

Haute Ecole de la Province de Liège – Département Sciences et techniques

Bachelier en Informatique

(Informatique de gestion & Informatique et systèmes opt. réseaux-télécommunication)

Java (V)

Une introduction à Android

UE: Programmation réseaux, web et mobiles / AA : Technologie de l'e-commerce et mobiles

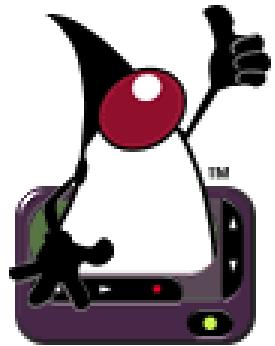
UE: Programmation Réseaux, Web et Mobiles / AA : Compléments de programmation
réseaux et mobiles

Claude VILVENS

claude.vilvens@hepl.be

<http://haute-ecole.provincedeliege.be>

-- I329 --
(2021-2022)



Sommaire



XXVII. Un autre Java mais embarqué : Android

1. Un framework pseudo-Java pour mobiles	3
2. Le modèle en couches d'Android	
2.1 Le schéma bloc	5
2.2 La couche Linux Kernel	5
2.3 La couche librairies	5
2.4 La couche Java ("Android Runtime")	6
2.5 La couche Application Framework	7
2.6 La couche annexe : Google Play	7
2.7 Le point de vue du développeur	8
3. La machine virtuelle Dalvik	
3.1 Une machine orientée registres	8
3.2 Un bytecode particulier	9
3.3 Plusieurs machines virtuelles	11
4. Le fichier de distribution apk	12
5. Les composants applicatifs Android	
5.1 Composants et threads	12
5.2 La communication par intents	13
5.3 Activités et processus	15
6. Le cycle de vie d'une activité Android	15
7. Une application Android de base	17
8. Le SDK Android et son installation	18
9. Le développement manuel en ligne de commande	
9.1 Les outils et la création du projet de base	23
9.2 La description des ressources	28
9.3 Le manifeste de l'application	30
9.4 La génération du fichier R	31
9.5 Le développement et le déploiement	34
9.6 Le processus de déploiement avec ant	36
9.7 L'exécution de l'application	40
10. Le plugin Android pour NetBeans	42
11. Crédit et développement d'un projet Netbeans	44
12. Installation et exécution d'une application sur un mobile	51
13. L'Android Debug Bridge (ADB)	56
14. Les interfaces et les événements graphiques	
14.1 View et ViewGroup	60
14.2 Déclaration et propriétés des composants d'un GUI	61
14.3 Les composants spécifiques	63
14.4 Les gestionnaires de mise en page	64

14.5 La gestion des événements graphiques	65
15. L'outil DroidDraw	69
16. Les menus	
16.1 Le menu classique	73
16.2 La barre des actions	77
17. Une application GUI simple	80
18. Les listes et les adapters	84
19. Les communications réseaux	90
20. Threads asynchrones et GUIs	97
21. Les intents	
21.1 Application, activités et intents	101
21.2 Un message asynchrone	102
21.3 Démarrage explicite	102
21.4 Communication entre deux activités : mode d'emploi	104
21.5 Démarrage implicite	105
21.6 Dialogue entre deux activités par intent	106
22. Les services	
22.1 Le cycle de vie d'un service Android	111
22.2 Un service local musical	111
22.3 Le dialogue entre une activité et un service distant	117
23. Les fragments	
23.1 Le principe des fragments	122
23.2 Une librairie de support des fragments	124
23.3 Le développement d'une application à fragments statiques	125
23.4 Le principe des fragments dynamiques	129
23.5 Fragments, ActionBar et communication entre fragments	131
24. SQLite	
24.1 Un moteur interopérable	139
24.2 Une application contexte avec ListActivity	141
24.3 Les bases de SQLite sous Android	147
24.4 Une couche DAO : SQLiteDatabase	149
24.5 Une classe DAO	150
24.6 Retour à ListActivity	153
24.7 Effacer les données	156
25. Les graphiques statistiques	
25.1 Un élément indispensable du data mining	157
25.2 Fonctionnement de base de la librairie AChartEngine	157
25.3 Un exemple de graphique circulaire	160
25.4 Un exemple de graphique linéaire	164
26. L'internationalisation	167

Ouvrages consultés

XXVII. Un autre Java embarqué : Android



La pensée ne commence qu'avec le doute.

(R. Martin du Gard, Correspondance avec A.Gide)

J2ME vise à fournir un framework configurable et adaptable à des mobiles extrêmement divers. Voici le point de vue diamétralement opposé : une technologie propriétaire, dédiée à une architecture bien précise avec comme langage de développement un Java non standard ... Une petite introduction à Android ?

1. Un framework pseudo-Java pour mobiles

Android est un framework combinant un système d'exploitation pour mobile et un SDK permettant le développement d'application pour tout mobile géré par cet OS.

En fait, il utilise une version modifiée du kernel Linux - mais il ne s'agit pas ici d'une distribution de Linux puisqu'il manque, par exemple, le système X Window et le jeu complet des librairies standard GNU. Android a été développé dès 2007 par Android Inc. (bien logiquement) puis racheté par Google pour finalement passer sous le contrôle de la **OHA** (**Open Handset Alliance**), un consortium d'entreprises diverses intéressées par le marché des mobiles comme Bouygues, Alcatel, Dell, Acer, Samsung, HTC, Sony Ericsson, Toshiba, Intel, Vodafone, etc (voir http://www.openhandsetalliance.com/oha_members.html pour une liste complète).

Le personnage nommé Bugdroid est le petit robot vert utilisé par Google pour présenter Android - il est resté le symbole d'Android, à l'instar de Duke pour le Java standard :



La configuration matérielle prévue pour Android correspond à ce que l'on trouve sur un HTC Dream : essentiellement un écran tactile, Wifi et Bluetooth, un trackball, un clavier coulissant Qwerty et une carte SD de 1Gb. Du point de vue logiciel, **Android utilise un Java propriétaire**, non compatible avec J2ME et J2SE, et des librairies natives, construites à nouveau avec des **librairies C non standards** (la librairie Bionic).

Les différentes versions d'Android sont désignées de trois manières :

- ◆ par le numéro de version de la plate-forme : 2.3, 3.2, 4.3, 5.0
- ◆ par un nom de code suivant l'ordre alphabétique : Cupcakes, Donuts, Ice Cream, Sandwiches, Lollipop, etc ; un même nom peut être conservé pour une nouvelle version, auquel cas on complète le nom de la version précédente par "MR1", "MR2", etc;
- ◆ par un niveau d'APIs (il s'agit donc en fait de la version du SDK) : 8, 15, 18, 20, ...



Les références "pseudo-gastronomiques" ont été abandonnées dans la dernière version de 2019: on ne parle plus que d'Android 10.

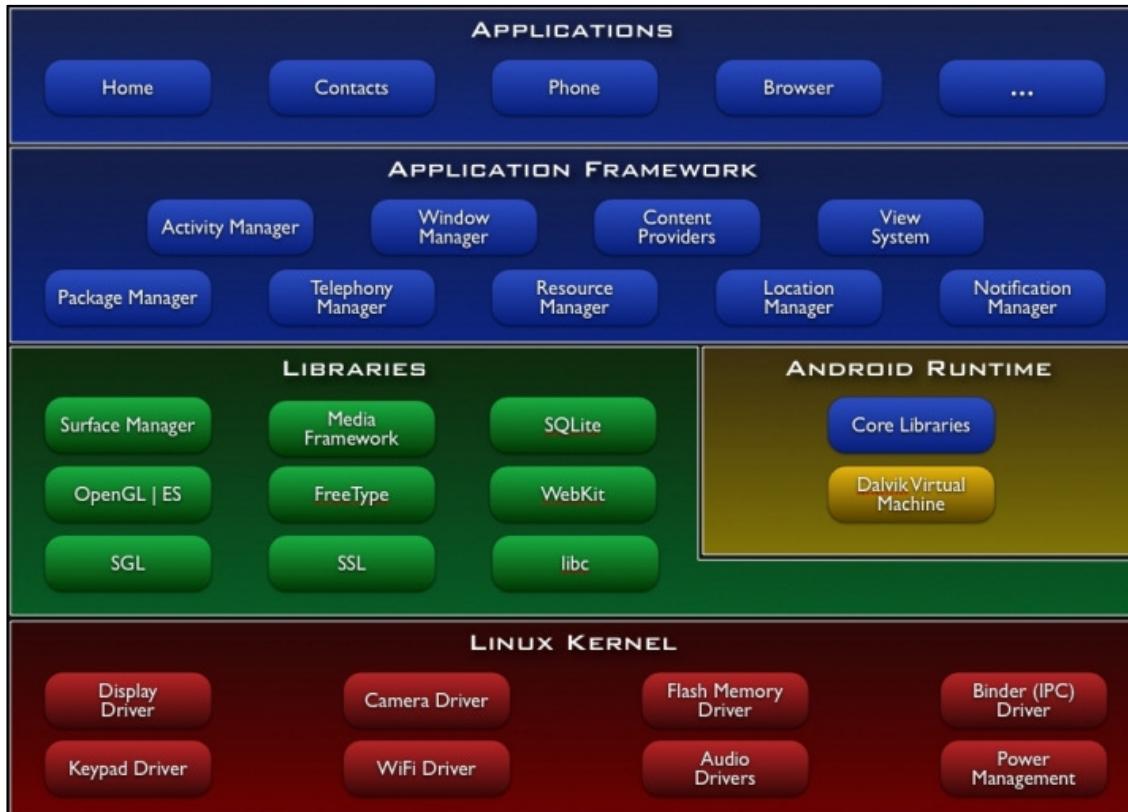
Le tableau suivant propose la synthèse des versions existantes en 2020 (source : <http://developer.android.com/guide/topics/manifest/uses-sdk-element.html>)

Platform Version	API Level	VERSION_CODE	Notes
Android 10.0	29	Q	Platform Highlights
Android 9	28	P	Platform Highlights
Android 8.1	27	O_MR1	Platform Highlights
Android 8.0	26	O	Platform Highlights
Android 7.1.1 Android 7.1	25	N_MR1	Platform Highlights
Android 7.0	24	N	Platform Highlights
Android 6.0	23	M	Platform Highlights
Android 5.1	22	LOLLIPOP_MR1	Platform Highlights
Android 5.0	21	LOLLIPOP	
Android 4.4W	20	KITKAT_WATCH	KitKat for Wearables Only
Android 4.4	19	KITKAT	Platform Highlights
Android 4.3	18	JELLY_BEAN_MR2	Platform Highlights
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1	Platform Highlights
Android 4.1, 4.1.1	16	JELLY_BEAN	Platform Highlights
Android 4.0.3, 4.0.4	15	ICE_CREAM SANDWICH_MR1	Platform Highlights
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM SANDWICH	
Android 3.2	13	HONEYCOMB_MR2	
Android 3.1.x	12	HONEYCOMB_MR1	Platform Highlights
Android 3.0.x	11	HONEYCOMB	Platform Highlights
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1	Platform Highlights

2. Le modèle en couches d'Android

2.1 Le schéma bloc

La structure du framework est traditionnellement représentée sur les sites Web (<http://developer.android.com>) de la manière suivante :



Donnons immédiatement quelques précisions sur les différentes couches.

2.2 La couche Linux Kernel

On l'a dit, Android utilise *une version modifiée du kernel Linux* (un 2.6) qui prend en charge les drivers, les accès aux ressources, l'alimentation électrique. Cette couche représente donc l'interface d'accès au hardware pour les couches supérieures. Cependant, si certains éléments du kernel Linux traditionnel ont été enlevés, d'autres ont été ajoutés pour permettre son utilisation dans le monde particulier des mobiles : on parle de "*patches*". Citons :

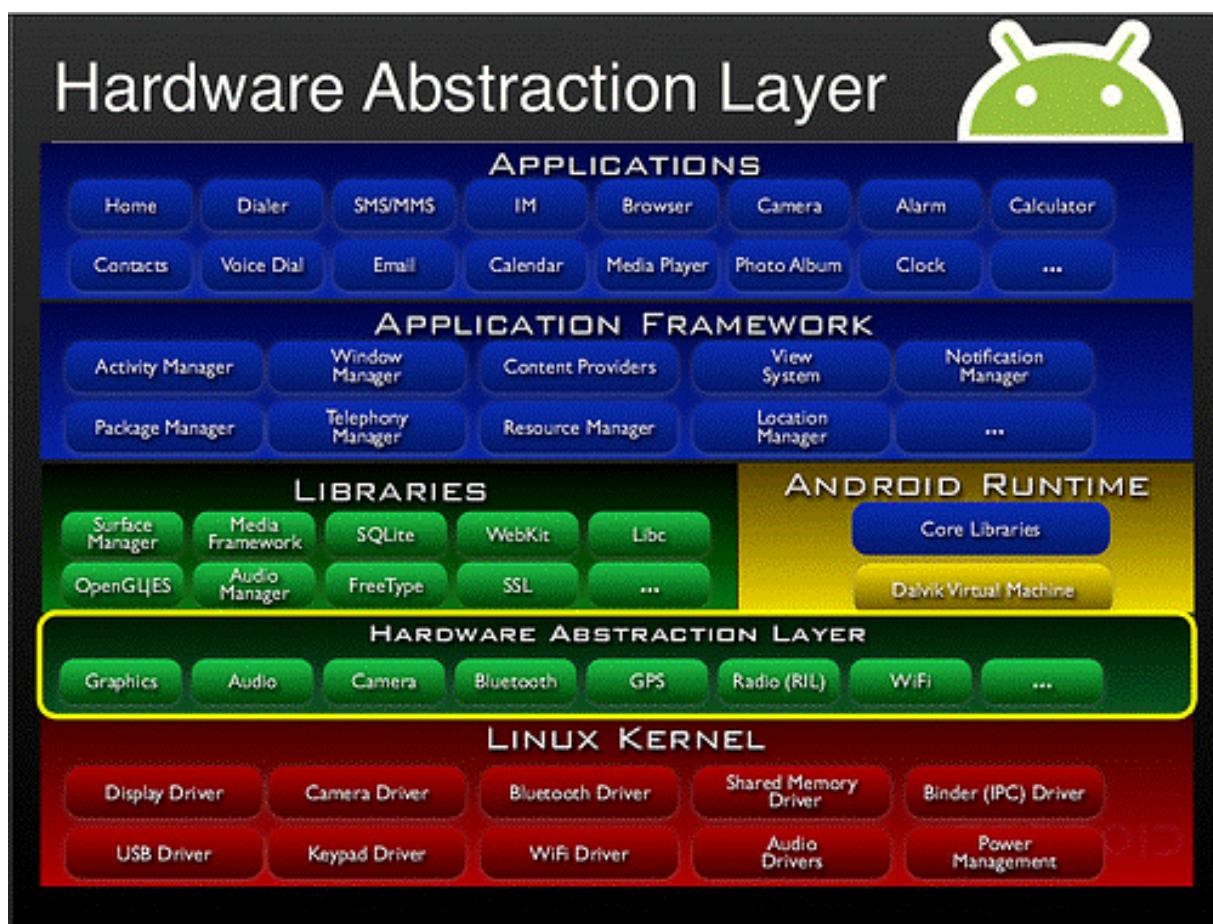
- ◆ Alarm : pour le réveil de l'appareil quand il est en veille;
- ◆ Ashmem : gestion du partage mémoire, nécessaire sur un mobile où l'espace est réduit;
- ◆ Binder : il s'agit d'un module prenant en charge de manière sécurisée les IPC permettant une communication entre les différentes applications qui tournent dans des processus différents; l'idée est en fait de ne laisser les développeurs d'applications créer leurs propres IPC, ce qui ressentirait comme potentiellement dangereux.

2.3 La couche librairies

Il s'agit d'un ensemble de librairies C/C++ qui seront utilisables depuis la couche supérieure (l'application framework). Citons :

- ◆ **Bionic LibC** : une espèce de *glibc revue à la baisse*, à nouveau pour tenir compte de l'environnement réduit d'un mobile;
- ◆ Webkit : une librairie de fonctions permettant aux logiciels d'afficher les éléments d'une page web - on parle encore de "*moteur de rendu*"; c'est un dérivé du moteur KHTML du projet KDE (ensemble de technologies dédiées à un environnement de bureau et de développement d'application dans un contexte multiplateformes);
- ◆ **SQLLite** : un SGBD "léger" particulièrement adapté au monde des mobiles;
- ◆ Surface Flinger : pour les rendus graphiques en 2d et 3D.

