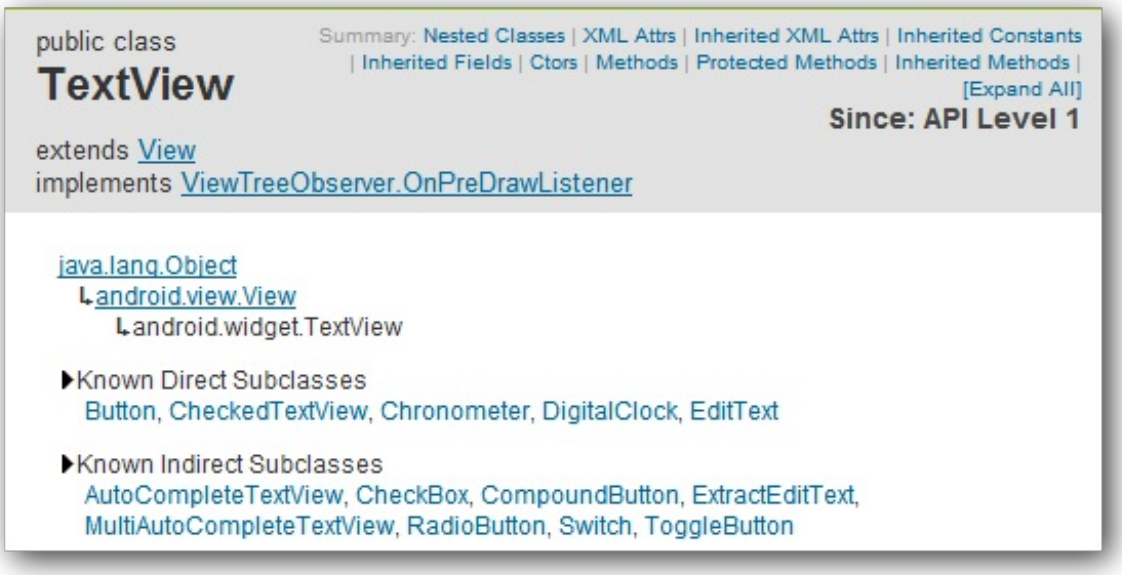
Utiliser la documentation pour trouver une information

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 un bon développeur Android -voire un bon développeur tout court- si vous ne savez pas chercher des informations par vous-mêmes.

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

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

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` :



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.

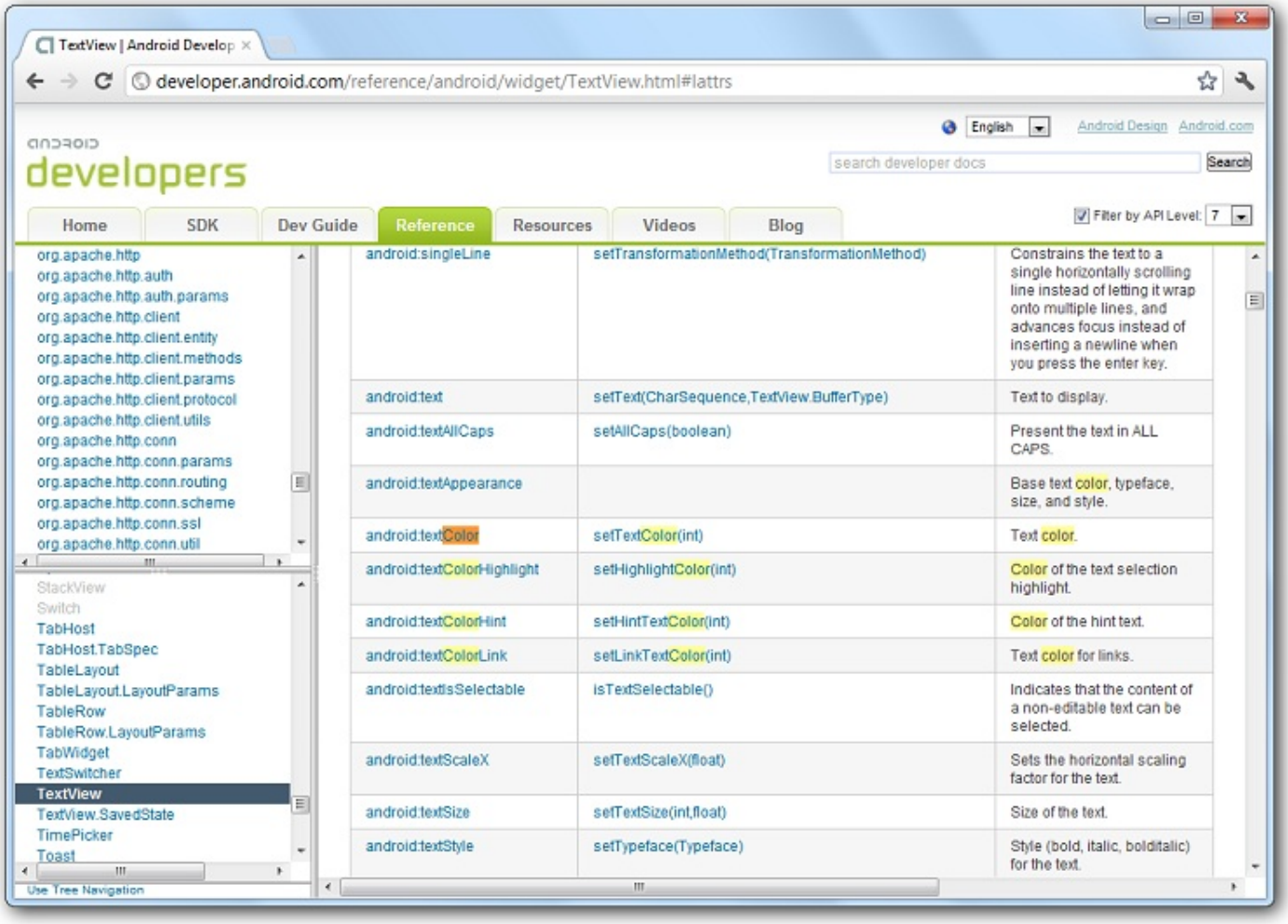
Ensuite, on peut voir les classes qui dérivent directement de cette classe (*Known Direct Subclasses*) et les classes qui en dérivent indirectement, c'est-à-dire qu'un des ancêtres de ces classes dérive de `View` (*Known Indirect Subclasses*).

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 été 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, ou alors je peux effectuer une recherche sur la page (le raccourcis standard pour cela est Ctrl + F :



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

Calcul de l'IMC - Partie 1

Énoncé

On va commencer un mini TP (TP signifie « Travaux Pratiques» ; ce sont des exercices pour vous entraîner à programmer). Vous voyez ce qu'est l'IMC ? 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 chiffre (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 obése d'après ce chiffre.

Pour l'instant, on va se contenter de faire l'interface graphique. Voici à quoi elle ressemblera :

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 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.
- Afin de centrer du texte dans un `TextView`, on utilise l'attribut `android:gravity="center"`.

Voici le layout de base :

Code : XML

```
<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 là -->

</LinearLayout>
```

Solution

Code : XML

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

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 récupère l'évènement et permet de le traiter. Ce type d'objet s'appelle un **Listener**. Un Listener est une interface dont chaque méthode correspond à une action qu'il est possible de faire à partir d'un évènement.

Par exemple, pour intercepter l'évènement « clic » sur un `Button`, on appliquera l'interface `View.OnClickListener` sur ce bouton. Cette interface contient la méthode `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 quoi faire en cas de clic. Il existe plusieurs méthodes pour utiliser ces `Listener`.

Pour gérer d'autres évènements, il existe d'autres interfaces avec d'autres méthodes à implémenter (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_de_la_touche_pressee, KeyEvent informations_sur_l_evenement)`. 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 » ?

C'est très simple, il faut indiquer à Android quand l'évènement a été effectué avec succès. Si la méthode a fait ce qu'on voulait qu'elle fasse, alors elle peut retourner **true**. Cependant si l'action que vous souhaitiez exécuter a échoué (par exemple vous vouliez diviser par zéro, ce qui bien entendu est impossible), la méthode ne doit pas renvoyer **false**, mais doit faire appel à la superclasse. Par exemple pour la fonction `onLongClick` on ferait :

Code : Java

```
boolean onLongClick(View vue) {
    //Ce que vous souhaitiez faire dans la méthode
   return super.OnLongClick(vue);
}
```

Nous allons maintenant voir les différentes façons d'utiliser ces Listener.

Par héritage

On va faire implémenter un `Listener` à notre classe, ce qui veut dire que l'activité essayera elle-même d'intercepter des évènements. Et n'oubliez pas que lorsqu'on implémente une interface, il faut nécessairement implémenter toutes les méthodes de cette interface. Enfin, on peut toujours laisser une méthode vide si on ne veut pas se préoccuper de ce style d'évènements.

Un exemple d'implémentation :

Code : Java

```
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;

public class Main extends Activity implements View.OnTouchListener,
View.OnClickListener {
    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… donc le contenu de cette méthode s'exécutera quel que soit le bouton sur lequel on a cliqué. 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 :

Code : Java

```
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 second 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 Listener, 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 :

Code : Java

```
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 :

Code : Java

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

public class AnonymousExampleActivity extends Activity {
    private Button touchAndClick = null;
    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 true;
     }
 });

        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.

Code : Java

```
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 false;
 }
    };

    private OnTouchListener touchListenerBouton3 = new
View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
```

```
            /* Réagir au toucher pour le bouton 3*/
            return false;
                }
        };

        Button b = null;
        Button b2 = null;
        Button b3 = null;

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

            setContentView(R.layout.main);

            b = (Button) findViewById(R.id.boutton);
            b.setOnTouchListener(touchListenerBouton1);
            b.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 toucher mais du clic, 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 taille du `TextView` sera fixée avec la méthode `setText(Math.abs(coordonnee_x - largeur_du_bouton / 2) + Math.abs(coordonnee_y - hauteur_du_bouton / 2))`.
- Pour obtenir la coordonnée en abscisse (X) on utilise `float getX ()` d'un `MotionEvent`, et pour obtenir la coordonnée en ordonnée (Y) on utilise `float getY ()`.