Un cas particulier est celui des "**Hardware Abstraction Libraries**" puisqu'elle fournit simplement les interfaces que doivent implémenter les différents kernels : les librairies utiliseront cet interface pour accéder aux ressources matérielles. Le bloc diagramme d'Android sera donc rectifié de la manière suivante :



2.4 La couche Java ("Android Runtime")

Du point de vue développement rapide d'applications, il s'agit bien sûr de la couche la plus intéressante. On peut remarquer :

- ◆ les **Cores Librairies** : ce sont les librairies Java de base, de type J2SE 1.5 mais propriétaires et différentes (par exemple, pas de javax.swing);
- ◆ la **machine virtuelle Dalvik** : nous la présenterons plus en détail ci-dessous.

A remarquer encore que si une application Java nécessite du code en C/C++, elle pourra le faire par JNI sur du code développé avec le Native Development Kit (**NDK**).

2.5 La couche Application Framework

Il s'agit ici du *toolkit* que toutes les applications utiliseront, quelles qu'elles soient. En fait, ce sont des services qui tournent en arrière-plan et qui sont essentiels au fonctionnement du mobile. Citons :

1) les Core Platform Services

Il s'agit évidemment des services fondamentaux pour le fonctionnement même des applications sur le mobile :

- ◆ Activity Manager : gestion du cycle de vie des applications et de la pile de navigation (pour revenir à l'application précédente quand une application se termine);
- ◆ Package Manager : chargement des fichiers .apk (voir ci-dessous);
- ◆ Window Manager : au-dessus du Surface Flinger, il gère les recouvrements des fenêtres;
- ◆ Resource Manager : bon, c'est clair;
- ◆ Content Provider : gestion de partage de données entre applications (provenant par exemple d'une base de données);
- ◆ View System : tous les composants graphiques classiques, comme les listes, grilles, zones d'édition, etc

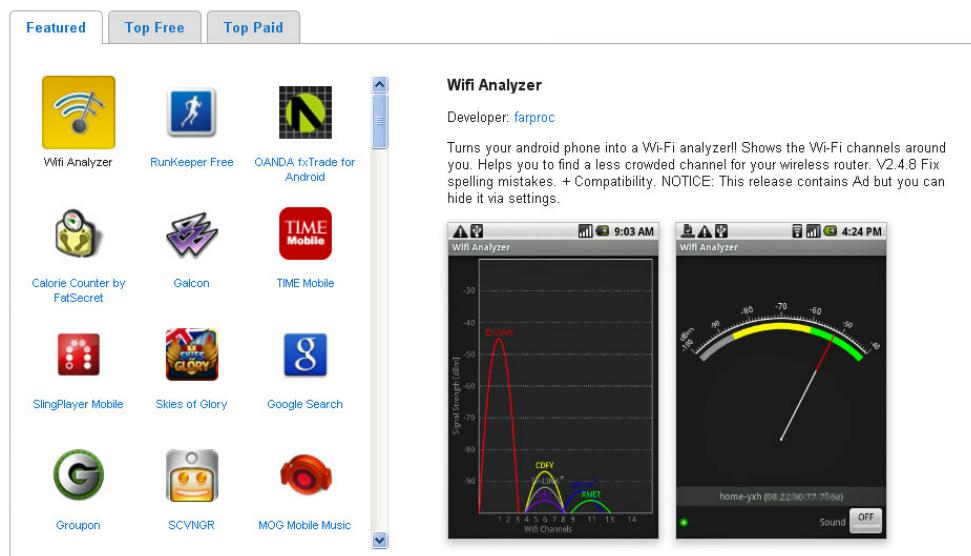
2) les Hardware Services

Il s'agit des APIs vers le matériel. Les citer suffit à les identifier :

- ◆ Telephony Service
- ◆ Location Service (pour le GPS)
- ◆ Bluetooth Service
- ◆ Wifi Service
- ◆ USB Service

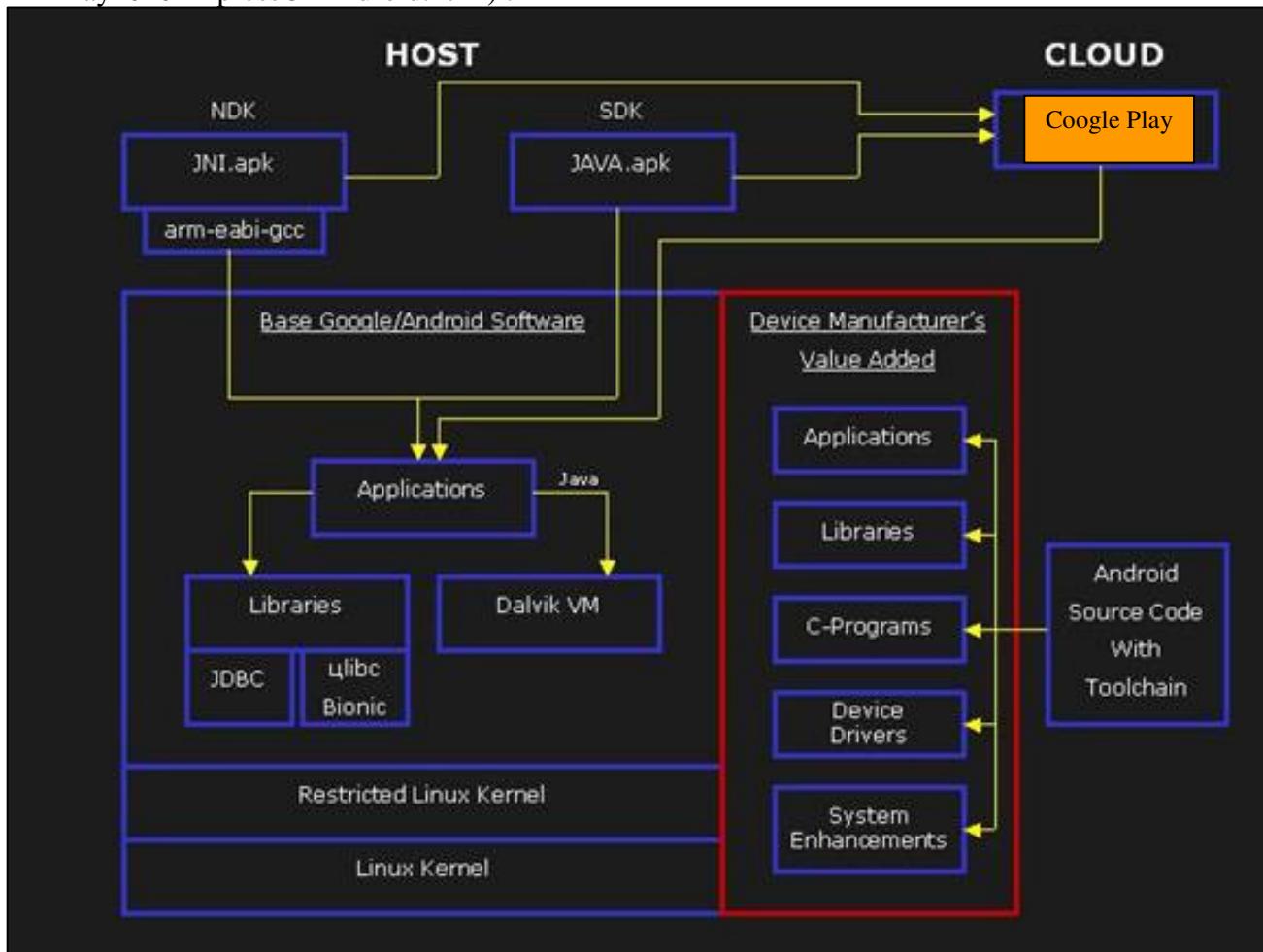
2.6 La couche annexe : Google Play

Il existe un **Google Play Store** (ex **Android Market**) (<http://www.android.com/market>) sur lequel on peut se fournir des applications gratuites ou payantes (avec une contribution de 30% laissée à Google) proposées par les développeurs Android :



2.7 Le point de vue du développeur

En résumé, on peut donc relire l'architecture d'Android selon le point de vue du développeur de la manière suivante (<http://www2.empress.com/whatsnew/techNews/May2010EmpressOnAndroid.html>) :



3. La machine virtuelle Dalvik

3.1 Une machine orientée registres

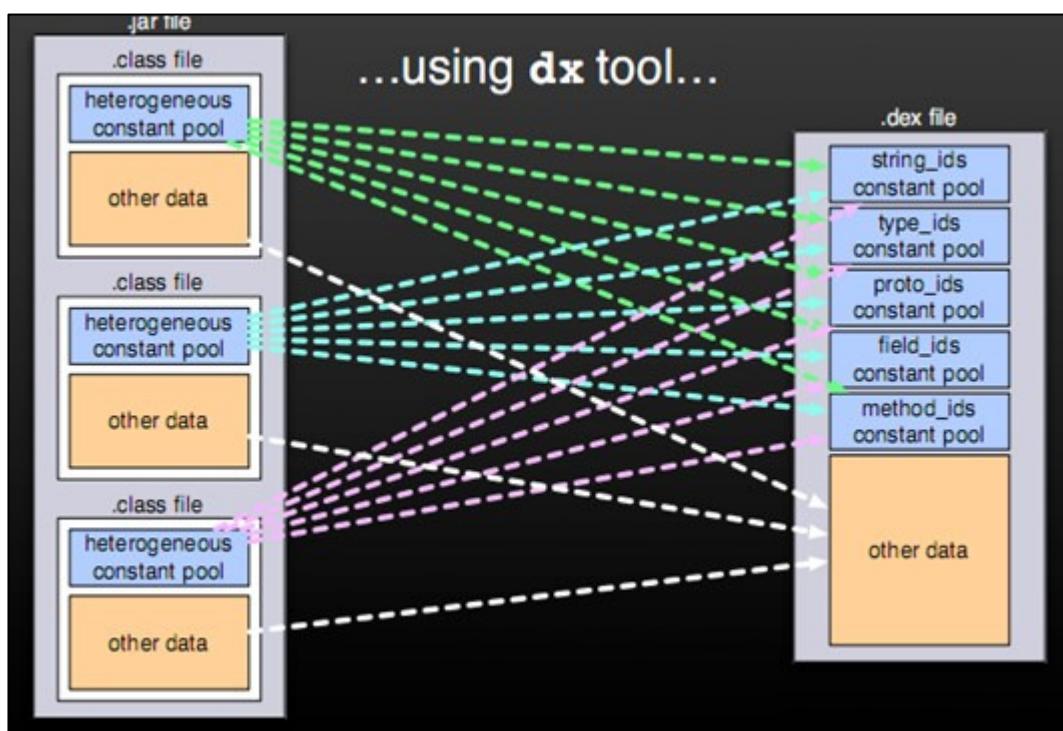
La machine virtuelle d'Android est tout à fait particulière. En effet, cette JVM nommée Dalvik (d'après le village des ancêtres de son inventeur, Dan Bornstein ...) possède une **architecture "orientée registres"** [*register-based architecture*], à l'opposé des JVM classiques à **architecture "orientée pile"** [*"stack machine"*]. La différence entre les deux se marque particulièrement pour les instructions de haut niveau qui manipulent des données et qu'il faut convertir en langage machine. Ainsi, sur une machine à pile, le mécanisme automatisé de la pile permet de faire l'économie des adresses dans les instructions machines mais réclame un nombre plus important d'instructions bas niveau pour manipuler les données. Par contre, sur une machine à registres, les instructions bas niveau contiennent directement les adresses des registres source et destination mais, par le fait même, sont plus longues. Les spécifications techniques des formats d'instructions se trouvent sur <http://www.dalvikvm.com>.

3.2 Un bytecode particulier

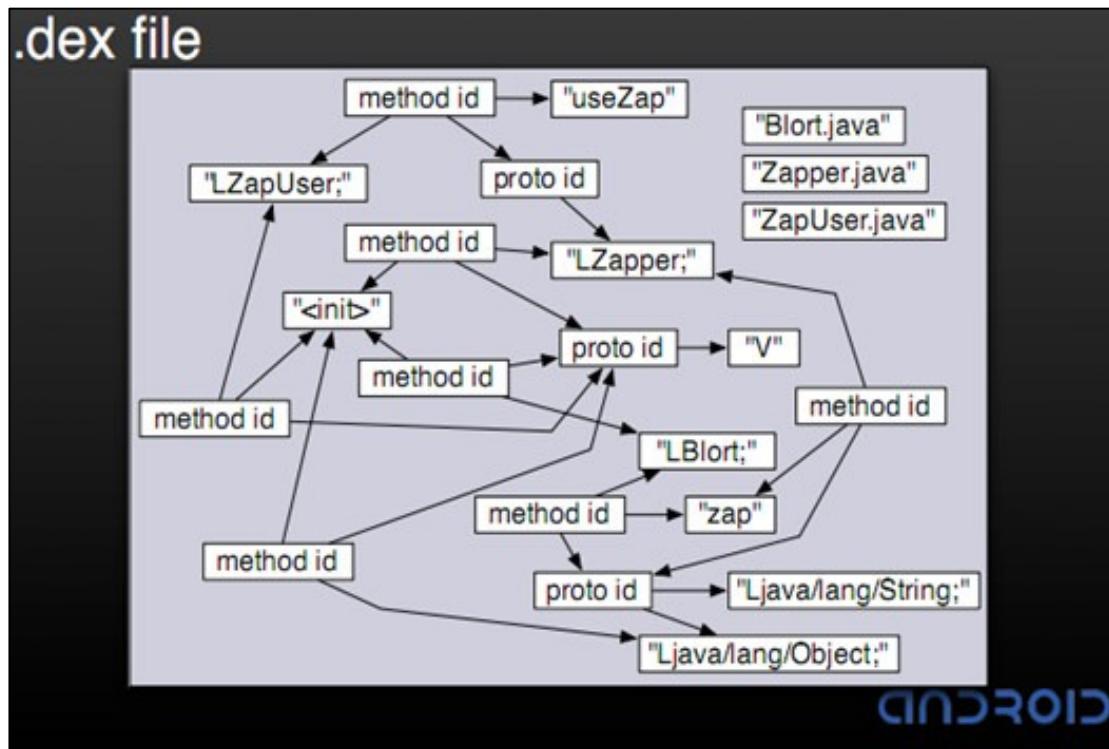
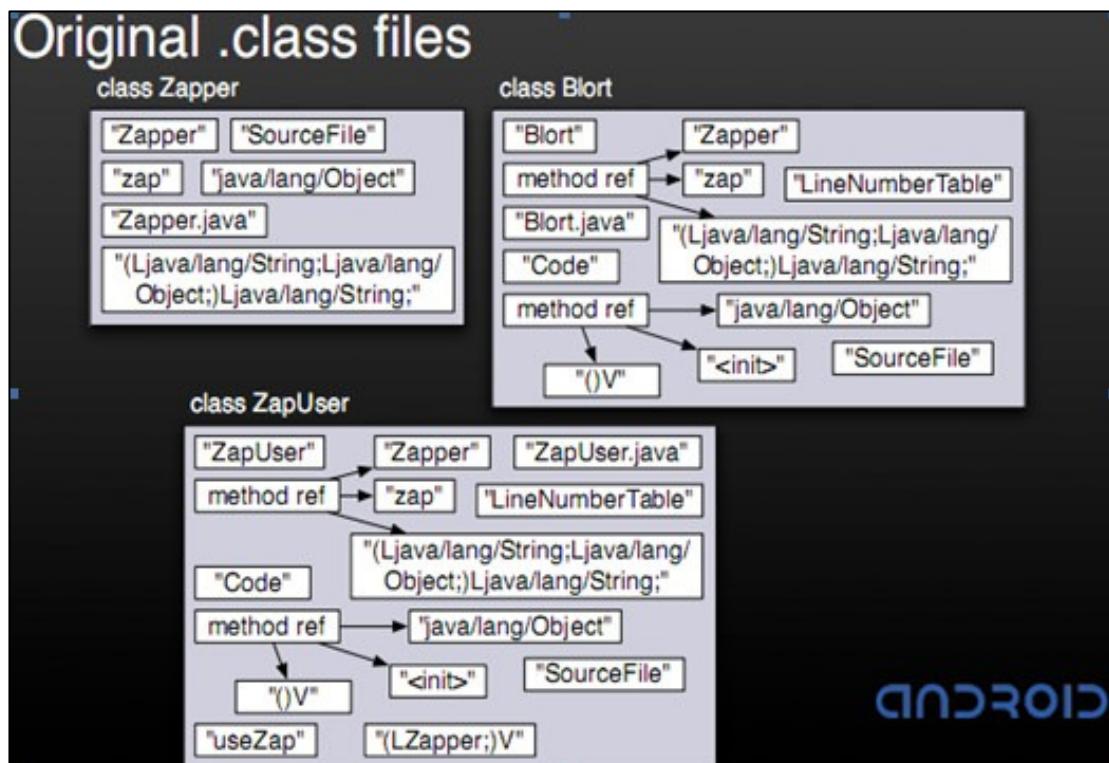
De plus, la JVM Dalvik ne consomme pas du bytecode classique, rassemblés en fichiers **.class**, mais exécute des applications converties en un fichier **.dex** (**Dalvik EXecutable**). Ce format dex est plus particulièrement adapté aux mobiles dotés d'un processeur de faible puissance et/ou d'une mémoire centrale limitée. Par exemple, les références multiples sont restructurées en des références uniques à un pool, supprimant ainsi les redondances et allégeant donc l'ensemble produit. Le bytecode qui sera fabriqué par le compilateur est donc converti en un nouveau set d'instructions Dalvik (au moyen d'un outil appelé **dx**) : on parle encore d'"**exécutable Dalvik**". Le schéma suivant (issu de <http://www-igm.univ-mlv.fr>) illustre la structure d'un fichier dex (à comparer avec l'adressage indexé de l'ASM et l'héritage virtuel du C++ ;-!) :



tandis que le suivant évoque le processus de transformation :