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

Code : XML

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

Code : Java

```
// 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 cast 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) du
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 puisque l'évènement a bien été trait
é
        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 $\frac{\text{Poids (en Kilogramme)}}{\text{Taille (en Metre)} \times \text{Taille (en Metre)}}$.

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

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

Consignes

- Voici la syntaxe pour construire un `Toast` : `static Toast makeText(Context le_context_d_ou_est_lance_le_toast, CharSequence le_texte_a_afficher, int la_duree_d_affichage)`. 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.
- Parce que c'est déjà bien assez compliqué comme ça, on se simplifie la vie et on ne prend pas en compte les cas extrêmes (taille ou poids < 0 ou **`null`** par exemple).

Ma solution

Code : Java

```
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 TroimsActivity 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 méga fonction
    private final String megaString = "Vous faites un poids parfait
! Wahou ! Trop fort ! On dirait Brad Pitt (si vous êtes un
homme)/Angelina Jolie (si vous êtes une femme)/Willy (si vous êtes
un orque) !";

    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.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 attribut un Listener adapté aux vues qui en ont besoin
     envoyer.setOnClickListener(envoyerListener);
     raz.setOnClickListener(razListener);
     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 true;
 }
    };

    // Uniquement pour le bouton "envoyer"
    private OnClickListener envoyerListener = new OnClickListener()
{
        @Override
        public void onClick(View v) {
            if(!mega.isChecked()) {
         // Si la mega fonction 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(TroimsActivity.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 seconde 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 mega fonction.
           private OnClickListener checkedListener = new OnClickListener()
    {
               @Override
       public void onClick(View v) {
            // On remet le texte par défaut si c'était le texte de la méga
    fonction qui était écrit
            if(!((CheckBox)v).isChecked() &&
    result.getText().equals(megaString))
                result.setText(defaut);
       }
           };
    }
```

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 !

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 ça et qu'il existe une très grande variété d'architectures différentes. C'est pourquoi nous allons maintenant étudier les différents layouts, afin d'apprendre à placer nos vues comme nous le désirons. Nous pourrons ainsi concevoir une application plus attractive, plus esthétique et plus ergonomique ! 

LinearLayout : placer les éléments sur une ligne

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

On peut lui donner deux valeurs :

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

On va faire quelques expériences pour s'amuser !

Premier exemple

Code : XML

```
<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 :

- 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.

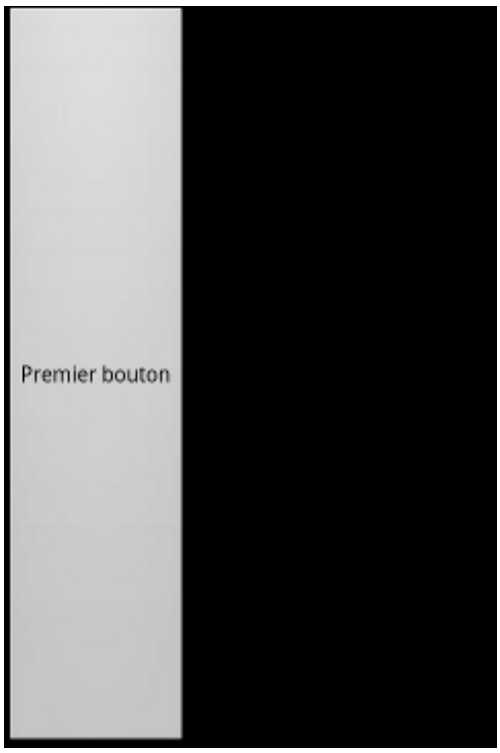
Deuxième exemple

Code : XML

```
<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>
```

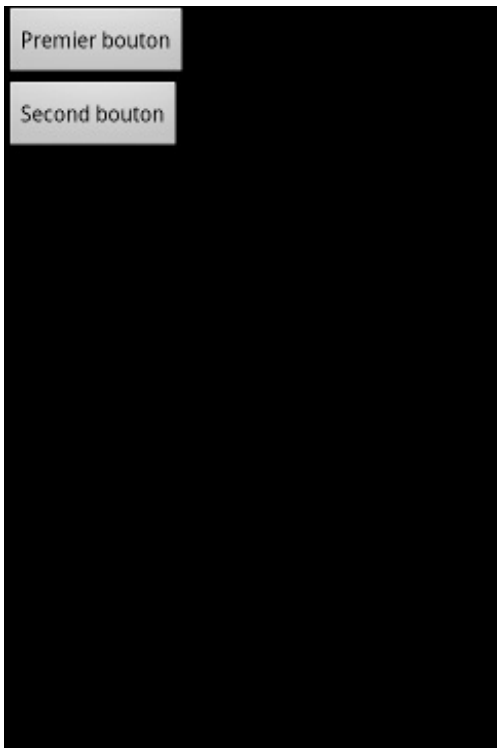


- Le `LinearLayout` est vertical et prend toute la place de son parent.
- Le premier bouton prend toute la place de son parent en hauteur et uniquement la taille nécessaire en largeur.
- Comme le premier bouton prend toute la place, alors le pauvre second bouton se fait écraser  . C'est pour ça qu'on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

Troisième exemple

Code : XML

```
<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>
```

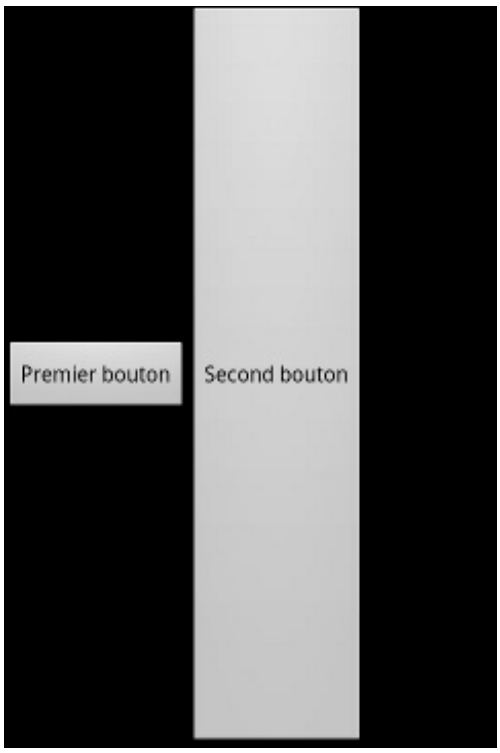


- Le `LinearLayout` est vertical et prend toute la place en largeur mais uniquement la taille nécessaire en hauteur : dans ce cas précis, la taille nécessaire sera calculée en fonction de la taille des enfants.
- Le premier bouton prend toute la place possible dans le parent. Comme le parent prend le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

Quatrième exemple

Code : XML

```
<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 `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 2 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 « pèse » le moins 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.

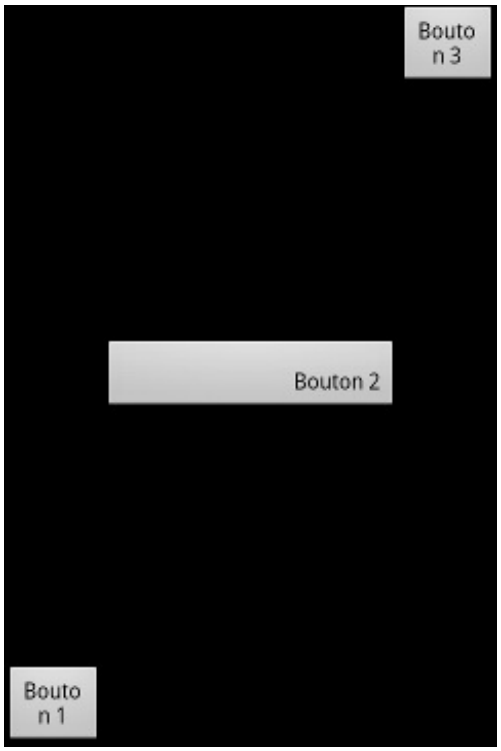
Dernier attribut particulier pour les widgets de ce layout, `android:layout_gravity`, qu'il ne faut pas confondre avec `android:gravity`. `android:layout_gravity` vous permet de déterminer comment se placera la vue dans le parent, alors que `android:gravity` vous permet de déterminer comment se placera 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 ? 

Code : XML

```
<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>
```



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 déterminer 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 ceci :

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.

Code : 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">
    <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.

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

Code : XML

```
<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>
```

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

Code : XML

```
<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>
```

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.

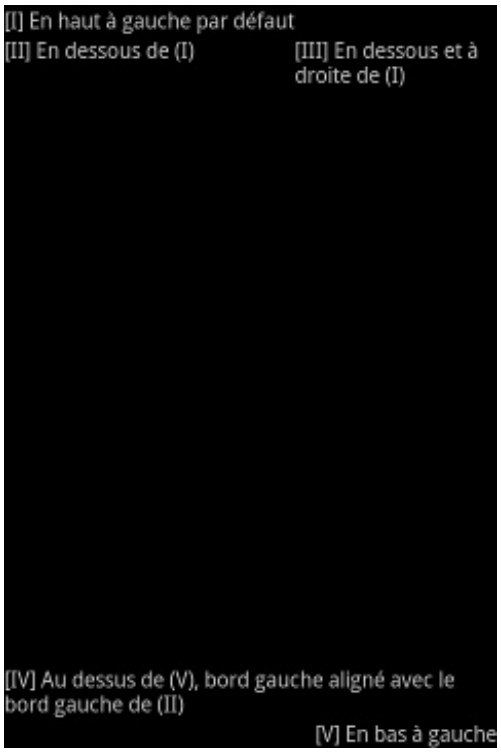
Enfin, il ne faut pas oublier que le principal intérêt de ce layout est de pouvoir placer les éléments les uns par rapport aux autres. Pour cela il existe deux catégories d'attributs :

- 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

Code : XML