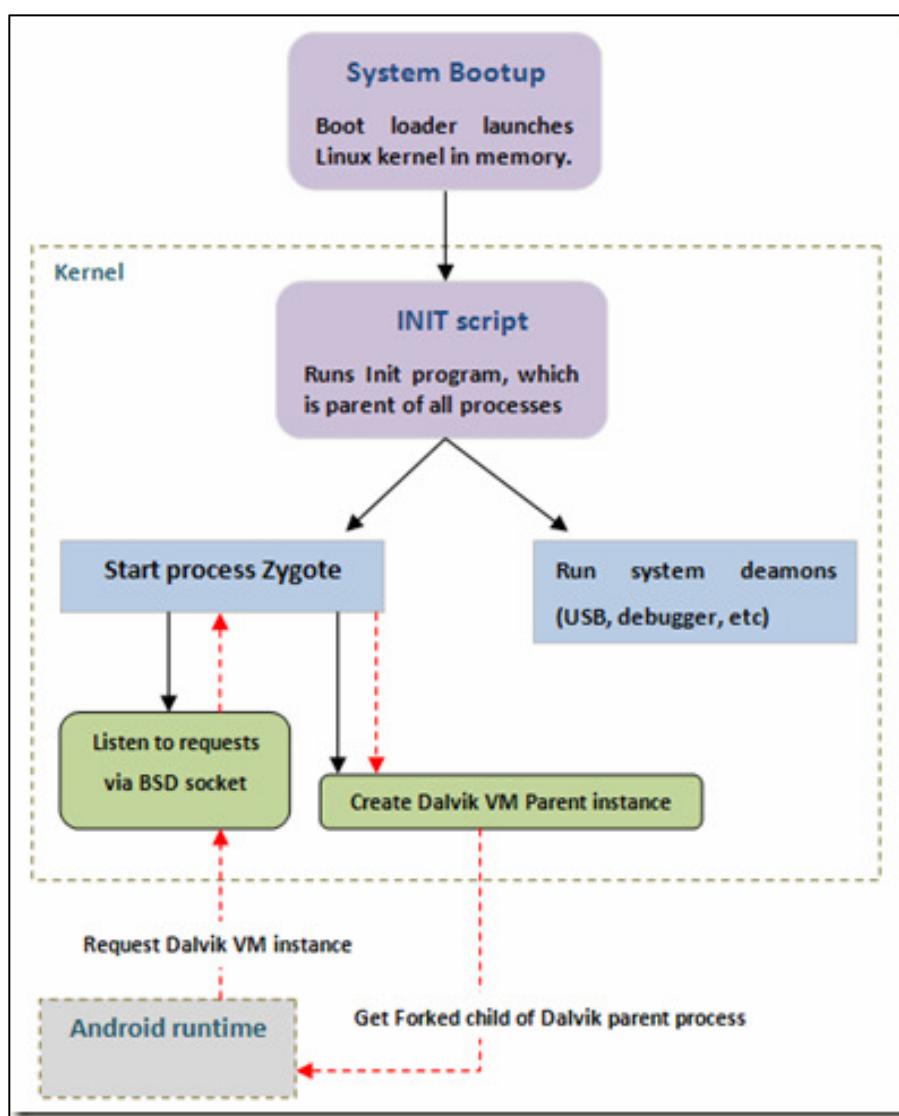
Un tel exécutable peut encore être modifié lors de son installation sur le mobile visé, ceci dans un but d'optimisation (modification du byte order, éviction de classes non utilisées, etc). Un exemple schématique permet de montrer la différence entre un fichier .jar contenant des fichiers .class et un fichier .dex (joyeusement pillé chez <http://www.ophonesdn.com/article/show/15>) :



3.3 Plusieurs machines virtuelles

Il faut savoir qu'une instance de la machine virtuelle Dalvik est créée pour chaque processus différent tournant sur l'appareil. Ce système a été choisi afin d'éviter une erreur générale dans le cas du plantage d'une des machines virtuelles. Mais par ce fait, il importait d'avoir une machine virtuelle beaucoup moins imposante qu'une JVM standard : le fichier dex répond à ce besoin.

Plus précisément, une fois le boot du système terminé, le système lance le programme INIT, comme sur un O.S. Linux classique, qui lui-même démarre les différents processus nécessaires au bon fonctionnement de l'O.S., dont le processus "Zygote", qui se met en attente de requêtes après avoir lancé la machine virtuelle "principale". *Chaque fois que le système créera une nouvelle application, il enverra une requête vers le processus Zygote* demandant la création d'un sous-processus de la JVM principale : c'est ce sous-processus faisant fonctionner une nouvelle instance de JVM qui sera attribuée à l'application.



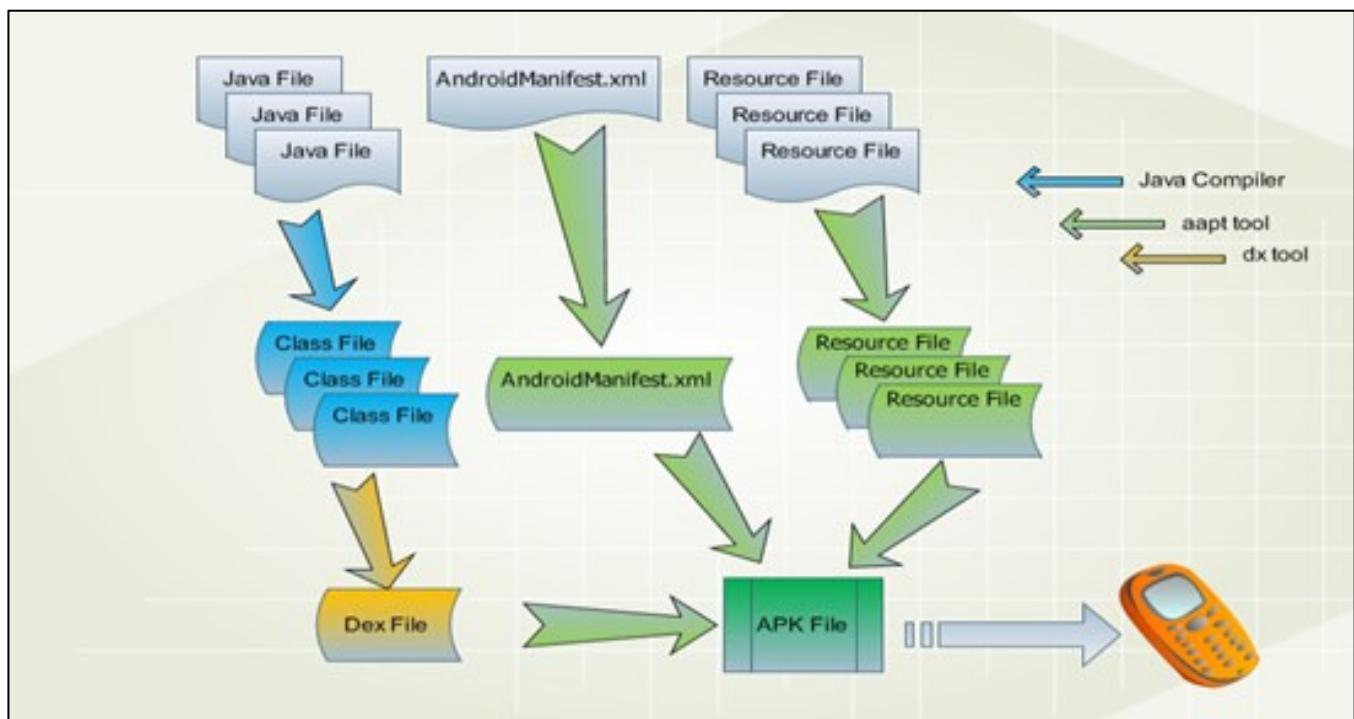
Une nouvelle machine virtuelle **ART** (Android Run Time) existe depuis 4.4 KitKat : elle ne fonctionne plus en Just-In-Time, mais en **Ahead-Of-Time** ("avant le temps"). A la différence de Dalvik, le code fonctionnant avec la machine virtuelle est traduit une seule fois lors de l'installation des applications. ART utilise un outil **dex2oat** pour réaliser cette compilation. L'allocation mémoire est aussi optimisée (nouveau garbage collector).

4. Le fichier de distribution apk

L'empaquetage final d'une application Android est réalisée dans un fichier **.apk** (Android PacKage), qui est une variante du format des fichiers **.jar** : on peut donc l'ouvrir avec les outils classiques (7-Zip, WinZip, WinRar, etc). Le type MIME est **application/vnd.android.package-archive**. Bien logiquement, ce fichier APK contient, outre l'habituel répertoire META-INF, les éléments suivants :

- ◆ le fichier **.dex**;
- ◆ un fichier XML décrivant l'application (**AndroidManifest.xml** - nous y reviendrons);
- ◆ les fichiers de ressources **.res** et **.arsc** : en fait, **res** est le répertoire qui contient les ressources proprement dites sous forme de **fichiers XML** tandis que le fichier **.arsc** est en fait le **code compilé de la classe R.java** décrivant les ressources (nous y reviendrons aussi).

Très schématiquement, les choses se passent donc ainsi :



5. Les composants applicatifs Android

5.1 Composants et threads

On l'a dit, une application Android est écrite en Java, mais il s'agit donc d'un Java non standard. Ainsi, par exemple, une telle application n'a pas de fonction `main()` qui servirait d'unique point d'entrée. En fait, une application [task] peut utiliser ses propres composants ou se servir des composants d'autres applications (si celles-ci le permettent) : elle démarre simplement la partie de code dont elle a besoin.



Mais qu'entend-on alors par "application" ? Le fichier apk dont il a déjà été question sera placé sur le mobile, constitue de facto ce qu'Android appelle une application. Celle-ci fonctionne au sein d'un processus Linux, avec son ID unique et, on l'a dit, sa propre machine

virtuelle, ce qui assure l'isolation du code. Elle possède seule (du moins à priori – on peut modifier cela) les permissions pour en utiliser les composantes et ressources.

Une application se présente donc comme un groupe d'un certain nombre de composants susceptibles d'être instanciés et exécutés : en fait, ils sont "empilés" selon les appels, le composant du sommet de la pile étant celui qui est actif. Par défaut, tout se passe dans un seul thread, ce qui implique que toute activité doit s'abstenir d'effectuer des tâches bloquantes. Si cela doit néanmoins être le cas, à sa charge de lancer un thread secondaire (instance de la classe Thread) pour gérer cette tâche; Android fournit des classes utilitaires pour permettre le dialogue entre ces threads au moyen de messages (classes Handler, Looper, HandlerThread).

Concernant les composants proprement dits, il peut s'agir

- ◆ d'une activité [activity] : objet instance de la classe **Activity** dont on peut lancer l'exécution pour réaliser une tâche quelconque utilisant un GUI (il possède une fenêtre par défaut); c'est l'association de ces activités qui permet de définir un interface graphique complet pour l'utilisateur de l'application;
- ◆ d'un service : objet instance de la classe **Service** dont on peut lancer l'exécution prolongée en tâche de fond (il fonctionne même si il n'est pas couplé à une activité) et qui ne possède donc pas d'interface graphique (exemple typique : une musique de fond); il est évidemment possible de démarrer un service mais aussi de se connecter à lui si il est déjà en cours d'exécution;
- ◆ d'un récepteur des annonces broadcast ou receveurs [broadcast receiver] : ces "receveurs" visent à réagir à un événement et s'arrêtent dès que leur travail est terminé; les annonces peuvent provenir du système (batterie faible, fuseau horaire modifié, etc) ou d'une autre application (exemples typiques : fin d'un téléchargement, fin d'une installation d'une application avec comme réponse l'ajout d'une icône au menu du smartphone).
- ◆ d'un fournisseur de contenu [content provider] : un tel objet rend des ressources de l'application disponibles aux autres applications (celles-ci utiliseront un "content resolver", instance de ContentResolver) pour viser le "content provider" en question et dialoguer avec lui).

5.2 La communication par intents

Les trois premiers types de composants sont activés au moyen d'un "**intent**" : il s'agit d'un message asynchrone spécifiant une action et contenant l'URI du composant à manipuler ou de l'action à initier. Ceci permet aux composants de communiquer car, de base, chaque application fonctionne dans une sandbox distincte. C'est le rôle d'un objet implementant la classe abstraite **Context** (objet fourni par le framework Android au démarrage) de provoquer les démarrages des activités et services, avec par exemple :

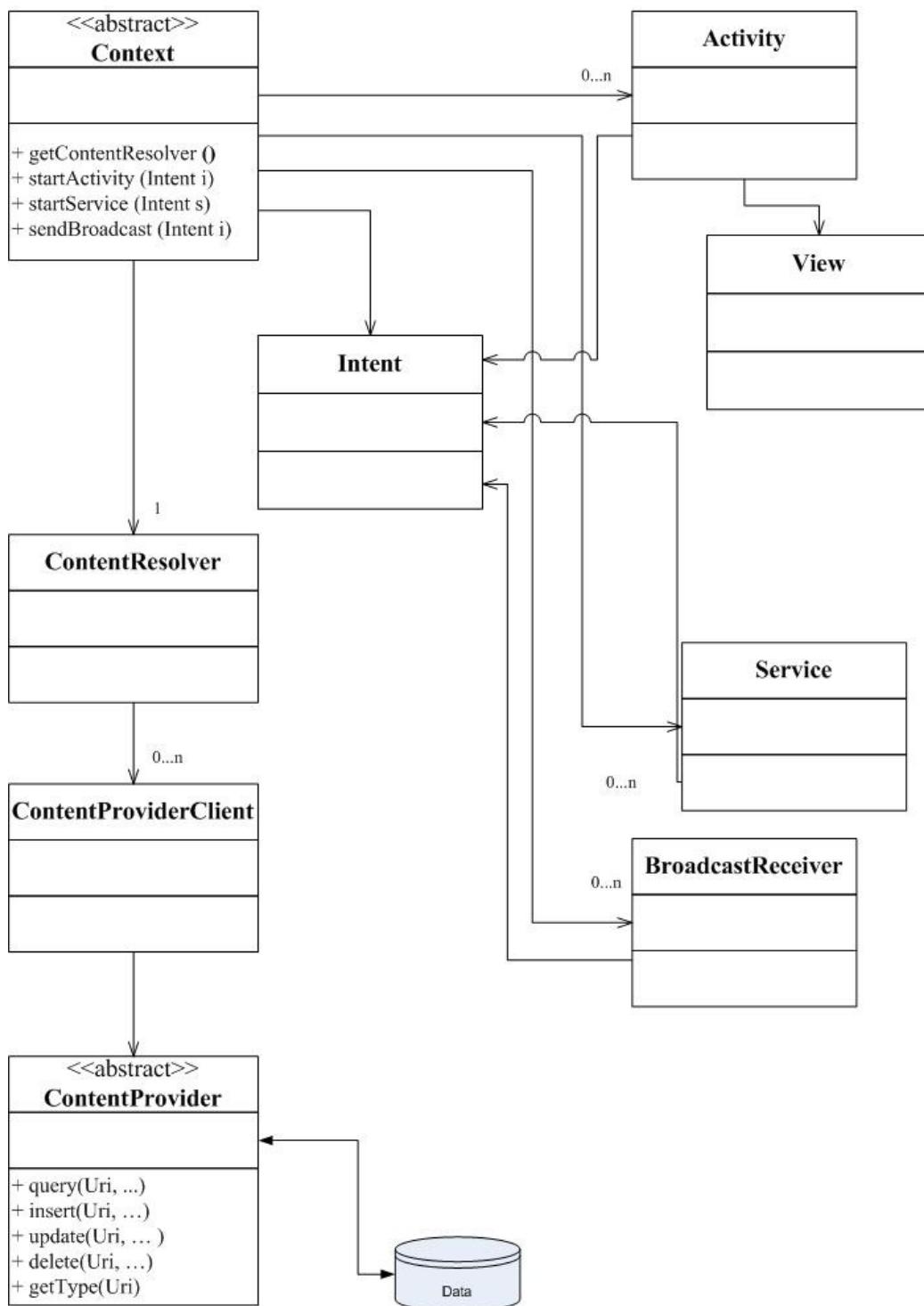
```
public abstract void startActivity (Intent intent)
public abstract ComponentName startService (Intent service)
public abstract boolean stopService (Intent service)
public abstract boolean bindService (Intent service, ServiceConnection conn, int flags)
public abstract void sendBroadcast (Intent intent)
```

et de permettre l'accès aux ressources de l'application, avec par exemple :

public abstract ContentResolver **getContentResolver** () (qui lui-même dispose de la méthode
 public final ContentProviderClient **acquireContentProviderClient** (Uri uri))

public abstract Resources **getResources** ()

Très schématiquement :



5.3 Activités et processus

En définitive, on peut se demander si une activité est en fait un processus ou pas. A vrai dire, cela dépend de la manière dont l'application est configurée dans `AndroidManifest.xml` (que nous examinerons plus en détail plus loin) :

- 1) par défaut : les diverses activités tournent dans un seul processus, même si leur communication est de type IPC;
- 2) on peut faire tourner les diverses activités dans des processus différents au moyen du tag "`android:process`" :

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=http://schemas.android.com/apk/res/android .... ...>
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".HelloBonjour" android:label="@string/app_name"
            android:process="basics.process1">
        </activity>
        <activity android:name=".AuRevoir" android:process=" basics.process3"> ..
        </activity>
    </application> ...
</manifest>
```

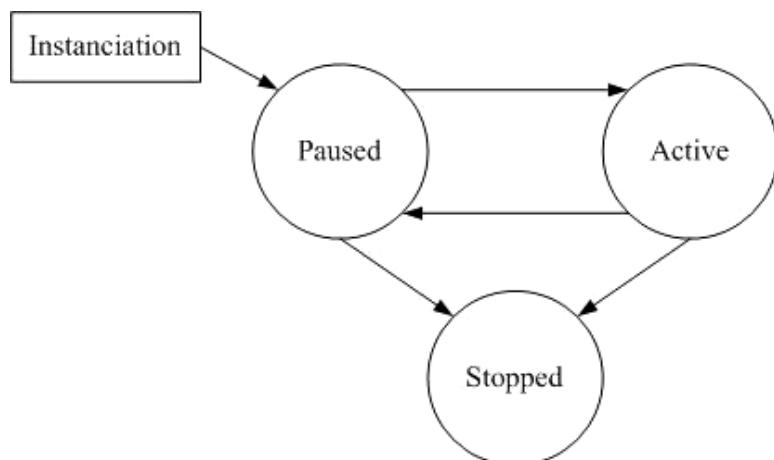
En anticipant, précisons déjà qu'il est possible de récupérer le pid de ce processus par :

```
int id = android.os.Process.myPid(); System.out.println("Process id de l'activité = "+id);
```

6. Le cycle de vie d'une activité Android

Un objet "Activity", instance d'une classe dérivée de la classe `android.app.Activity`, est donc un objet dont on peut lancer l'exécution pour réaliser une tâche quelconque utilisant un GUI, en fait une instance d'une classe dérivée de `android.view.View`. Une application peut consister en une seule activité ou en un groupe d'activités, l'une d'entre elles étant désignée comme celle de départ et chacune pouvant passer la main à une autre.

Le cycle de vie d'une activité rappelle classiquement celui d'une MIDlet de J2ME :



Le passage d'un état à l'autre est notifié à l'application par l'appel aux méthodes (protected) :

cycle de vie de base

void onCreate(Bundle savedInstanceState);	void onDestroy();
---	-------------------

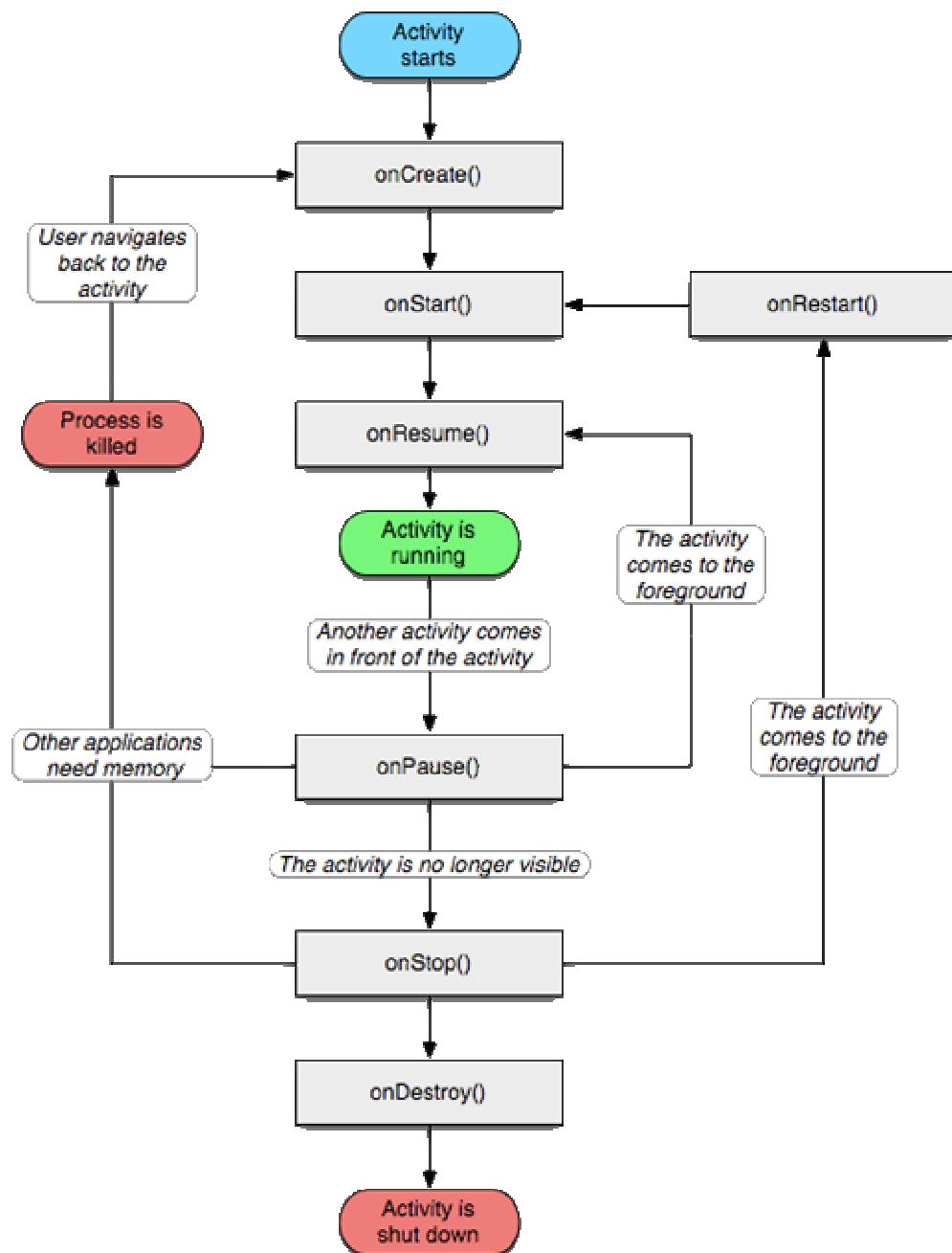
cycle de vie visible

void onStart();	void onStop();
-----------------	----------------

cycle de vie à l'arrière-plan

void onPause();	void onRestart();	void onResume();
-----------------	-------------------	------------------

Les transitions sont schématisées dans la documentation Android de la manière suivante :



On remarque donc que si une activité est à l’arrêt (temporaire ou définitif), le processus qui lui donne son contexte peut être détruit en cas de besoin de mémoire pour d’autres applications.

7. Une application Android de base

Une application peut être constituée de plusieurs activités, mais l'utilisateur n'interagit qu'avec une seule d'entre elles à un instant donné. A priori (même si ce n'est pas indispensable), une application possède au moins une activité. Une classe basique dérivée d'**Activity** va *typiquement redéfinir deux méthodes* :

- 1) protected void **onCreate** (Bundle savedInstanceState)
- Cette méthode est appelée quand l'activité est démarrée.

Le paramètre instance de android.os.**Bundle** est une implémentation de l'interface android.os.**Parcelable** caractérisant les classes qui peuvent être sauveées ou restaurées à partir d'un objet android.os.**Parcel**, celui-ci constituant ce qui est transmis par IPC dans le contexte de la gestion des processus sur le mobile (il ne s'agit donc en aucun cas d'une sérialisation au sens habituel du terme).

Classiquement, la méthode onCreate() appellera notamment :