```
<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>
```



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 `+` ! Et 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

Secret (cliquez pour afficher)

Code : XML

```
<?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 modification 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 du plus puissant tout en étant l'un des moins nécessiteux 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 :

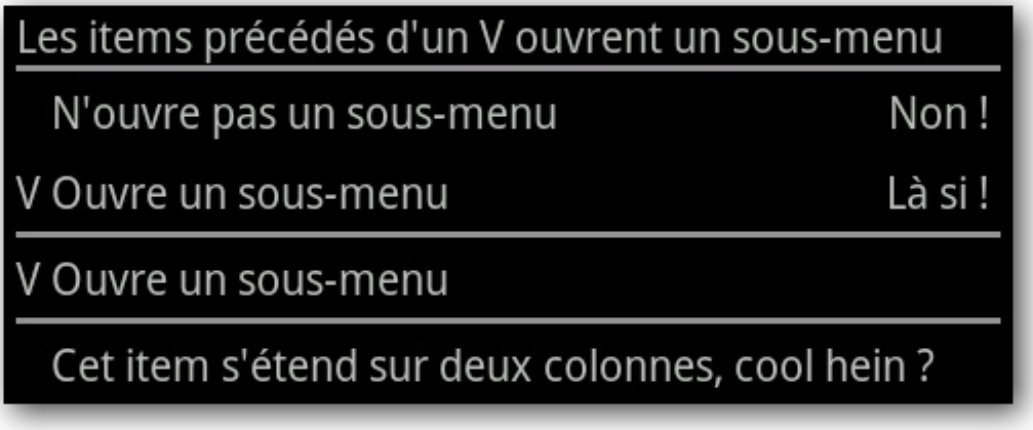
Code : XML

```
<?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>
```

Tous les morceaux de code dans ce paragraphe feront référence à cet exemple-ci

Ce qui donne :



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>**.

Code : XML

```
<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>**

Code : XML

```
<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 composées 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 ! ».

Code : XML

```
<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 seconde colonne grâce à `android:layout_column="1"`.

Code : XML

```
<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 seconde 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 colonne à l'aide de l'attribut `android:layout_span`. Dans notre exemple, le dernier item s'étend de la seconde colonne à la troisième. Il est possible de faire de même sur les lignes avec l'attribut `android:layout_column`.

Code : XML

```
<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 seconde colonne et s'étend sur deux colonnes, donc jusque 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 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

Secret (cliquez pour afficher)

Code : XML

```xml
<?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 en 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. Il peut prendre trois valeurs :

- `visible` (`View.VISIBLE`) la valeur par défaut.
- `invisible` (`View.INVISIBLE`) ne s'affiche pas mais est pris en compte pour l'affichage du layout niveau spatial (on lui réserve de la place).
- `gone` (`View.GONE`) ne s'affiche pas 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`, il les empile les uns au-dessus des autres, avec le premier élément du XML qui se trouve en dernière position, et le dernier ajouté se trouve 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. Il est par ailleurs un peu particulier puisqu'il 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 :

Code : XML

```
<?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 ça ne saurait tarder.

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. 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 de rentrer 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.

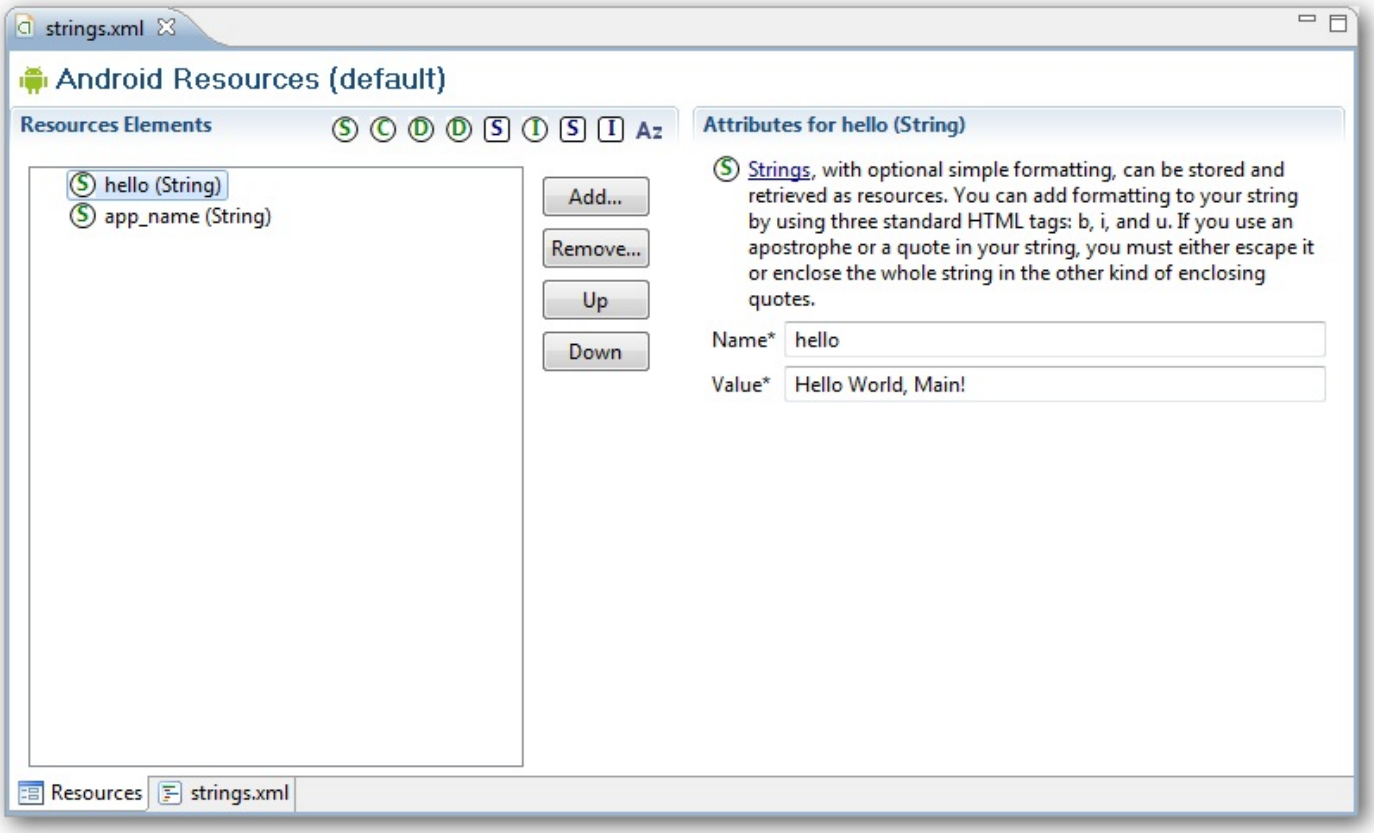
Aspect général des fichiers de ressources

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-là :

Code : XML

```
<?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.



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 types d'organisations :

- 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.

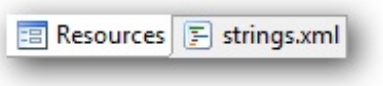
Si vous souhaitez opter pour la seconde organisation, alors le meilleur moyen de s'y retrouver est de savoir trier les différentes ressources à l'aide du menu qui se trouve en haut de la fenêtre. Il vous permet de filtrer la liste des données en fonction de leur type. Voici la signification de tous les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

Icône/bouton	Action
	Afficher uniquement les chaînes de caractères (`String`).
	Afficher uniquement les couleurs (`Color`).
	Afficher uniquement les dimensions (`Dimension`).
	Afficher uniquement les drawables (`Drawable`).
	Afficher uniquement les styles et thèmes (`Style`).
	Afficher uniquement les éléments qui appartiennent à un ensemble (à un tableau par exemple) (`Item`).
	Afficher uniquement les tableaux de chaînes de caractères (`String Array`).
	Afficher uniquement les tableaux d'entiers (`Int Array`).
	Ranger dans 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 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 :



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 :

Code : Java

```
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 :

Code : XML

```
@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"` :

Code : XML

```
<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. En effet, 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 (\).

Code : XML

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

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

Code : XML

```
<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 (****, **<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` ****`"Tom Sawyer"`**** `?` ». Les balises **** vous permettent de mettre du texte en gras.

Instructions

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

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

Ma correction

Le fichier `strings.xml` :

Code : 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 :

Code : Java

```
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 attribut 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 second argument, etc.) ;
- et `$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).

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

On peut le faire en récupérant la ressource :

Code : Java

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

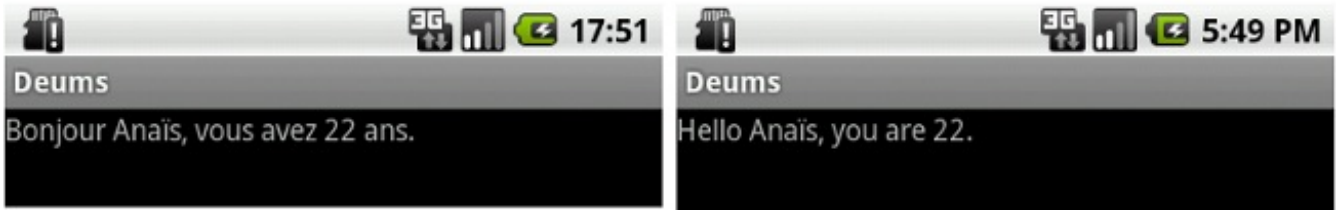
Ou alors sur n'importe quelle chaîne avec une fonction statique de `String` :

Code : Java

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

Application

C'est simple, je vais vous demander d'effectuer l'exemple suivant :



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

Code : XML

```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

Code : XML

```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 :

Code : XML

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

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

Code : Java