```
public void setContentView (View view)  
ou  
public void setContentView (int layoutResID)
```

la première forme utilise une classe android.view.**View** qui désigne simplement une zone rectangulaire susceptible de contenir des graphiques et d'interagir avec l'utilisateur; la deuxième faisant référence à une vue considérée comme une ressource définie dans R.java (déjà évoqué - nous allons y revenir).

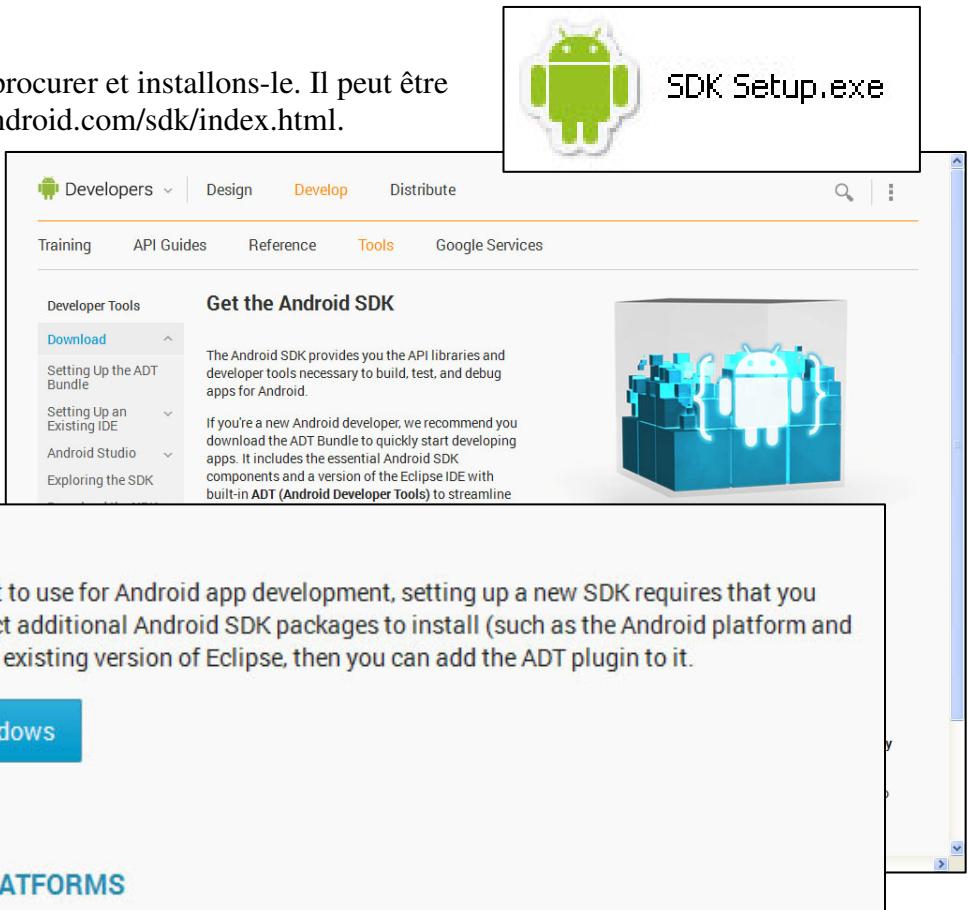
- 2) protected void **onPause** ()
- appelée quand l'activité passe à l'arrière-plan sans être détruite pour la cause.

En pratique, on peut développer en ligne de commande ☺ mais il nous faudrait évidemment un EDI pour nous simplifier la vie ☺ ... Mais il nous faut d'abord le SDK.

8. Le SDK Android et son installation

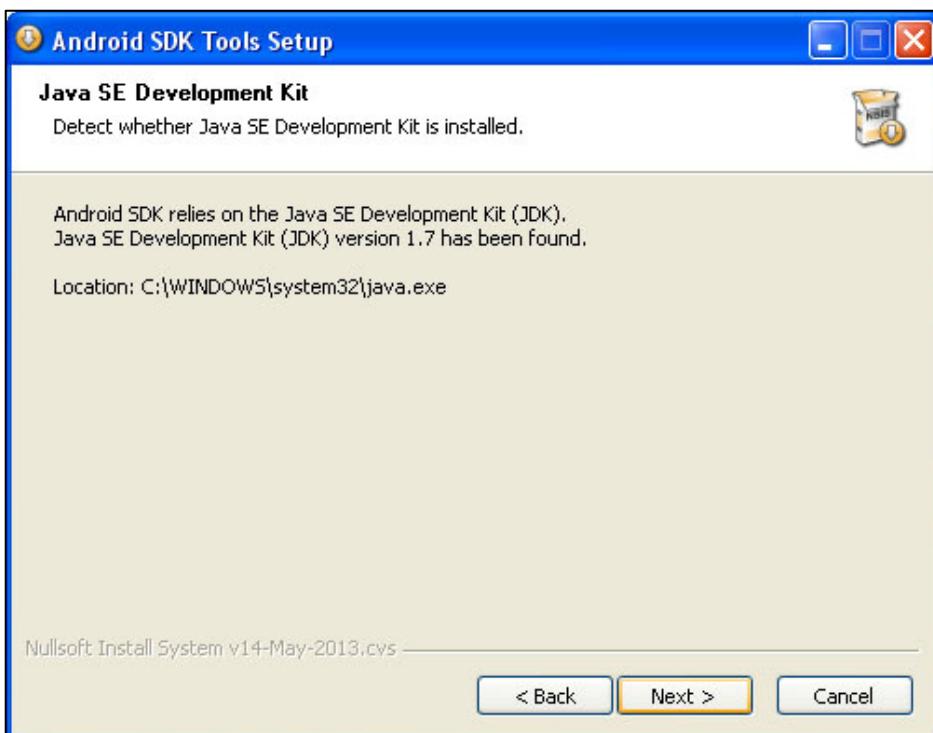
On suppose le JDK "standard" installé et il nous faut il nous faut donc installer un **SDK Android**.

a) Commençons par nous le procurer et installons-le. Il peut être obtenu sur <http://developer.android.com/sdk/index.html>.

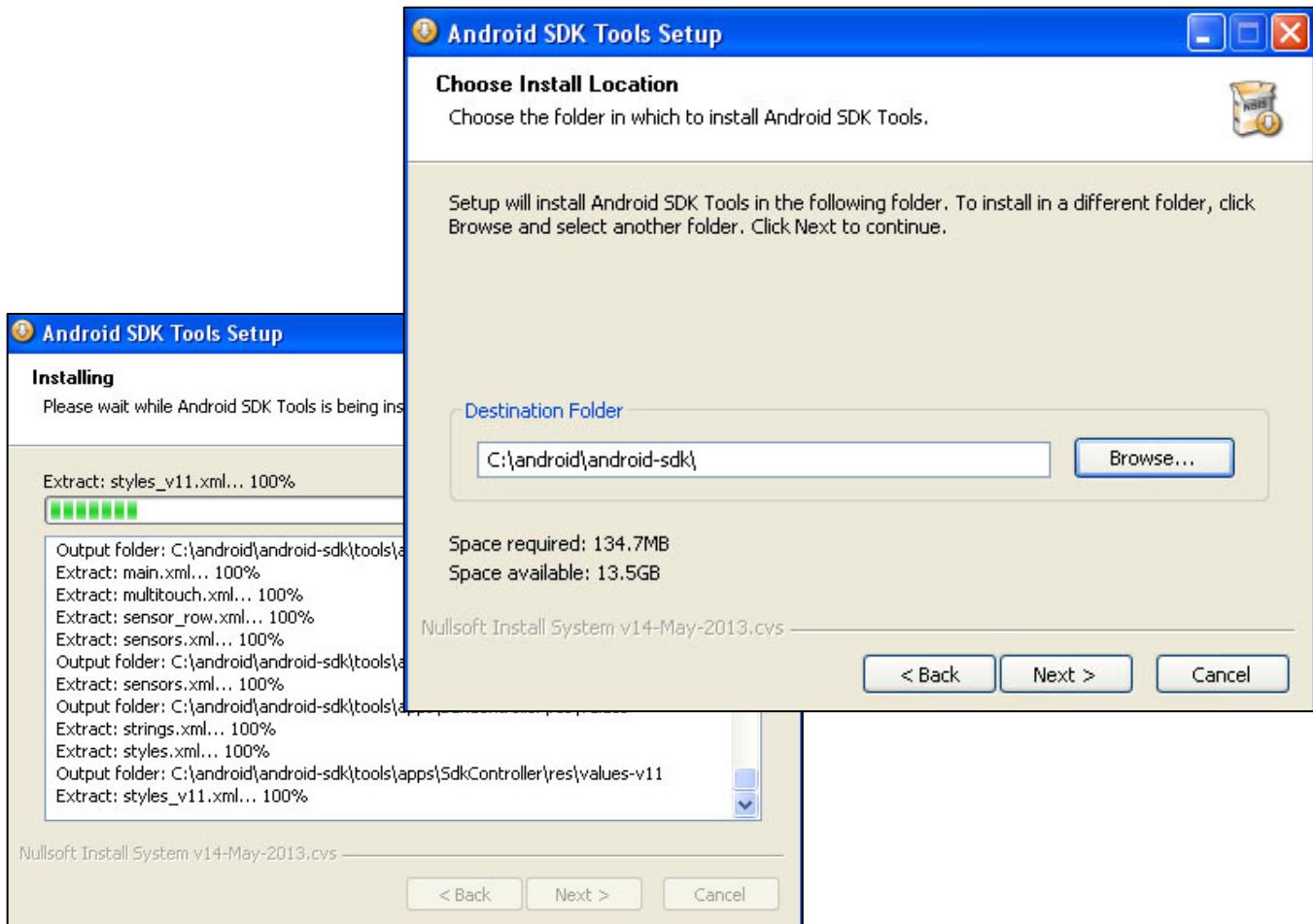


The screenshot shows the official Android developer website. At the top, there's a navigation bar with links for Developers, Design, Develop (which is highlighted in orange), Distribute, Training, API Guides, Reference, Tools, and Google Services. Below this, a sidebar titled 'Developer Tools' has a 'Download' section expanded, showing options like 'Setting Up the ADT Bundle', 'Setting Up an Existing IDE', 'Android Studio', and 'Exploring the SDK'. To the right of the sidebar, a main content area has a heading 'Get the Android SDK' with a sub-section titled 'Download'. It includes a brief description of what the SDK provides and a recommendation to download the ADT Bundle. There's also a large image of the Android robot standing in front of a stack of colorful blocks. At the bottom left, there are two collapsed sections: 'USE AN EXISTING IDE' and 'SYSTEM REQUIREMENTS'. A blue button labeled 'Download the SDK Tools for Windows' is prominently displayed.

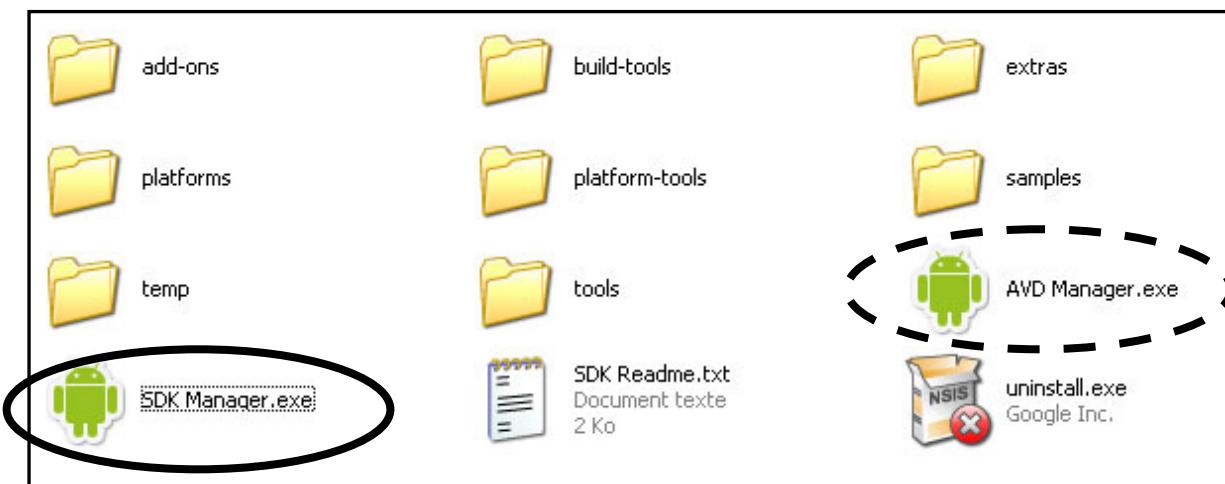
Le programme d'installation recherche le JDK :



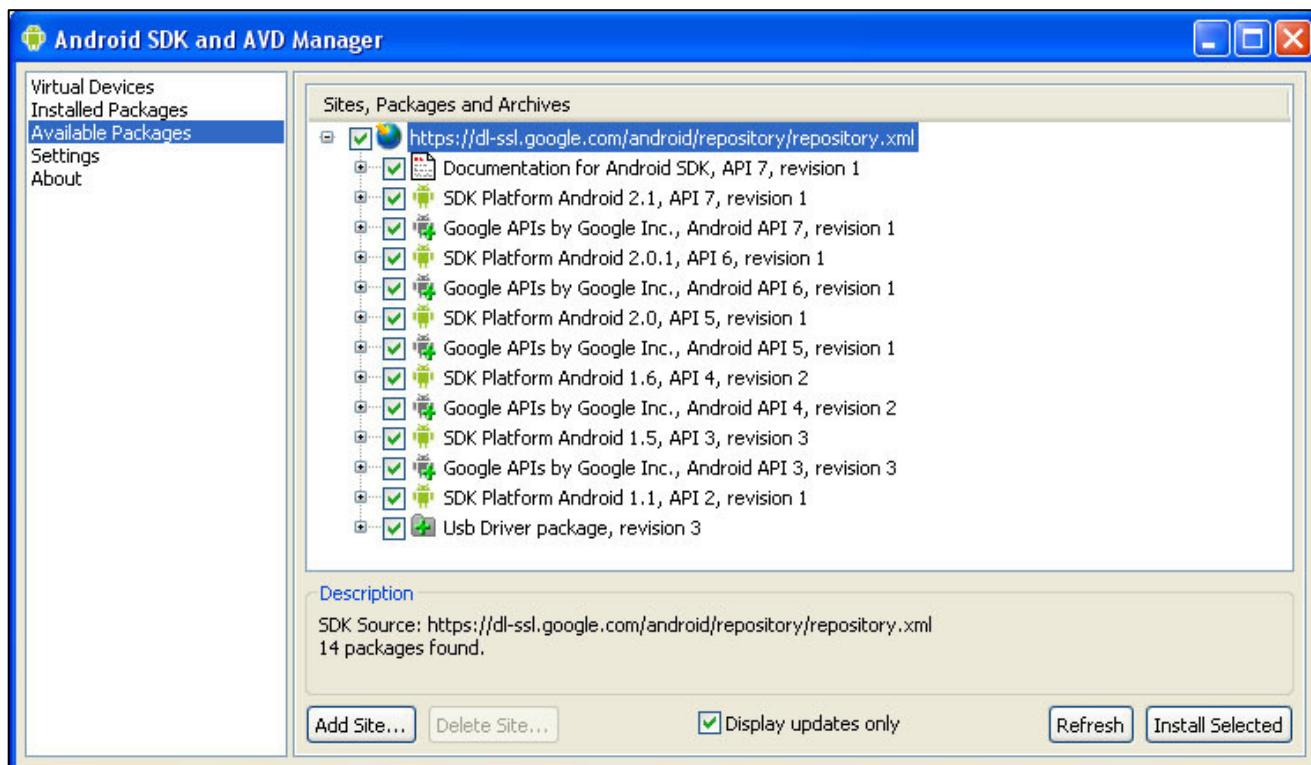
On installe ce SDK dans un répertoire "android-sdk" (par exemple) :



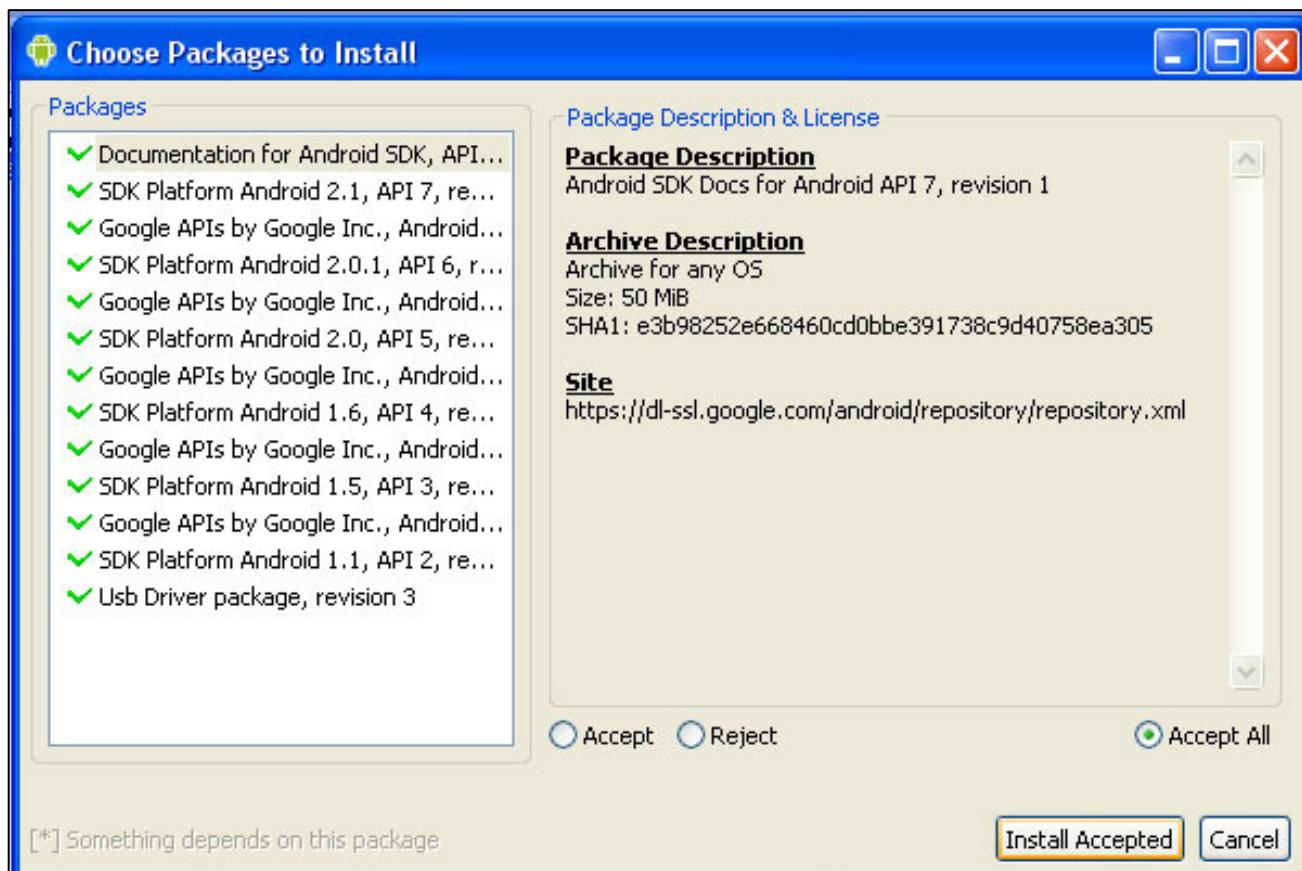
Résultat :

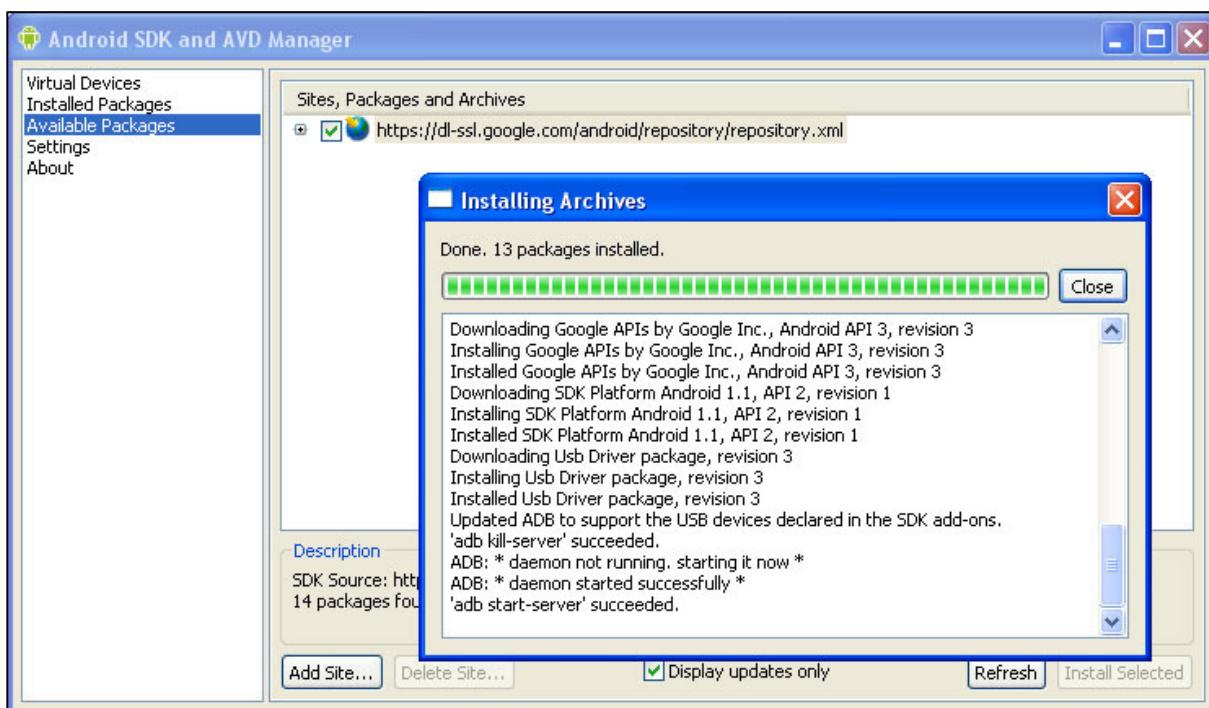


b) Une fois le SDK Manager lancé, les **packages Android** disponibles et installables sont présentés :

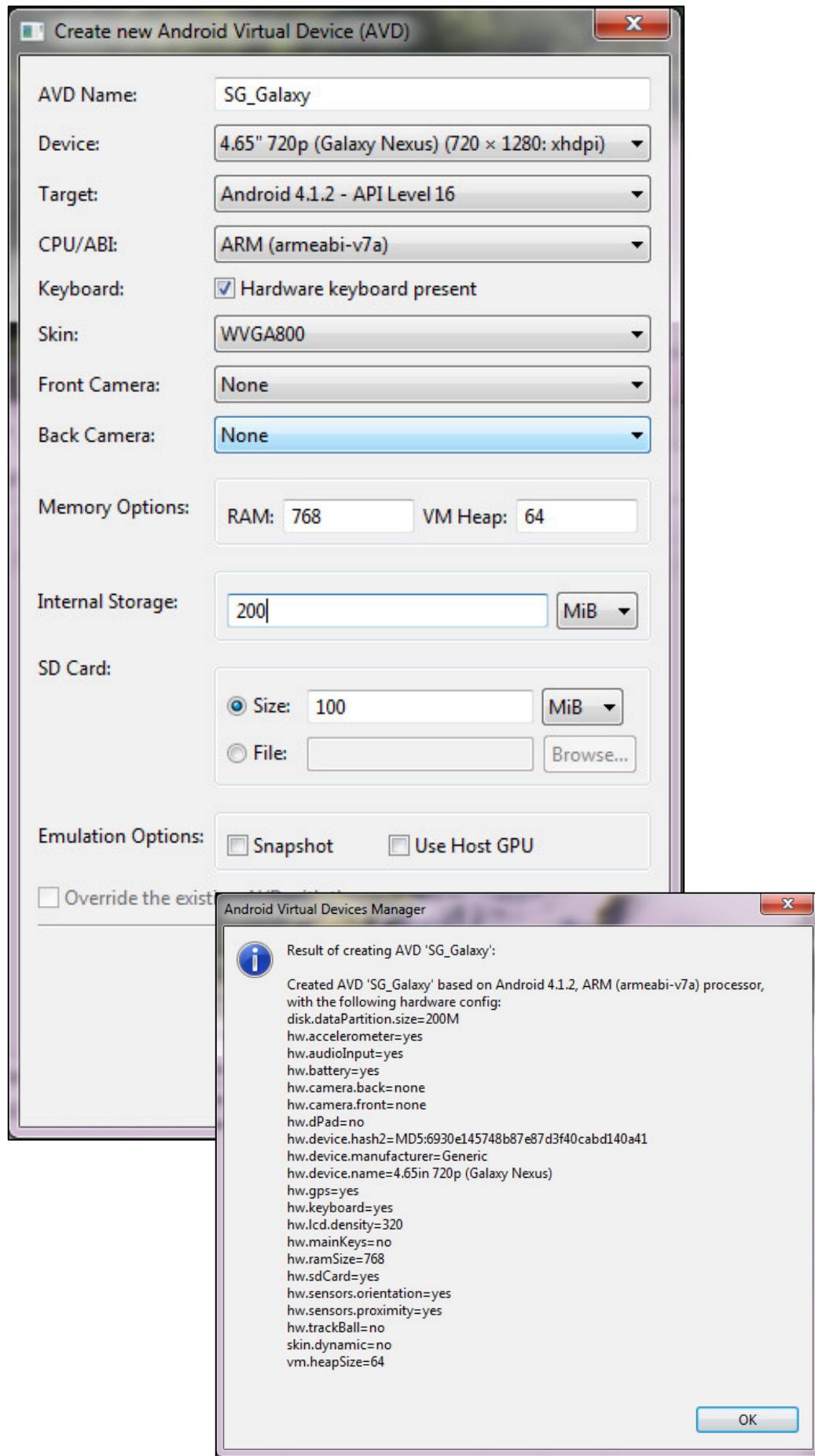


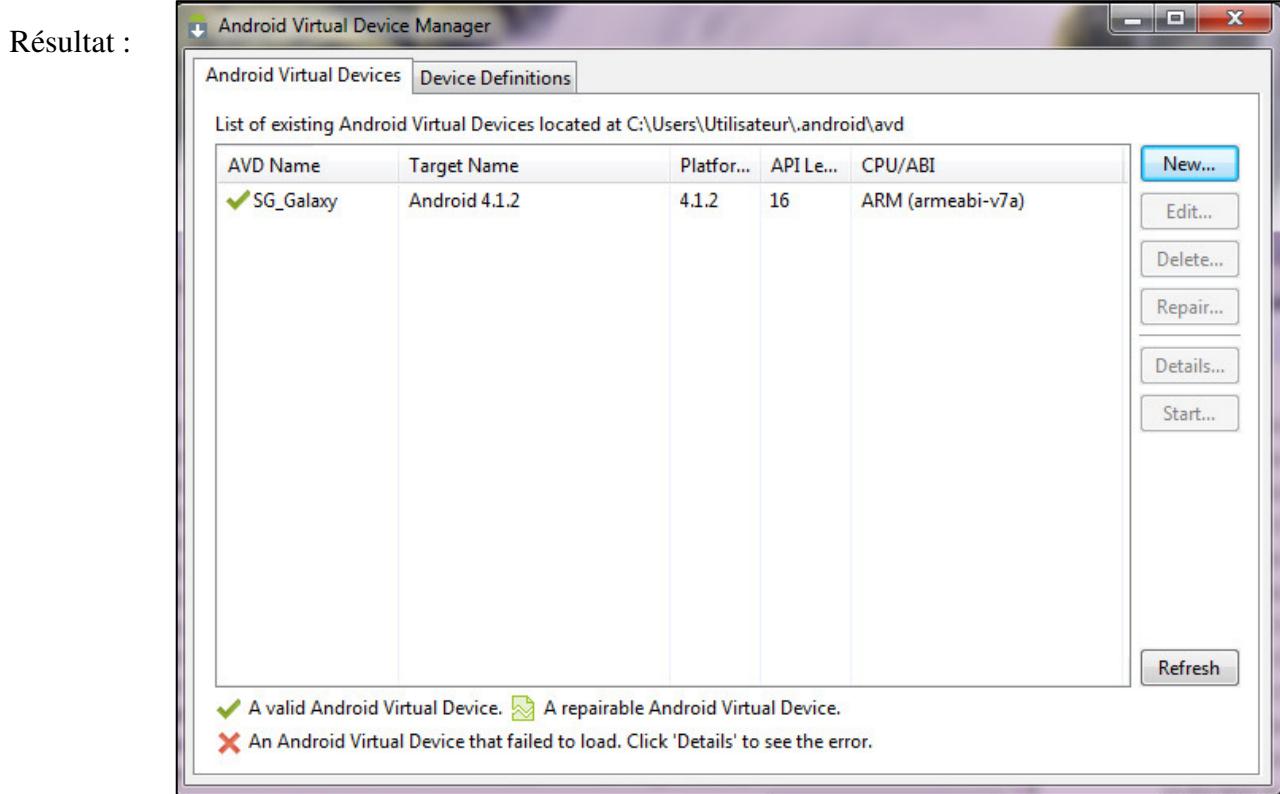
et après avoir sélectionné les packages nécessaires, on peut lancer l'installation.





c) Mais il faut encore installer un **AVD** (Android Virtual Device), c'est-à-dire **une version de l'émulateur de mobile** configuré selon certaines options pour représenter au mieux un mobile qui existe dans la réalité. L'étape est obligatoire, car il n'y a pas d'AVD par défaut. Cela peut se faire avec l'"AVD Manager", appelé directement ou depuis le SDK Manager, dans lequel on peut définir un appareil virtuel. Un tableau initialement vide est destiné à contenir la liste des AVD; un appui sur le bouton New donne le panneau de définition d'un mobile :





9. Le développement manuel en ligne de commande

Une fois le SDK installé et au moins un AVD défini, on peut passer au développement d'applications. Une fois dans notre vie, nous allons faire cela manuellement – histoire d'apprécier ce que les IDEs réalisent pour notre confort ;-). Bien qu'il soit en effet peu probable que l'on soit amené à développer une application Android entièrement manuellement depuis la ligne de commande (encore qu'il semblerait que ce soit parfois nécessaire pour des applications manipulant de grosses ressources qui posent problème à Netbeans ou Eclipse), il n'est en effet pas intéressant de présenter les étapes successives d'un tel développement pour bien appréhender le processus de développement occulté en partie par Eclipse, NetBeans et les autres.



9.1 Les outils et la création du projet de base

L'exécutable de développement se nomme très finement "android" et se trouve dans le répertoire tools. Pour en voir les possibilités (le répertoire racine n'est qu'un exemple) :

```
C:\android\android-sdk_r06-windows\android-sdk-windows\tools>android --help
```

Usage:

android [global options] action [action options]

Global options:

-v --verbose Verbose mode: errors, warnings and informational messages are printed.

-h --help Help on a specific command.

-s --silent Silent mode: only errors are printed out.

Valid actions are composed of a verb and an optional direct object:

```
- list      : Lists existing targets or virtual devices.  
- list avd   : Lists existing Android Virtual Devices.  
- list target : Lists existing targets.  
- create avd  : Creates a new Android Virtual Device.  
- move avd    : Moves or renames an Android Virtual Device.  
- delete avd   : Deletes an Android Virtual Device.  
- create project : Creates a new Android Project.  
- update project : Updates an Android Project (must have an AndroidManifest.xml).  
...  
- update adb    : Updates adb to support the USB devices declared in the SD  
K add-ons.  
- update sdk     : Updates the SDK by suggesting new platforms to install if  
available.
```

Action "list":

Lists existing targets or virtual devices.

Options:

No options

...

```
C:\android\android-sdk_r06-windows\android-sdk-windows\tools>
```

Bigre ;-) ... Nous pouvons par exemple demander la liste des plateformes installées :

```
C:\android\android-sdk_r06-windows\android-sdk-windows\tools>android list targets
```

Available Android targets:

id: 1 or "android-2"

Name: Android 1.1

Type: Platform

API level: 2

Revision: 1

Skins: HVGA (default), HVGA-L, HVGA-P, QVGA-L, QVGA-P

id: 2 or "android-3"

Name: Android 1.5

Type: Platform

API level: 3

Revision: 4

Skins: HVGA (default), HVGA-L, HVGA-P, QVGA-L, QVGA-P

...

id: 7 or "android-8"

Name: Android 2.2

Type: Platform

API level: 8

Revision: 2

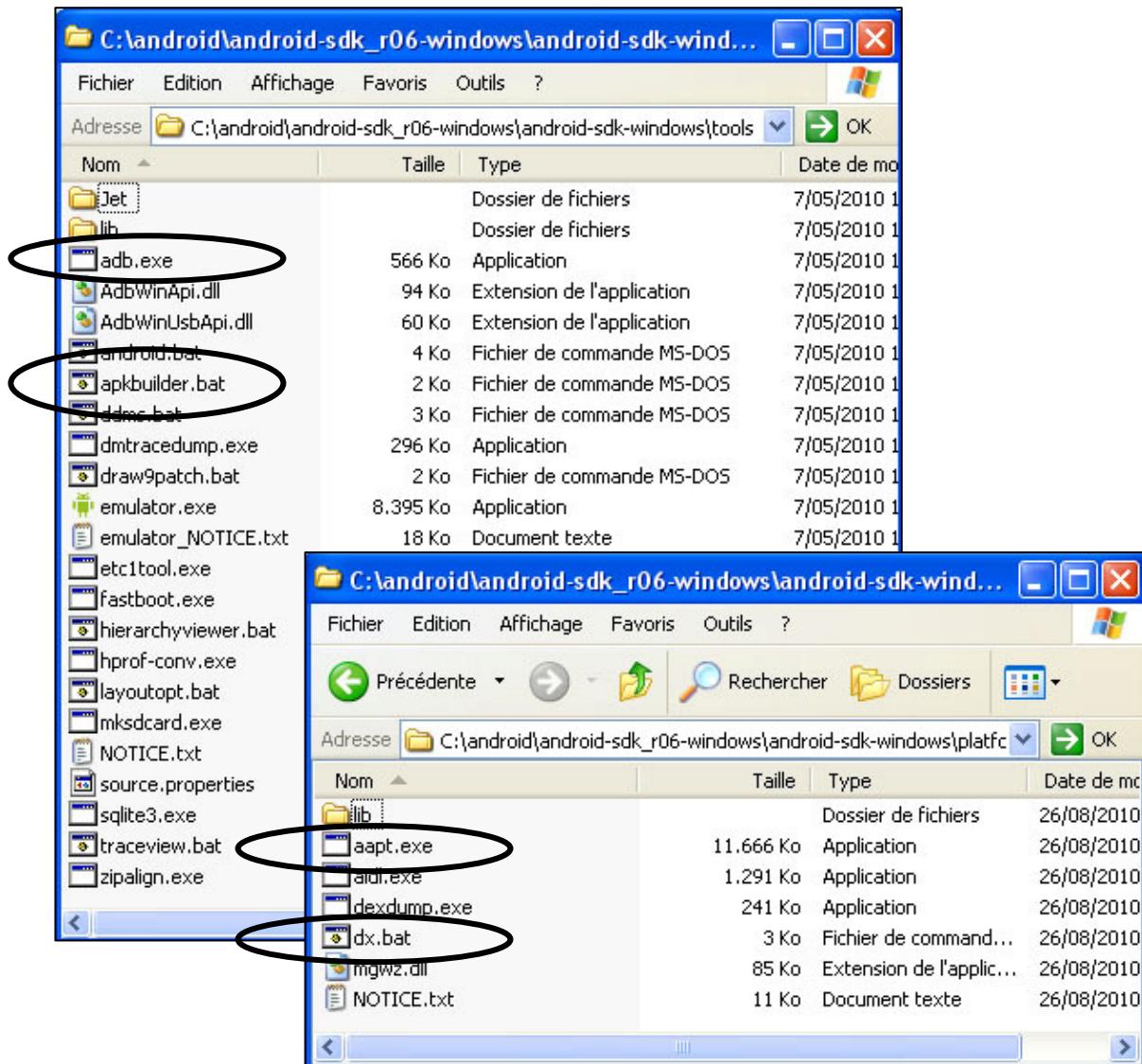
Skins: HVGA (default), QVGA, WQVGA400, WQVGA432, WVGA800, WVGA854

...

```
C:\android\android-sdk_r06-windows\android-sdk-windows\tools>
```

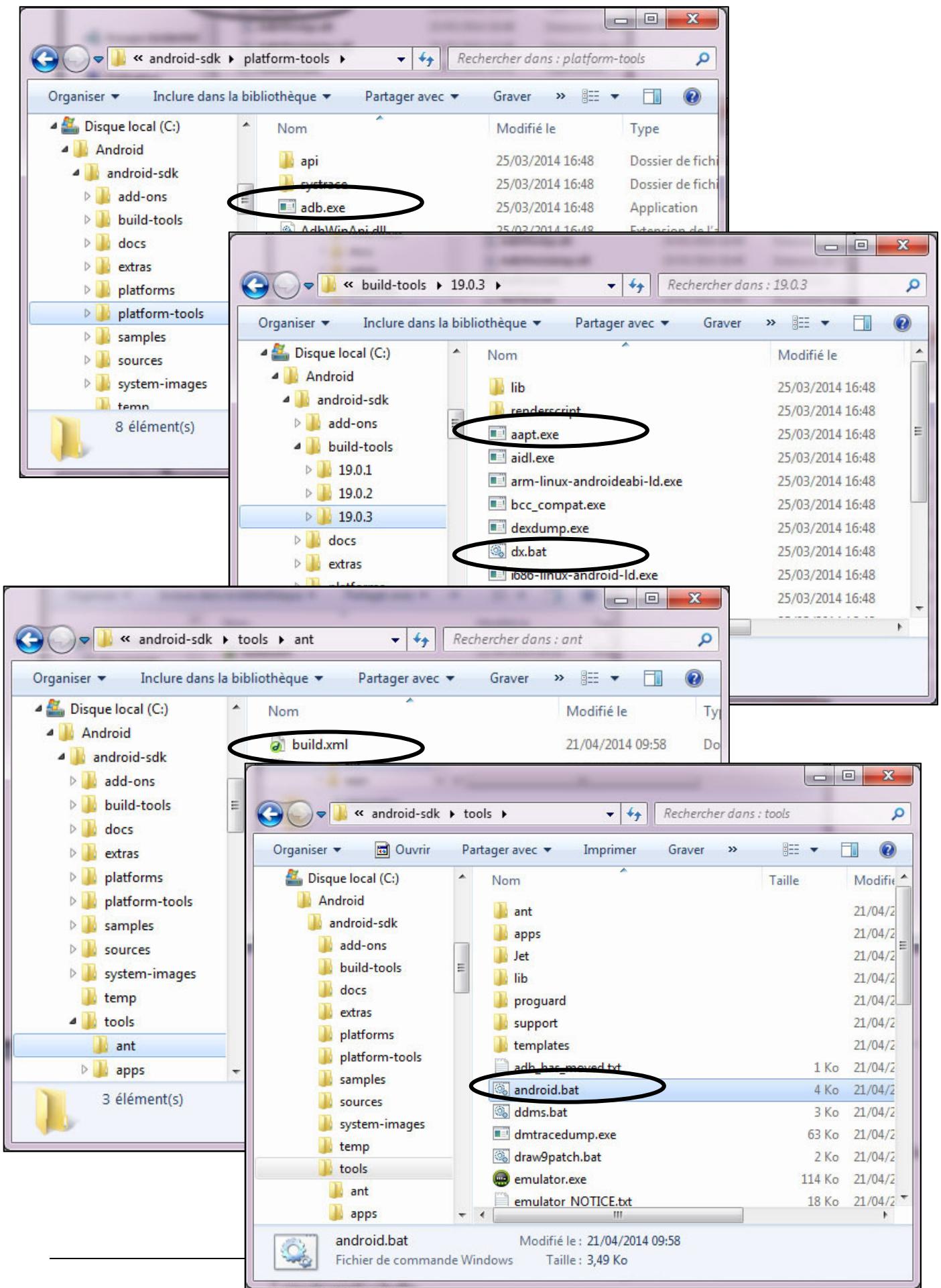
Venons-en au développement proprement dit. On utilise pour ce travail de titan des outils qui se trouvent dans des répertoires du SDK. Pour les anciennes versions :