```java
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

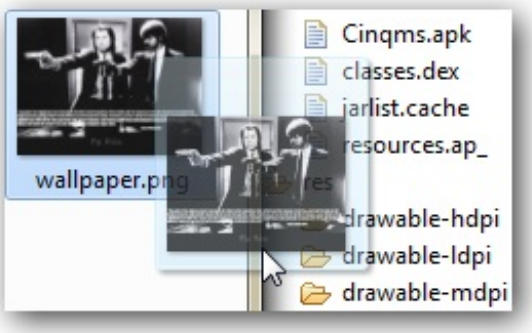
La dénomination « drawable » rassemble tous les fichiers « dessinables » (oui ce mot n'existe pas en français, mais « drawable » n'existe pas non plus en anglais après tout), c'est-à-dire les dessins ou les images. Je ne parlerai que des images puisque ce sont les drawables les plus utilisés et les plus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



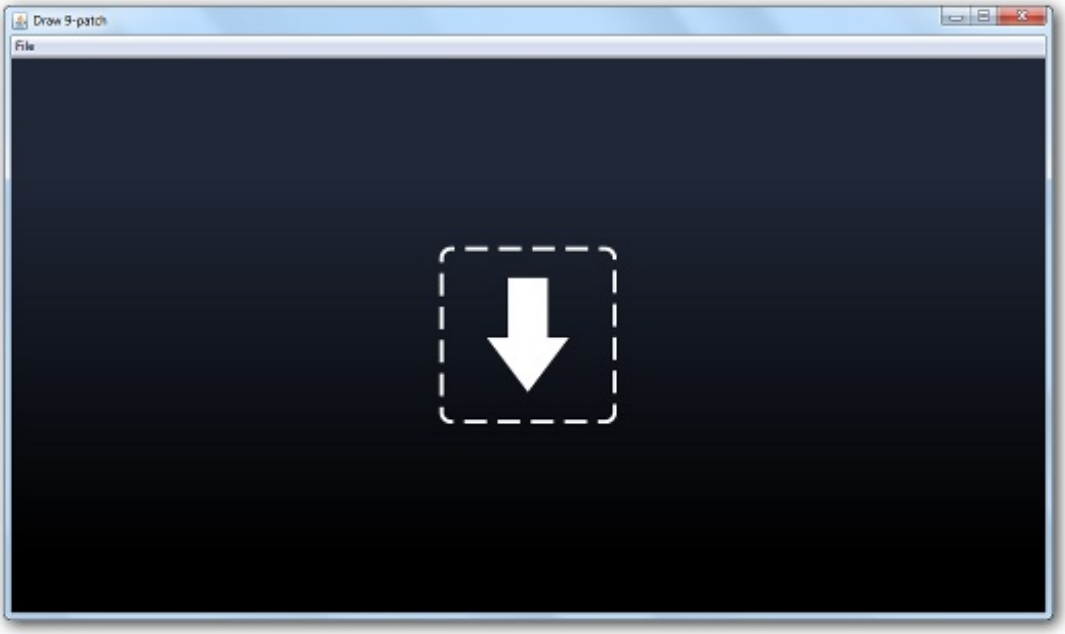
On se contente de glisser-déposer l'image dans le répertoire voulu et Android fera le reste

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`.

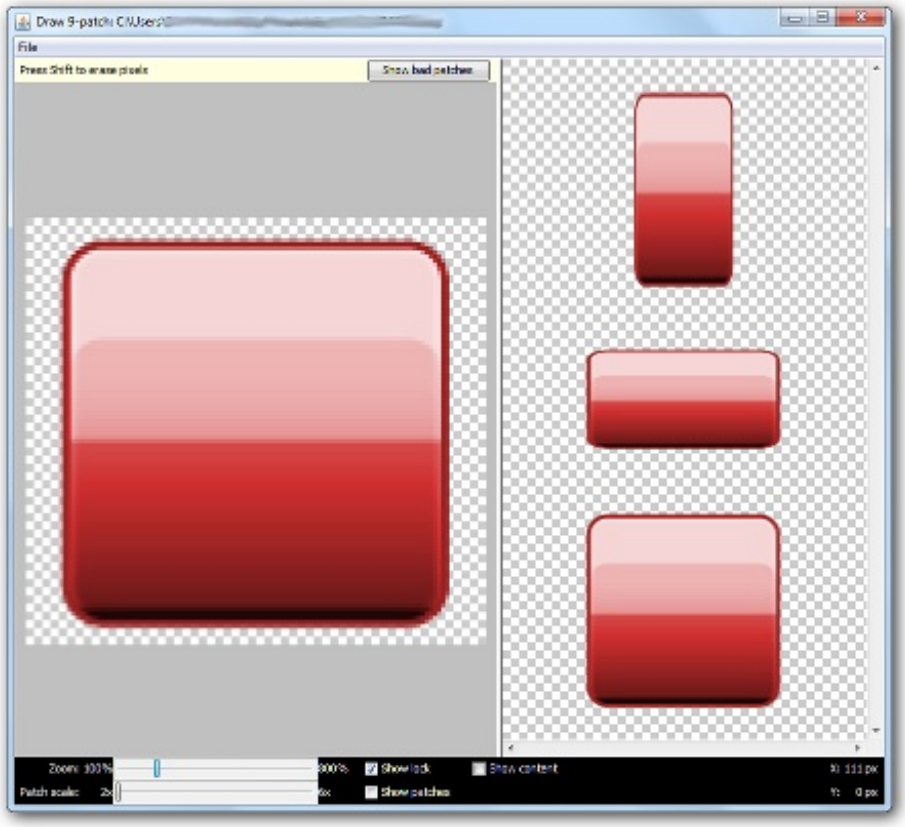
Les images extensibles

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 suivante qui est aimablement prêtée par ce grand monsieur, qui nous autorise à utiliser ses images, même pour des projets professionnels, un grand merci à lui.

Cette image ne paye pas de mine mais elle pourra être étendue jusqu'à former une image immense sans pour autant être toute pixelisée. L'astuce consiste à indiquer quelles parties de l'image peuvent être étendues ; et le SDK d'Android contient un outil pour vous aider dans votre démarche. Par rapport à l'endroit où vous avez installé le SDK, il se trouve dans `\Android\tools\draw9patch.bat`



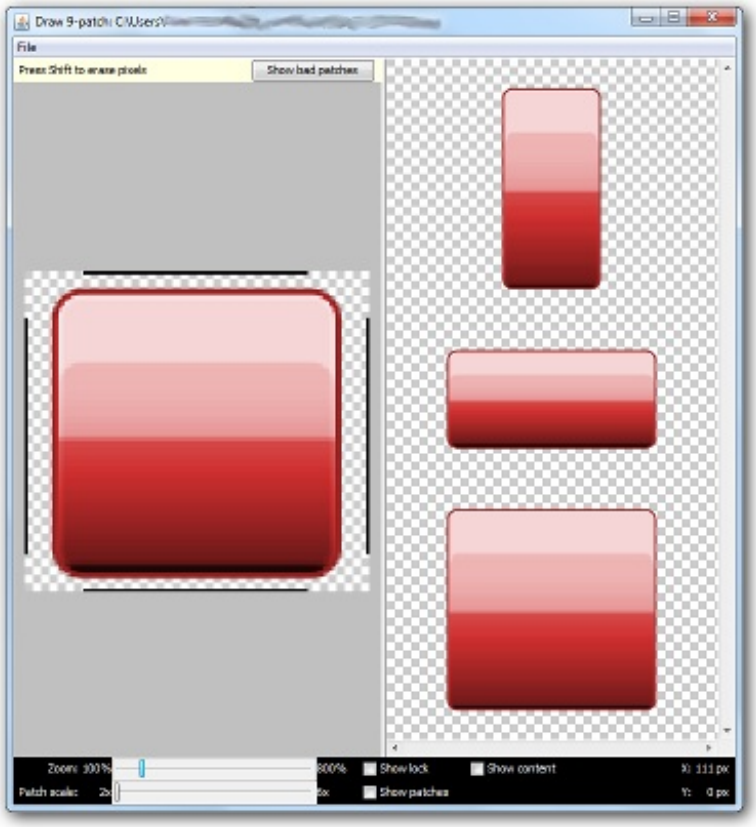
Vous pouvez directement glisser l'image dans l'application pour l'ouvrir ou bien aller dans `File > Open 9-patch...`.



Cet 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 eu ! Vous ne pouvez en fait dessiner que sur la partie la plus extérieur 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) :



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.

Les deux images suivantes vous montreront plus clairement à quoi correspondent les bords :

Les commandes

- Le slider `Zoom` vous permet de vous rapprocher ou vous éloigner de l'image originale.
- Le slider `Patch scale` vous permet de vous rapprocher ou vous éloigner des agrandissements.
- `Show patches` montre les zones qui peuvent être étendue dans la zone de dessin.
- `Show content` vous montre la zone où vous pourrez insérer du contenu (image ou texte) dans Android.

- Enfin, vous voyez un bouton en haut de la zone de dessin, `Show bad patches`, qui une fois coché vous montre les zones qui pourraient provoquer des désagréments une fois l'image agrandie, l'objectif sera donc d'en avoir le moins possible (voire aucune).

Les styles

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

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

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 :

Code : XML

```xml
<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 :

Code : XML

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

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

Code : XML

```xml
<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>
```

Code : XML

```xml
<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 !

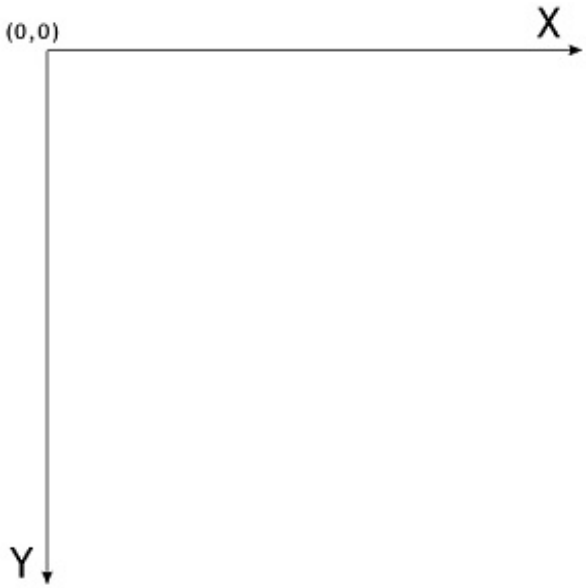
Il est ensuite possible d'attribuer un style à une vue en XML avec l'attribut `style="identifiant_du_style"`. Cependant, un style ne s'applique pas de manière dynamique en Java, il faut alors préciser le style à utiliser dans le constructeur. Regardez le constructeur d'une vue : `public View (Context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

Code : Java

```java
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. 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 « translater » (bouger comme sur un rail rectiligne) une vue.

- `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 :

Code : XML

```
<?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 seconde 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 secondes et soit suivi par un <gitaliqueras>alpha</italique> qui dure 2 secondes, voici ce qu'on écrira :

Code : XML

```
<?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 l'étaient avant le début de l'animation. Heureusement, il est possible de demander à votre animation de changer les vues pour quelles 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`.

Code : Java

```
// 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 l'animation 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 :

Code : XML

```
<?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 :

Code : Java

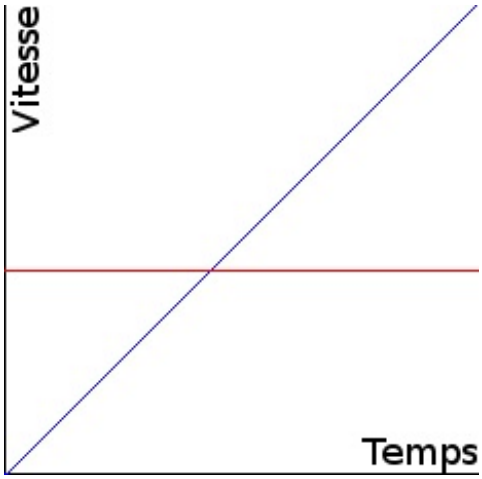
```
LayoutAnimationController animation =
AnimationUtils.loadLayoutAnimation(contexte_dans_lequel_se_situe_la_vue,
identifiant_de_l_animation);
layout.setLayoutAnimation(animation);
```

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 supers, 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 bon en maths pour utiliser les interpolateurs.



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 à utiliser. Préférez plutôt la combinaison des attributs `android:repeatCount` et `android:repeatMode`. Le premier`</minicode>` `définit le nombre de répétitions de l'animation qu'on veut effectuer (-1 pour une 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` **`<minicode`** `type="zcode">`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 à chaque début d'une répétition. C'est aussi simple que :

Code : Java

```
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 ?
    }
});
```

Il existe encore d'autres ressources, dont un bon nombre qui ne sont pas vraiment indispensables. On peut par exemple citer les identifiants, les dimensions, les couleurs, les booléens... Si un jour vous vous intéressez au sujet, vous trouverez plus d'informations sur cette page, et je suis certain que vous vous débrouillerez comme des chefs tellement elles sont faciles à utiliser.

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.

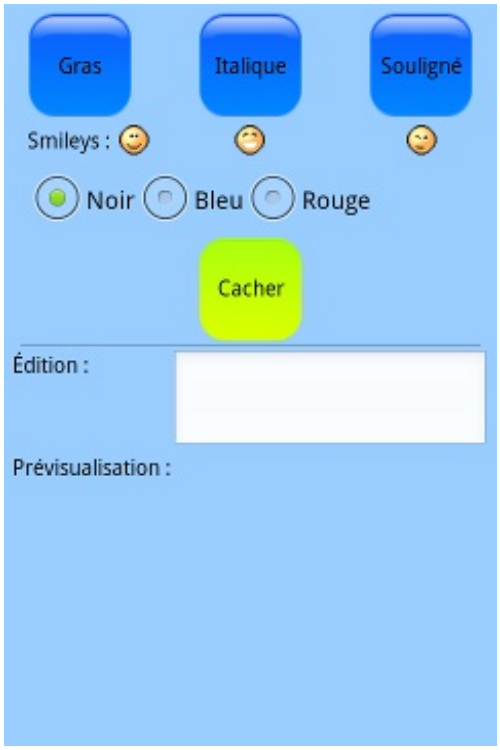
En théorie, vous verrez à peu près tout ce qui a été abordé jusque là, donc s'il vous manque une information, pas de panique, on respire un bon coup et on regarde dans les chapitres précédents en quête d'informations. Je vous donnerai évidemment la solution à ce TP, mais ce sera bien plus motivant pour 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.

Voici ce que donne le résultat final chez moi :



Vous pouvez voir que l'écran se divise en deux zones :

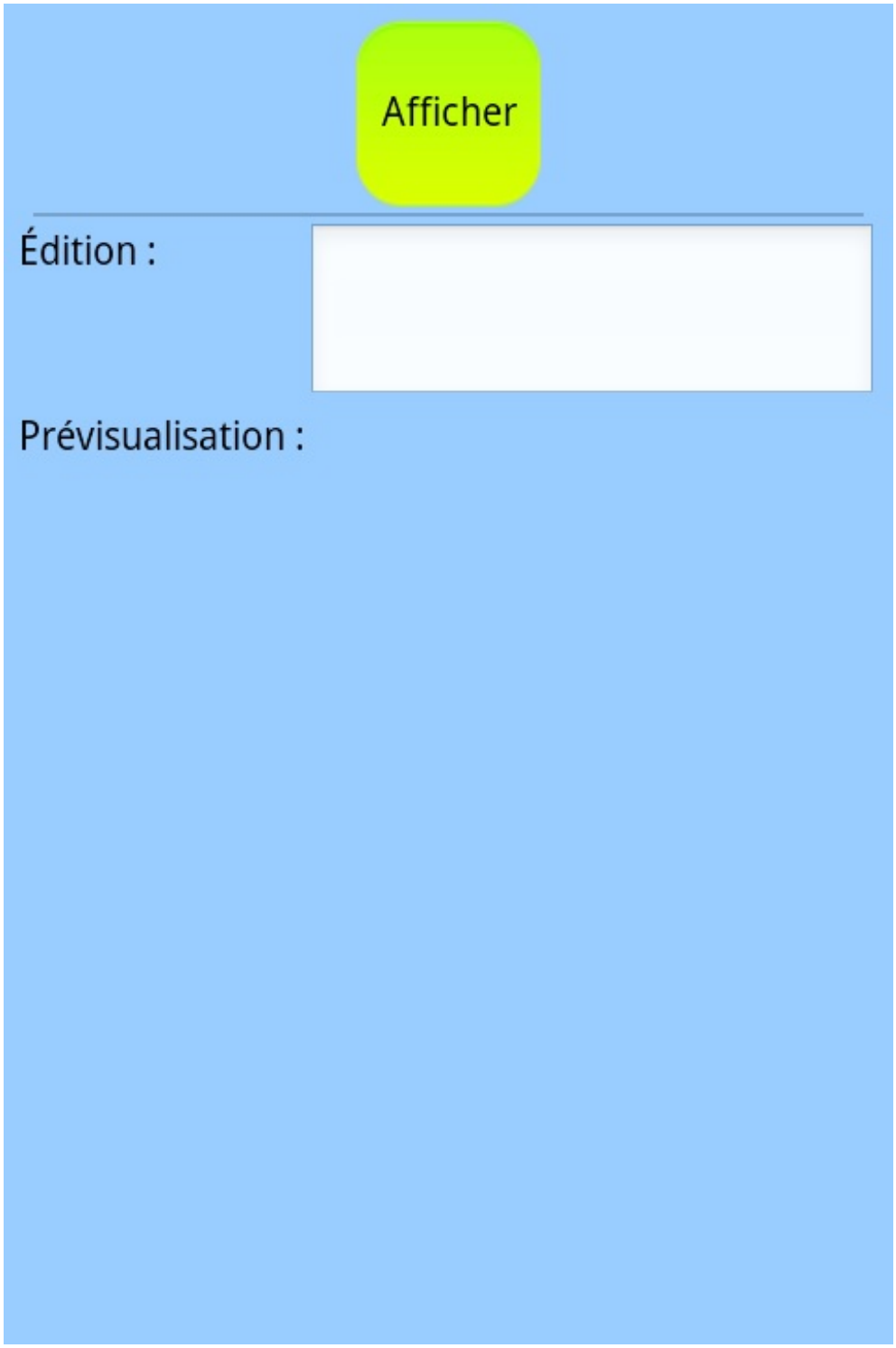
- celle en haut avec les boutons constituera le menu,
- celle du bas avec l'`EditText` et les `TextView`.

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).

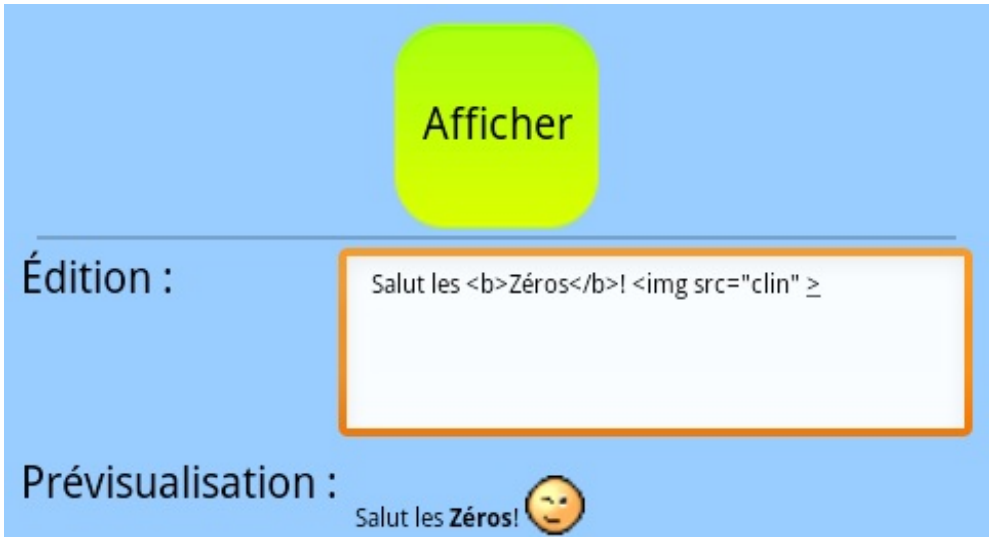
Ce menu est mouvant. En appuyant sur le bouton Cacher, le menu se rétracte vers le haut jusqu'à disparaître. Puis, le texte sur le

bouton devient « Afficher » et cliquer dessus le fait à nouveau descendre.



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 :



Spécifications techniques
Fichiers à utiliser

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

Ensuite, j'ai utilisé des 9-patches pour les boutons. Voici les fichiers :

Le HTML

Les balises

Comme vous avez pu le constater, nos textes seront formatés à l'aide du langage de balisage HTML. Rappelez-vous, je vous avais déjà dit qu'il était possible d'interpréter du HTML dans un `TextView`, cependant on va procéder un peu différemment ici comme je vous l'indiquerai plus tard.

Heureusement, vous n'avez pas à connaître le HTML, juste certaines balises de base que voici :

Effet désiré	Balise
Écrire en gras	`<b>Le texte</b>`
Écrire en italique	`<i>Le texte</i>`
Souligner du texte	`<u>Le texte</u>`
Insérer une image	`<img src="Nom de l'image">`
Changer la couleur de la police	`<font color="Code couleur">Le texte</font>`

L'évènementiel

Ensuite, on a dit qu'il fallait que le `TextView` interprète en temps réel le contenu de l'`EditText`. Pour cela, il suffit de faire en sorte que chaque modification de l'`EditText` provoque aussi une modification du `TextView` : c'est ce qu'on appelle un évènement. Comme nous l'avons déjà vu, pour gérer les évènements, nous allons utiliser un `Listener`. Dans ce cas précis, ce sera un objet de type `TextWatcher` qui fera l'affaire. On peut l'utiliser de cette manière :

Code : Java