```
C:\....>set PATH=%android\android-sdk_r06-windows\android-sdk-windows\platforms\android-5\tools%;%PATH%
```



et pour les versions plus récentes (où l'exécutable apkbuilder.bat a été remplacé par un appel de ant sur le fichier build.xml qui contient un item <apkbuilder>) :

```
C:\....>C:\Android\android-sdk>set PATH=C:\Android\android-sdk;C:\Android\android-sdk\tools;C:\Android\android-sdk\platform-tools;%PATH%
```



Le mode opératoire type est du style qui va être décrit dans les paragraphes suivants.

On commence par créer le projet : on crée un projet de base (c'est-à-dire du type "Hello world !") avec

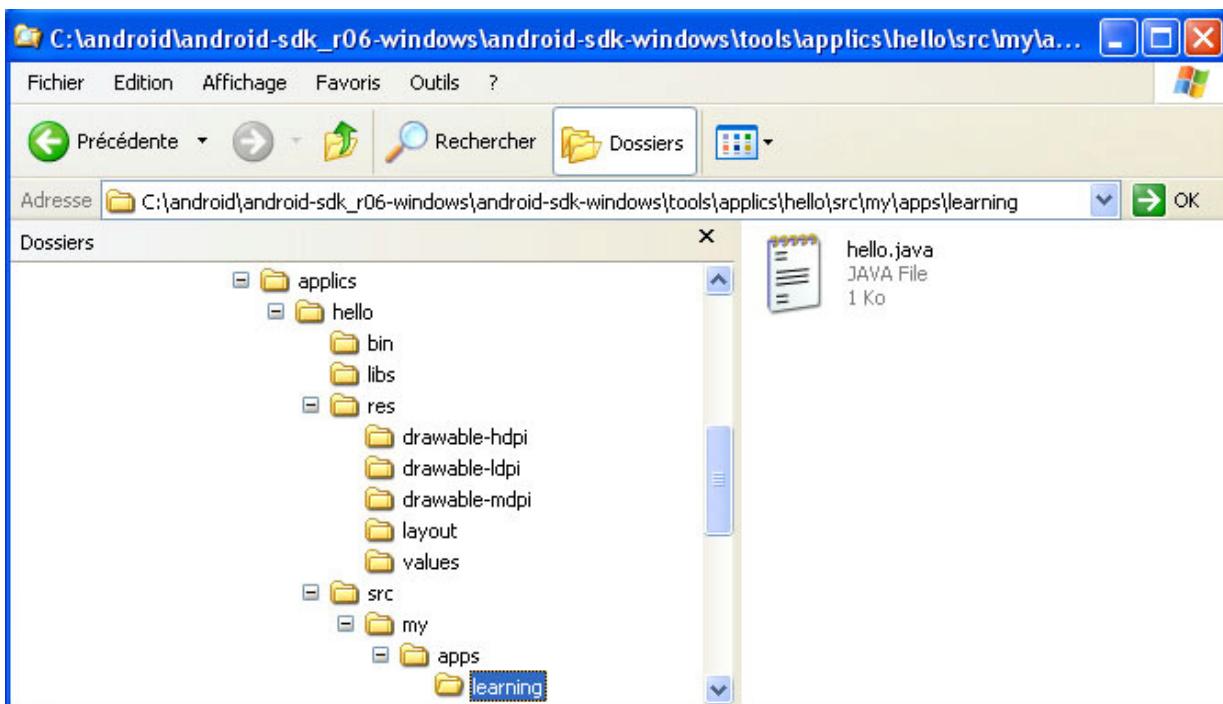
```
C:\Android\android-sdk\tools>>android create project --target "Google Inc.:Google APIs:8"
--path applics/hello --activity hello --package my.apps.learning
```

Bien sûr, on choisit le niveau d'APIs compatible avec l'AVD installé (par exemple: "Google Inc.:Google APIs:**16**"). Le résultat est du type suivant et reflète une certaine activité :

```
Created project directory: C:\android\android-sdk_r06-windows\android-sdk-windows\tools\ap
plics\applications\hello
Created directory C:\android\android-sdk_r06-windows\android-sdk-windows\tools\ap
plics\hello\src\my\apps\learning
Added file C:\android\android-sdk_r06-windows\android-sdk-windows\tools\ap
plics\hello\src\my\apps\learning\hello.java
Created directory C:\android\android-sdk_r06-windows\android-sdk-windows\tools\ap
plics\hello\res
Created directory C:\android\android-sdk_r06-windows\android-sdk-windows\tools\ap
plics\hello\bin
Created directory C:\android\android-sdk_r06-windows\android-sdk-windows\tools\ap
plics\hello\libs
Created directory C:\android\android-sdk_r06-windows\android-sdk-windows\tools\ap
plics\hello\res\values
Added file C:\android\android-sdk_r06-windows\android-sdk-windows\tools\ap
plics\hello\res\values\strings.xml
Created directory C:\android\android-sdk_r06-windows\android-sdk-windows\tools\ap
plics\hello\res\layout
Added file C:\android\android-sdk_r06-windows\android-sdk-windows\tools\ap
plics\hello\res\layout\main.xml
Created directory C:\android\android-sdk_r06-windows\android-sdk-windows\tools\ap
plics\hello\res\drawable-hdpi
Created directory C:\android\android-sdk_r06-windows\android-sdk-windows\tools\ap
plics\hello\res\drawable-mdpi
Created directory C:\android\android-sdk_r06-windows\android-sdk-windows\tools\ap
plics\hello\res\drawable-ldpi
Added file C:\android\android-sdk_r06-windows\android-sdk-windows\tools\ap
plics\hello\AndroidManifest.xml
Added file C:\android\android-sdk_r06-windows\android-sdk-windows\tools\ap
plics\hello\build.xml

C:\android\android-sdk_r06-windows\android-sdk-windows\tools>
```

Résultat :



Les répertoires bin et libs sont vides. Le fichier hello.java ressemble à ceci :

hello.java

```
package my.apps.learning;

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

public class hello extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

9.2 La description des ressources

On observe aussi l'apparition de fichiers XML. L'idée est donc que toutes les propriétés des composants visuels (les "**widgets**") sont accessibles dans le code Java et peuvent également être fixées par des attributs dans le fichier XML du layout. Les avantages de cette manière de faire sont :

- ♦ une séparation de la conception visuelle et du code de la logique de l'application car il suffit alors qu'une équipe travaille sur l'aspect visuel de l'application (sur les fichiers XML) alors qu'une autre équipe s'occupe des traitements de l'application (le code "métier") :
 - ceci permet de rendre le code métier plus bref car moins surchargé;

- cette séparation permet des changements plus rapides et aisés dans les choix des différents paramètres de l'interface

- ◆ une réduction des risques qu'une modification du code source de la vue perturbe l'application : un changement de l'aspect visuel n'implique pas de changer quoi que ce soit dans le code métier car il suffit de modifier le fichier XML.

La classe View utilisée comme telle ne permet pas de développer des interfaces très avancées. Cette classe est en effet tellement générique qu'elle n'a que peu d'intérêt si elle est employée comme telle. Il faut plutôt utiliser une de ses classes filles, Button, ImageView, CheckBox, etc.

L'utilisation et le positionnement des vues dans une activité se feront la plupart du temps en utilisant une mise en page qui sera composée d'un ou plusieurs layouts et de widgets (voir plus loin – paragraphe 16).

Pour notre modeste exemple :

```
res\layout\main.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="Hello World, hello"
    />
</LinearLayout>
```

et

```
res\values\strings.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">hello</string>
</resources>
```

Le répertoire res peut en réalité comporter d'autres répertoires qui contiennent des ressources particulières, comme **res\drawable** (pour les images utilisées par l'application), **res\menu** (pour les fichiers xml décrivant les menus utilisés par l'application), **res\anim** (pour la description d'animations de certains éléments) et **res\raw** (pour de grosses ressources extérieures, comme un fichier pdf). Nous y reviendrons.

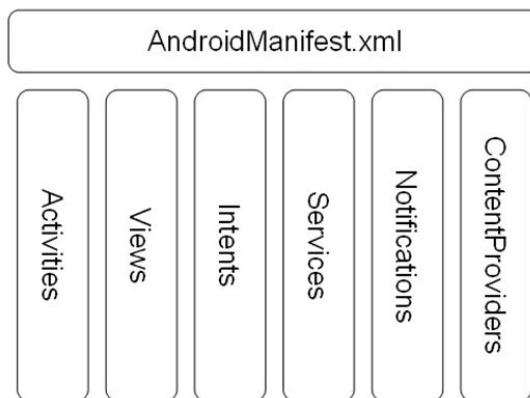
A remarquer également les répertoires **res\drawable-hdpi**, **res\drawable-ldpi** et **res\drawable-mdpi** destinés à contenir des ressources graphiques adaptées à la taille du smartphone utilisateur (petit ou grand smartphone, tablette). Le problème de la taille d'un texte ou d'un composant adapté au mobile utilisé est géré de la même manière au moyen de

fichier res\layout-small, res\layout-normal, res\layout-large, res\layout-xlarge et res\values-small, res\values-normal, etc.

Enfin, l'internationalisation peut être gérée au moyen de fichiers res\values-en\strings.xml, res\values-fr\strings.xml, res\values-nl\strings.xml, ...

9.3 Le manifeste de l'application

Toute application Android doit posséder un fichier **AndroidManifest.xml** dans son répertoire racine. Son rôle est bien sûr de décrire l'application en termes de packages, classes, versions, permissions, etc. Mais sa première raison d'être est de déclarer les composants utilisables par le système : les composants qui n'y sont pas déclarés sont invisibles au système et ne pourront donc pas être exécutés.



Dans notre cas, le fichier manifeste est simplement :

AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="applics.basics"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".HelloBonjour"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="7" />
</manifest>
  
```

Seules les balises <manifest> et <application> sont requises. A remarquer que le nom de classe précédé d'un point ("HelloBonjour") est un raccourci pour signifier que ce nom doit être préfixé du nom du package (ici, "applics.basics"). On remarquera aussi la définition de l'**intent** (pour rappel, un intent est un message asynchrone spécifiant une action et contenant l'URI de la donnée à manipuler) : il spécifie qu'il faut lancer (LAUNCHER) l'activité "HelloBonjour" en tant que composant initial (MAIN).

9.4 La génération du fichier R

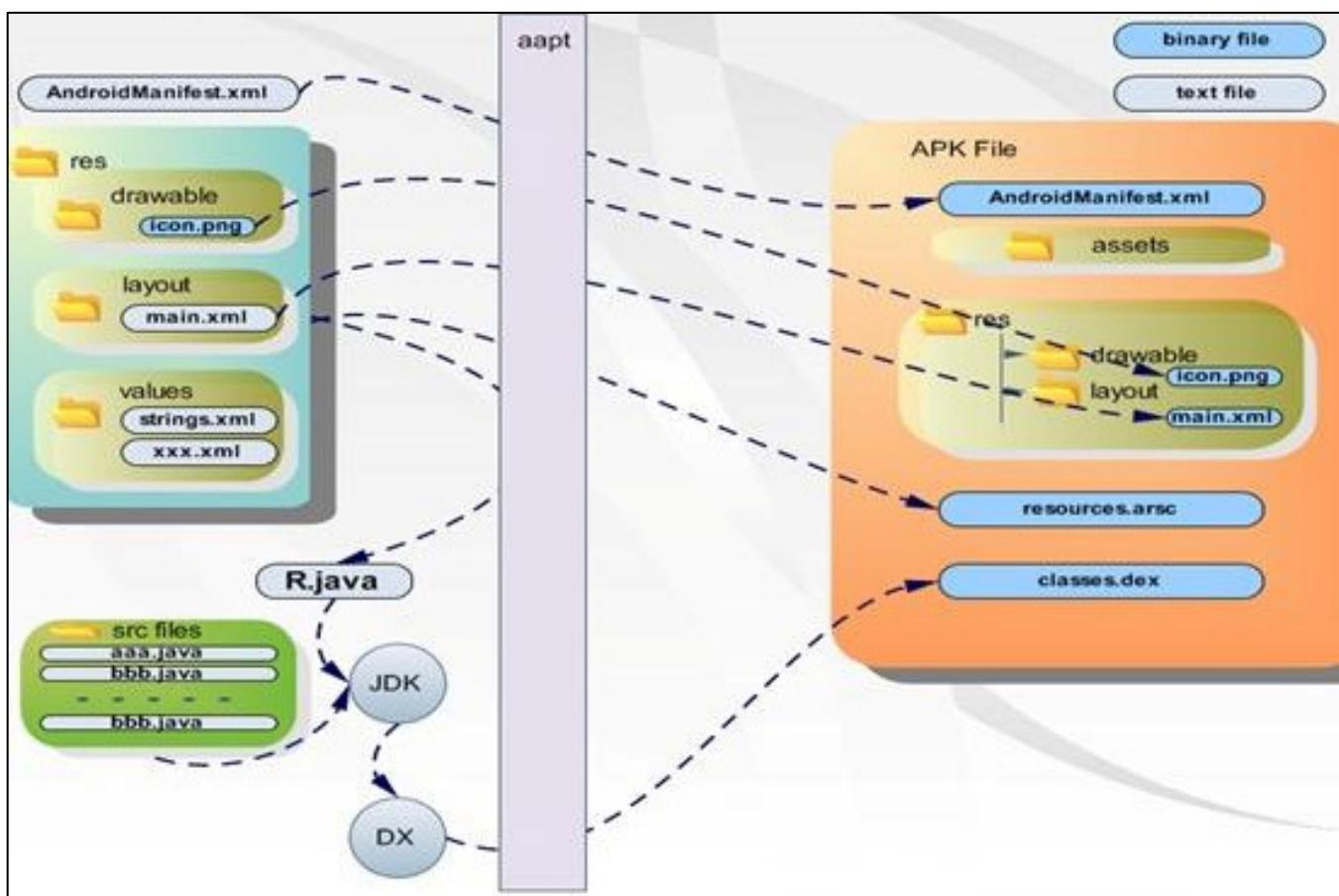
Le premier outil à invoquer ensuite est **aapt** : son rôle est de *construire le fichier R.java* qui décrit les ressources et *le fichier resources.arsc* qui contient la description de ces ressources sous forme binaire. Pour cette action, on sous-entend donc que

- ◆ on a bien rédigé (ou généré) un fichier **AndroidManifest.xml**;
- ◆ on a créé un répertoire de ressources (typiquement nommé "res") structuré en sous-répertoires comme par exemple **layout** et **values** pour contenir les fichiers descriptifs XML (que l'on aura généré en utilisant les templates qui se trouvent dans android-sdk-windows\platforms\android-X\templates) ainsi que **drawable** pour une icône png, le tout empaqueté dans un fichier au format zip.

La syntaxe de la ligne de commande est alors du type :

```
aapt package -f -M <fichier manifeste> -F <fichier empaquetant les ressources> -I <path pour les jar d'Android> -S <rédertoire des ressources d'Android> [-m -J <rédertoire où placer le R.java produit>]
```

En fait, toutes les ressources, qu'il s'agisse de fichiers séparés ou de ressources définies dans le code, se voient en fait attribuer par aapt un identifiant de 32bits (1^{er} byte = référence du package, usuellement 0x7f pour les applications, 2^{ème} byte = type de la ressource, les deux derniers bytes étant le nom de la ressource). Après cette attribution et la compilation des sources, aapt génère le fichier R (dans un dossier "gen" - logique) :



Dans notre cas, celui-ci a la forme suivante :

```
R.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 my.apps.learning;

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=0x7f040000;
    }
}
```

où l'on reconnaît un identifiant comme "app_name". En fait, il ne tiendra qu'à nous de créer des identifiants pour tous les éléments de nos fichiers XML (un composant, une chaîne de caractères, une image, ...) avec un attribut du type :

`android:id="@+id/<nom_donné au composant>"`

pour qu'ils soient répercutés dans le fichier R.java (après chaque recompilation) sous la forme d'une classe statique supplémentaire id. Par exemple :

a) dans main.xml :

```
...
<TextView
    android:id="@+id/texte_horloge"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Heure de Bruxelles"
    android:layout_gravity="center_horizontal" />
...
</TextView>
```

b) dans R.java :