```
editText.addTextChangedListener(new TextWatcher() {
    @Override
 /**
* s est la chaîne de caractère qui permet
*/
    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 débute 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 soit
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 à quelles touches presse l'utilisateur et réagir en fonction du type de touches. 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 :

Code : Java

```
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 le « 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 :

Code : Java

```
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 translater 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 translater 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 translatera 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 sa vitesse avec la fonction `public void setDuration (long durationMillis)`. Pour rajouter un interpolateur on peut utiliser la fonction `public void setInterpolator (Interpolator i)`, moi 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 :

Code : XML

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

Liens

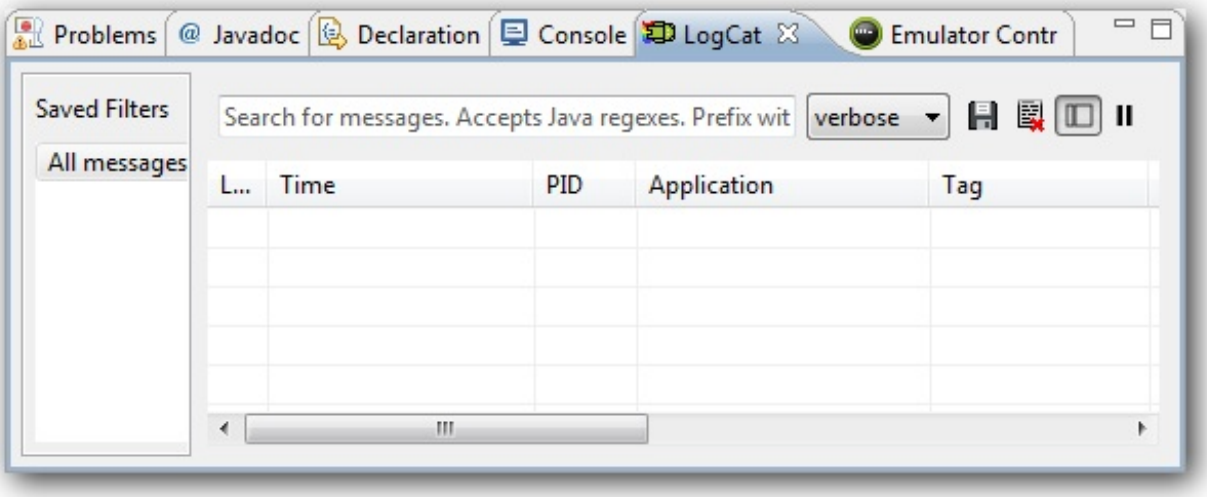
Plus d'informations :

- EditText
- Html et Html.ImageGetter
- TextView
- TextWatcher
- TranslateAnimation

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 le 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 cette partie :



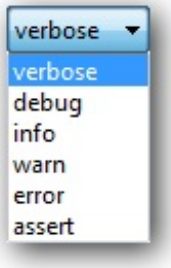
Première chose à faire, c'est de cliquer sur le troisième bouton en haut à droite :

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 second d'effacer toutes les entrées actuelles du journal afin d'obtenir un journal tout vierge et le dernier bouton permet de mettre en pause pour ne plus le voir 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 caractères informatifs.
- `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 :



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

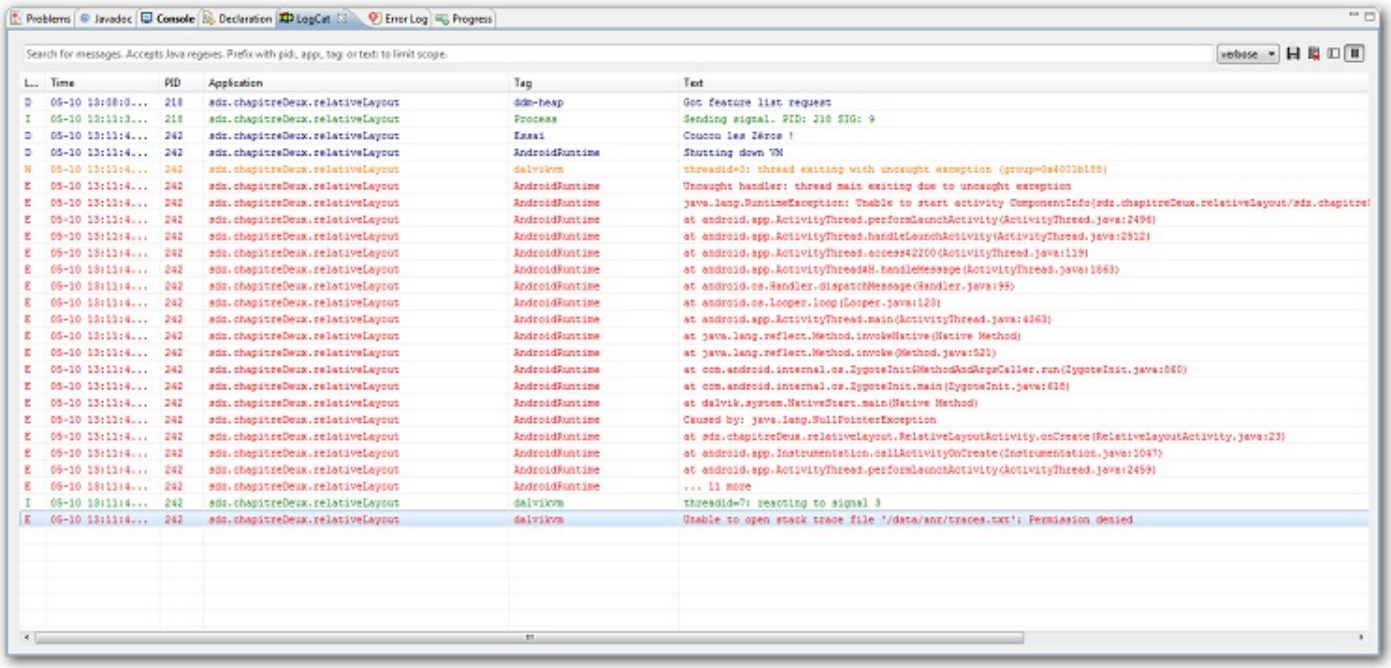
Sachez aussi que si votre programme lance une exception non catchée, c'est dans le Logcat que vous verrez ce qu'on appelle le « *stack trace* », c'est-à-dire les différents appels à des méthodes qui ont amené au lancement de l'exception.

Par exemple avec le code :

Code : Java

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

On obtient :



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 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 dans la seconde 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. Il contient :

Code : XML

```
<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 :

Code : XML

```
<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 le fond des boutons qui affichent des smileys en transparent.

Les chaînes de caractères

Sans surprise, j'utilise des ressources pour contenir mes strings :

Code : XML

```
<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.

En gros, cette classe ne possède qu'une grosse méthode qui permet d'ouvrir ou de fermer le menu :

Code : Java

```
/**
* 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`, alors je l'appelle avec la syntaxe `sdz.chapitreDeux.notepad.Slider` :

Code : XML

```
<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 :

Code : XML

```
<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"`.

Code : XML

```
<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 !

Code : XML

```
<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 rétrécie avec android:shrinkColumns et on ajoute un élément invisible qui prend le plus de place possible en largeur. Regardez-vous même :

Code : XML

```
<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 :

Code : Java

```
        /**
        * 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'oeil..
     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 :

Code : Java

```
  public class NotepadActivity extends Activity {
   /* Recupération 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 recupère 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 cache 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 ajouter 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 le
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().toString() + "</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 ouvrantes et fermantes 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 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 ouvrantes et fermantes 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 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 ouvrantes et fermantes 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 les 3
    caractères de la balise <b>)
          editable.insert(selectionEnd + 3, "</u>");
         }

        }
       });
         }
    }
```

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 de ce **9-Patch** :

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 :

Code : XML

```
<?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 ? 

Partie 3 : Annexes

L'architecture d'Android

Après quelques réflexions et quelques recherches, je me suis dit que c'était peut-être une bonne idée de présenter aux plus curieux l'architecture d'Android. Vous pouvez considérer ce chapitre comme facultatif s'il vous ennuie ou vous semble trop compliqué, vous serez tout de même capable de développer correctement sous Android mais un peu de culture technique ne peut pas vous faire de mal .

Concrètement, l'architecture d'Android, c'est ça :

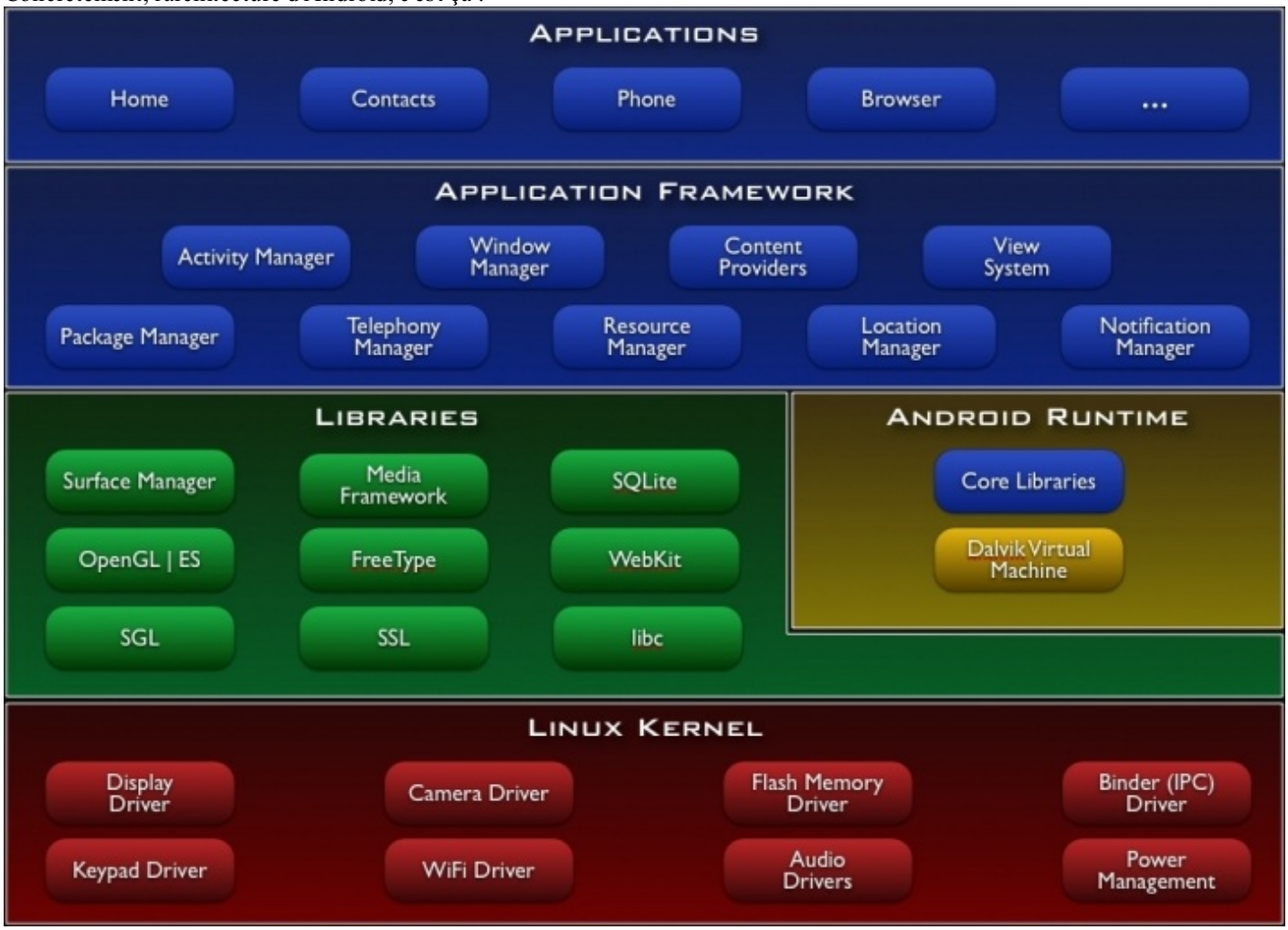


Schéma qui provient de la page suivante, provenant du site d'Android destiné aux développeurs.

En soi, ce schéma est incompréhensible et franchement pas glamour, mais il résume de manière concrète ce que je vais vous raconter. Ce que vous observez est une pile des composants qui constituent le système d'exploitation. Le sens de lecture se fait de bas en haut, puisque le composant de plus bas niveau est le noyau Linux et celui de plus haut niveau sont les applications.

Le noyau Linux

Je vous avais déjà dit que le système d'exploitation d'Android se basait sur Linux. Si on veut être plus précis, c'est le noyau (« **kernel** » en anglais) de Linux qui est utilisé. Le noyau est l'élément du système d'exploitation qui permet de faire le pont entre le matériel et le logiciel. Par exemple les pilotes Wifi permettent de contrôler la puce Wifi. Quand Android veut activer la puce Wifi, on peut imaginer qu'il utilise la fonction « allumerWifi() », et c'est au constructeur de spécifier le comportement de « allumerWifi() » pour sa puce. On aura donc une fonction unique pour toutes les puces, mais le contenu de la fonction sera unique pour chaque matériel.

La version du noyau utilisée avec Android est une version conçue spécialement pour l'environnement mobile, avec une gestion avancée de la batterie et une gestion particulière de la mémoire. C'est cette couche qui fait en sorte qu'Android soit compatible avec tant de supports différents.

Ça ne signifie pas qu'Android est une distribution de Linux, il a le même coeur mais c'est tout. Vous ne pourrez pas lancer d'applications destinées à GNU/Linux sans passer par de petites manipulations, mais si vous êtes bricoleur...

Si vous regardez attentivement le schéma, vous remarquerez que cette couche est la seule qui gère le matériel. Android en soi ne s'occupe pas de ce genre de détails. Je ne veux pas dire par là qu'il n'y a pas d'interactions entre Android et le matériel, juste que quand un constructeur veut ajouter un matériel qui n'est pas pris en compte par défaut par Android, il doit travailler sur le kernel et non sur les couches au-dessus, qui sont des couches spécifiques à Android.

Les bibliothèques pour Android

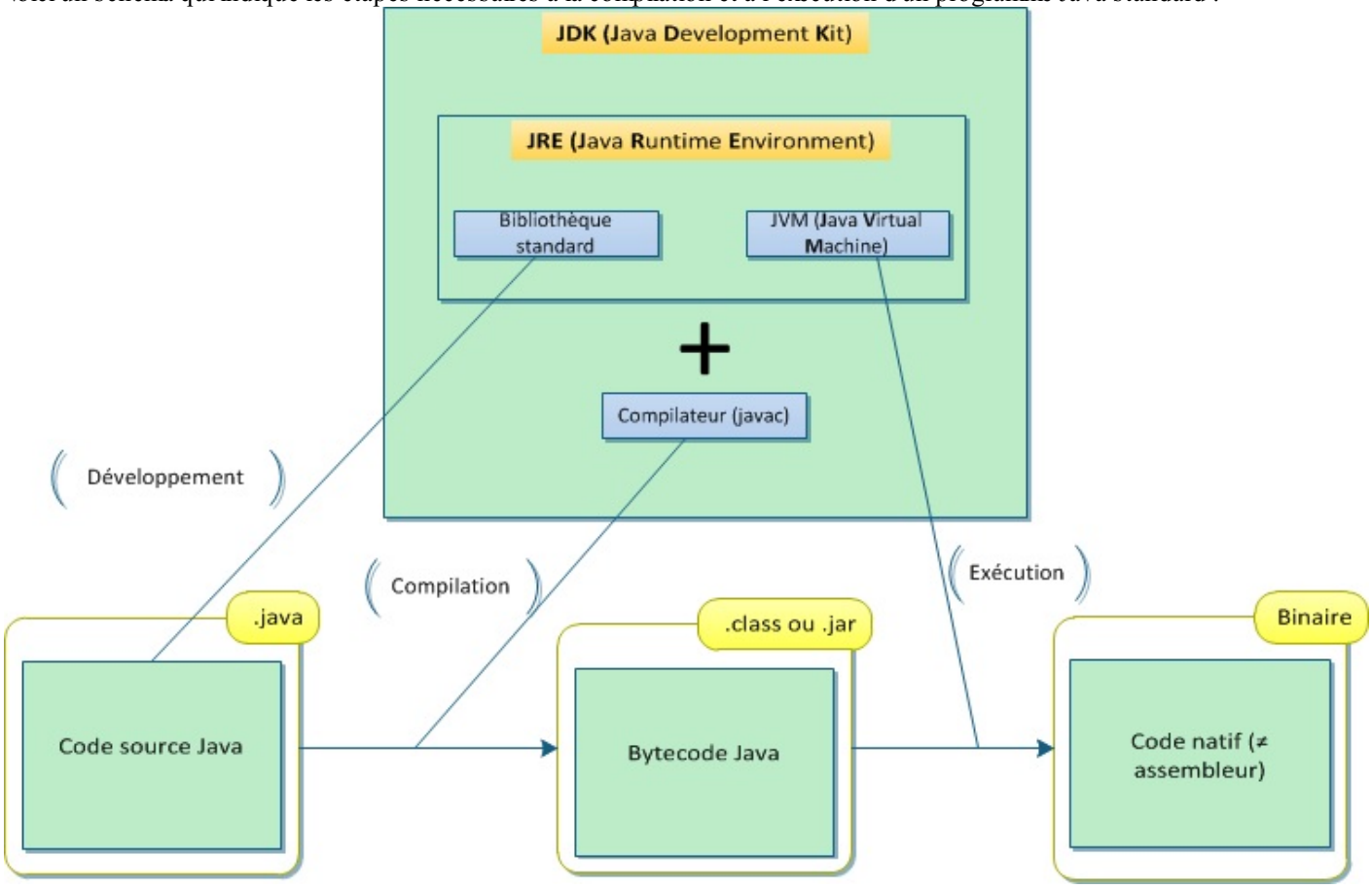
Ces bibliothèques proviennent de beaucoup de projets open-sources, écris en C/C++ pour la plupart, comme SQLite pour les bases de données, WebKit pour la navigation web ou encore OpenGL afin de produire des graphismes en 2D ou en 3D. Vous allez me dire qu'Android ne peut pas utiliser ces bibliothèques puisqu'elles ne sont pas en Java, cependant vous auriez tort puisqu'Android comprend très bien le C et le C++ !

Le moteur d'exécution Android

C'est cette couche qui fait qu'Android n'est pas qu'une simple « implémentation de Linux pour portables ». Elle contient certaines bibliothèques de base du Java accompagnées de bibliothèques spécifiques à Android et la machine virtuelle « **Dalvik** ».

Un moteur d'exécution (« **runtime system** » en anglais) est un programme qui permet l'exécution d'autres programmes. Vous savez peut-être que pour utiliser des applications développées en Java sur votre ordinateur vous avez besoin du JRE (« **Java RUNTIME Environment** »), et bien il s'agit du moteur d'exécution nécessaire pour lancer des applications écrites en Java.

Voici un schéma qui indique les étapes nécessaires à la compilation et à l'exécution d'un programme Java standard :



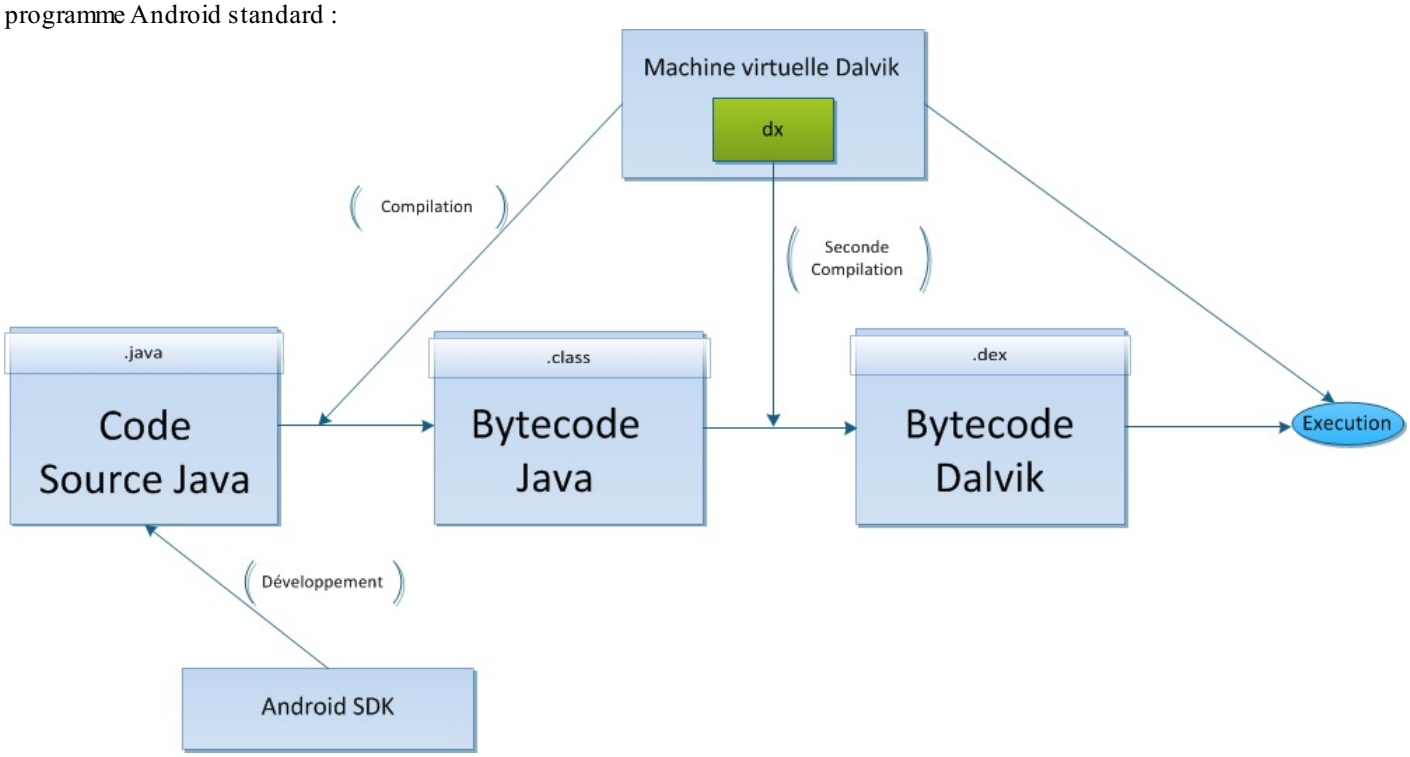
Votre code est une suite d'instructions que l'on trouve dans un fichier « **.java** » et ce fichier sera traduit en une autre suite d'instructions dans un autre langage que l'on appelle le « **bytecode** ». Ce code est contenu dans un fichier « **.class** ». Le bytecode est un langage spécial qu'une machine virtuelle Java peut comprendre et interpréter. Les différents fichiers « **.class** » sont ensuite regroupés dans un « **.jar** » et c'est ce fichier qui est exécutable. En ce qui concerne Android, la procédure est différente. En fait ce que vous appelez Java est certainement une variante particulière de Java qui s'appelle « **Java SE** ». Or, pour développer des applications pour Android, on n'utilise pas vraiment Java SE. Pour ceux qui savent ce qu'est « **Jave ME** », ce n'est pas non plus ce framework que l'on utilise (Java ME est une version spéciale de Java destinée au développement mobile, mais pas pour Android donc).

A noter que sur le schéma le JDK et le JRE sont réunis, mais il est possible de télécharger le JRE sans télécharger le JDK.

La version de Java qui permet le développement Android est une version réduite amputée de certaines fonctionnalités qui n'ont rien à faire dans un environnement mobile. Par exemple, la bibliothèque graphique Swing n'est pas supportée, on trouve à la place un système beaucoup plus adapté. Mais Android n'utilise pas une machine virtuelle Java ; une machine virtuelle toute

étudiée pour les systèmes embarqués a été développée, et elle s'appelle « **Dalvik** ». Cette machine virtuelle est optimisée pour mieux gérer les ressources physiques du système. Je citerai par exemple qu'elle permet de laisser moins d'empreinte mémoire (la quantité de mémoire allouée à une application pendant son exécution) et qu'elle puise moins dans la batterie qu'une machine virtuelle Java.

La plus grosse caractéristique de Dalvik est qu'elle permet d'instancier (terme technique qui signifie « créer une occurrence de ». Par exemple quand vous créez un objet en java, on instancie une classe puisqu'on crée une occurrence de cette classe) un nombre très important d'occurrences de lui-même : chaque programme a sa propre occurrence de Dalvik et elles peuvent vivre sans se perturber les unes les autres. Voici un schéma qui indique les étapes nécessaires à la compilation et à l'exécution d'un programme Android standard :



On voit bien que le code Java est ensuite converti en bytecode Java comme auparavant. Mais souvenez-vous, je vous ai dit que le bytecode Java ne pouvait être lu que par une machine virtuelle Java, et je vous ai aussi dit que Dalvik n'était PAS une machine virtuelle Java. Il faut donc procéder à une autre conversion à l'aide d'un programme qui s'appelle « **dx** ». Ce programme s'occupera de traduire les applications de bytecode Java en bytecode Dalvik, qui lui est compréhensible par la machine virtuelle.

La puissante machine virtuelle Dalvik est destiné uniquement à Android, mais il est possible de développer une machine virtuelle similaire et qui ne fonctionne pas sous Android. Par exemple, le **Nokia N9** pourra exécuter des applications Android sans utiliser ce système.

Les frameworks pour les applications

Il s'agit d'un framework qui...

Un quoi??

Un framework peut se traduire littéralement par un cadre de travail. Il s'agit d'un ensemble de composants qui définissent les fondations ainsi que les grandes lignes directrices de l'organisation d'un code, en d'autres termes on peut parler de son architecture ou de son squelette. Un framework prodigue aussi quelques fonctionnalités de base (accès à la base de données par exemple). Cet outil fournit ainsi une démarcation radicale entre plusieurs aspects d'un programme et permet de mieux diviser les tâches (toi tu fais l'interface graphique, toi l'interaction avec le réseau, etc).

En fait ce sont ces frameworks là qui vous sont disponibles quand vous programmez en Java, et qui s'occupent d'interagir avec la bibliothèque Android. Vous pouvez vous contenter d'appeler des fonctions déjà faites à l'avance par d'autres et les regarder s'exécuter tranquillement.

Les applications

Il s'agit tout simplement d'un ensemble d'applications que l'on peut trouver sur Android, par exemple les fonctionnalités de base inclues un client pour recevoir/envoyer des emails, un programme pour envoyer/recevoir des SMS, un calendrier, un répertoire, etc. C'est ici que vous intervenez, cette stèle est celle de vos futures applications développées en Java. Vous l'aurez compris, vous accédez aux mêmes ressources que les applications fournies par défaut, vous avez donc autant de possibilités qu'elles, vous avez même la possibilité de les remplacer. C'est aussi ça la force d'Android.

Voilà, c'est terminé ! Non je plaisante, on est encore très loin de la fin, et de toute façon on a des projets de livres, puis

d'adaptations cinématographiques, puis d'adaptation en jeux vidéo !

Remerciements

Pour leur(s) critique(s) perspicace(s), je tiens à remercier:

- les bétas testeurs.
- Merci aux validateurs et **Hilaia**.
- Remerciements spéciaux à **John-John**, **AndroiWiiid** et **AnnaStretter** pour leurs conseils et critiques pertinentes.

Pour l'icône laide qui me fait perdre trois lecteurs par jour, merci à Bérenger Pelou.