```
...
public final class R {
    ...
    public static final class id {
        ...
        public static final int texte_horloge=0x7f060002;
        ...
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    ...
}
```

Toutes les ressources sont alors accédées dans le code Java au moyen de ce fichier R, dès qu'elles sont déclarées dans un fichier XML placé dans le répertoire res/* adéquat :

R sert de pont entre les fichiers xml et le code applicatif Java en étant constitué de toute une série de sous-classes statiques (en fait, une par type de composant) contenant les identifiants des diverses ressources sous formes de variables de classe (comme "app_name" ou "texte_horloge").

Les ressources sont alors désignées selon la syntaxe :

R.type_ressource.nom_ressource

qui est de type int ("type_ressource" est "id" pour un composant graphique, "string" pour une chaîne, etc). Il s'agit en fait de l'identifiant de la ressource qui permet de retrouver la ressource visée, le fichier resources.arsc contenant ces informations sous forme binaire, dans un format permettant un parcours efficace lors de l'exécution de l'application.

On peut donc utiliser cet identifiant pour récupérer la ressource en utilisant

- ♦ pour les objets graphiques (au sens large), une méthode spécifique de l'activité permet de les récupérer à partir de leur id, ce qui permet d'agir sur ces instances même si elles ont été créées via leur définition XML:

public final View **findViewById** (int id)

ex:

```
TextView texte = (TextView)findViewById(R.id. texte_horloge);
texte.setText("All you need is Java");
```

- ♦ ou alors la classe Resources :

```
Resources res = getResources();
String napp = res.getString(R.string.app_name);
```

La correspondance entre les fichiers xml définis dans les divers sous-répertoires de res et les classes statiques de la classe R est résumée dans le tableau ci-dessous :

sous-répertoire	ressources décrites par xml	classe statique R associée
anim/	animations	R.anim
color/	listes de couleurs	R.color
drawable/	formulaires, listes, images	R.drawable
layout/	layout au sens habituel en Java	R.layout.
menu/	menus	R.menu
raw/	ressources diverses et inclassables, obtenues par Resources.openRawResource()	R.raw
values/	chaînes de caractères, nombres, etc : integers.xml colors.xml strings.xml	R.integer R.colors R.string
xml/	ressources obtenues par Ressources.getXML().	

9.5 Le développement et le déploiement

2) On peut ensuite compiler le source .java et le fichier R.java obtenu à l'étape précédente au moyen du compilateur javac du JDK :

| **javac *.java**

3) On génère ensuite le bytecode modifié avec l'utilitaire **dx** :

dx –dex –output=<nom du fichier dex produit> <répertoire des fichiers .class à traiter> <fichiers jar à utiliser>

4) On peut enfin créer le fichier apk avec l'utilitaire shell **apkbuilder** :

apkbuilder <nom du fichier apk produit> -u -z <fichier des ressources> -f <fichier dex>

On produit ainsi un fichier apk non signé (c'est la signification du commutateur –u).

5) A priori, on ne distribue que des applications signées – on usera donc du keytool pour créer ou gérer un keystore :

keytool –list <keystore utilisé>

6) Il reste à signer l'apk (qui, rappelons-le, a un format zip/jar) au moyen de l'utilitaire jarsigner :

jarsigner -keystore <nom du keystore> -storepass <mot de passe du keystore> -keypass <mot de passe de l'alias utilisé> -signedjar <nom du fichier signé à produire> <nom du fichier à signer> <alais dans le keystore>

7) Il reste à installer l'application au moyen de l'outil adb (Android Debug Bridge) :

adb -d install -r <fichier apk>

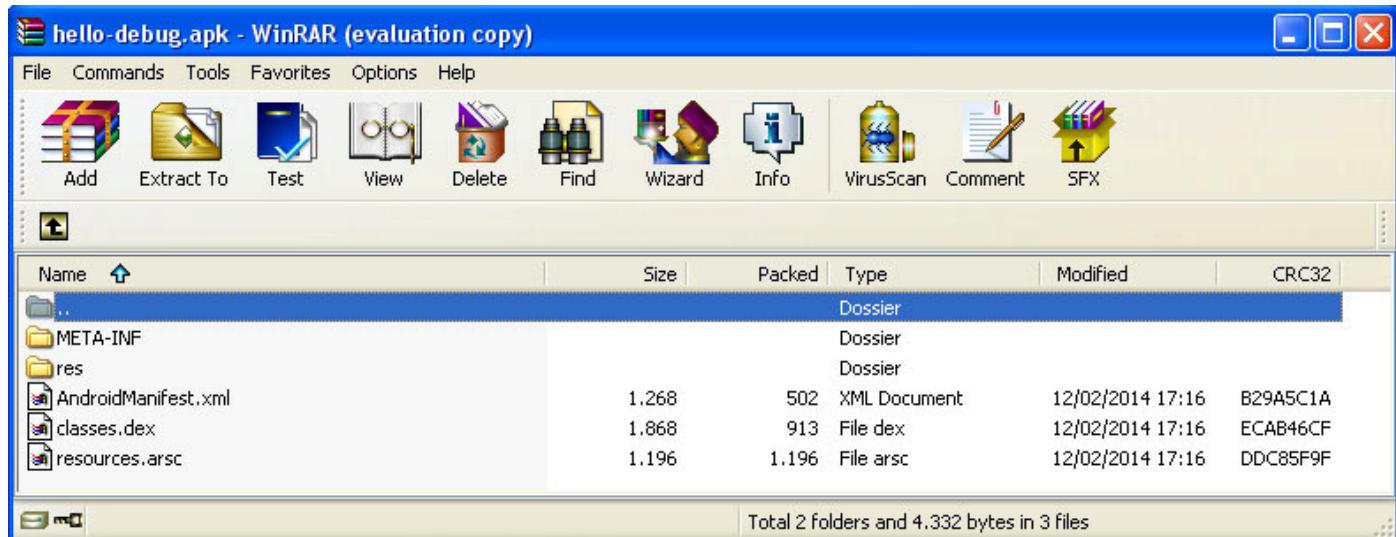
8) Accessoirement, on peut visualiser le contenu du fichier apk avec l'outil aapt :

```
C:\java-android-application\Coucou\bin>aapt list -v Coucou.apk
```

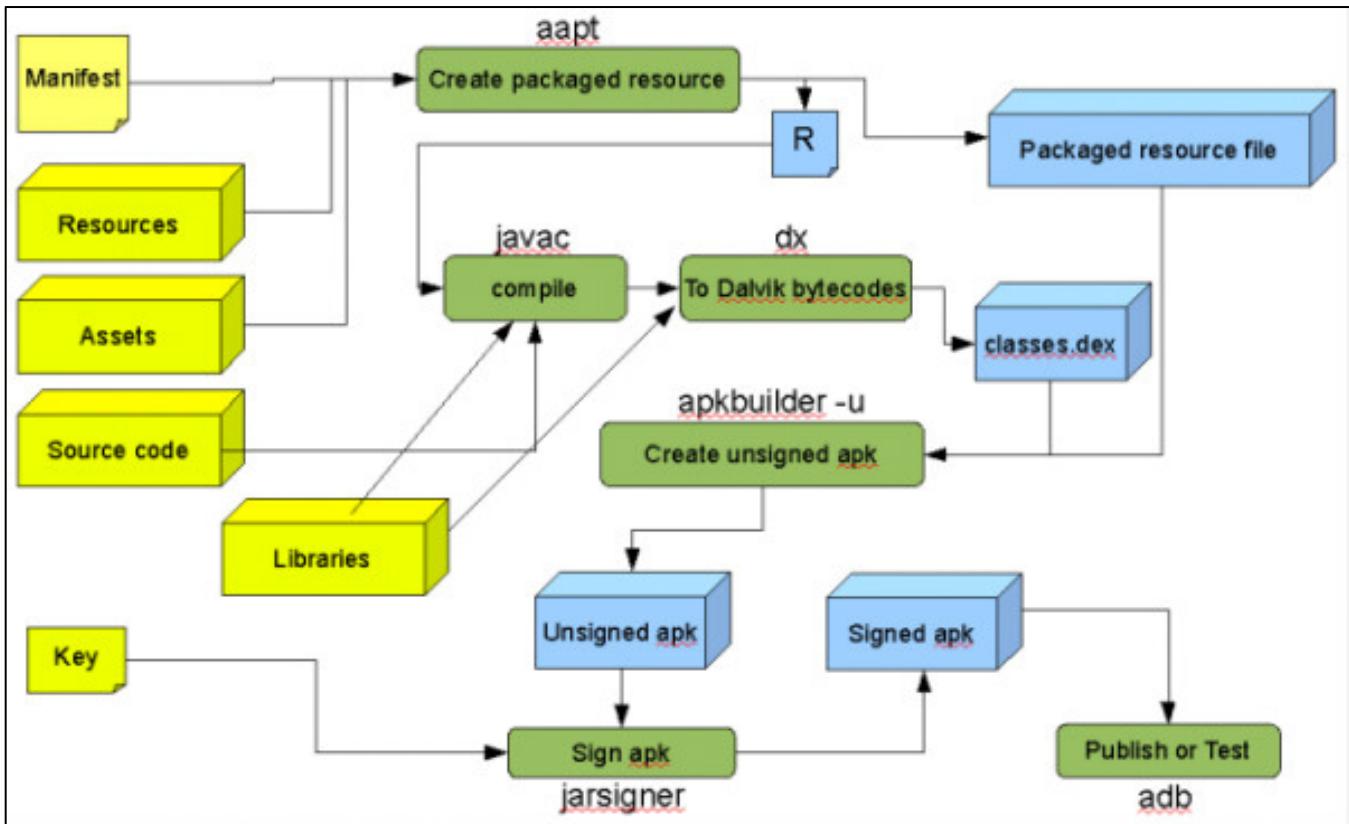
Archive: Coucou.apk

Length	Method	Size	Ratio	Date	Time	CRC-32	Name
2200	Stored	2200	0%	08-26-10	20:34	99a4f90b	res/drawable/icon.png
640	Deflate	277	57%	08-26-10	20:34	8ae7db9b	res/layout/main.xml
1424	Deflate	555	61%	08-26-10	20:34	17fb42dc	AndroidManifest.xml
1060	Stored	1060	0%	08-26-10	20:34	4e2f08e7	resources.arsc
1900	Deflate	932	51%	08-26-10	20:34	ad5379e5	classes.dex
401	Deflate	267	33%	08-26-10	20:34	e7e2db67	META-INF/MANIFEST.MF
454	Deflate	299	34%	08-26-10	20:34	bf3ab414	META-INF/CERT.SF
776	Deflate	603	22%	08-26-10	20:34	8d19d2aa	META-INF/CERT.RSA
<hr/>							
8855		6193	30%				8 files

ou encore avec un outil de la famille Winzip-Winrar-Winace :



En résumé, les différentes phases peuvent se schématiser ainsi :



9.6 Le processus de déploiement avec ant

En fait, tout le processus de génération et d'installation du fichier apk qui représentera l'application, peut se faire en regroupant toutes ces opérations au moyen de l'outil ant :

```

C:\android\android-sdk_r06-windows\android-sdk-windows\tools\apps\hello>ant c
lean install
Buildfile: C:\android\android-sdk_r06-windows\android-sdk-windows\tools\apps\hello\build.xml

[setup] Android SDK Tools Revision 6
[setup] Project Target: Google APIs
[setup] Vendor: Google Inc.
[setup] Platform Version: 2.2
[setup] API level: 8
[setup] WARNING: No minSdkVersion value set. Application will install on all
Android versions.
[setup] Importing rules file: platforms\android-8\ant\ant_rules_r2.xml
.....
install:
[echo] Installing C:\android\android-sdk_r06-windows\android-sdk-windows\to
ols\apps\hello\bin\hello-debug.apk onto default emulator or device...
[exec] error: device not found
[exec] * daemon not running. starting it now *
[exec] * daemon started successfully *
  
```

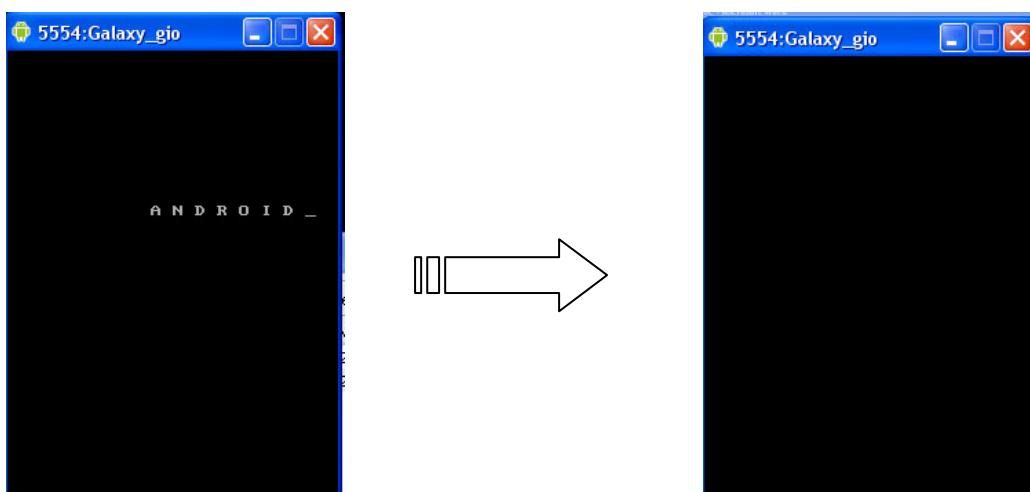
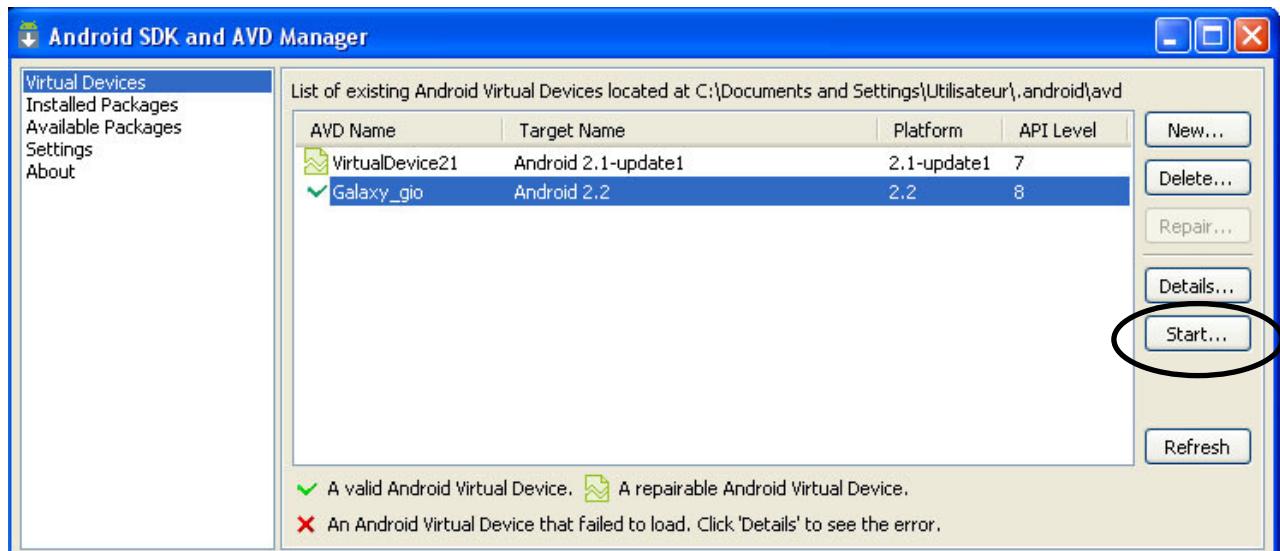
BUILD FAILED

C:\android\android-sdk_r06-windows\android-sdk-windows\platforms\android-8\ant\ant_rules_r2.xml:362: The following error occurred while executing this line:
C:\android\android-sdk_r06-windows\android-sdk-windows\platforms\android-8\ant\ant_rules_r2.xml:191: exec returned: 1

Total time: 11 seconds

C:\android\android-sdk_r06-windows\android-sdk-windows\tools\apps\hello>

Eh oui : il convient de faire démarrer l'émulateur au préalable, puisqu'on ira jusqu'au déploiement :



On recommence :

C:\android\android-sdk_r06-windows\android-sdk-windows\tools\apps\hello>ant c
lean install
Buildfile: C:\android\android-sdk_r06-windows\android-sdk-windows\tools\apps\hello\build.xml
[setup] Android SDK Tools Revision 6

```
[setup] Project Target: Google APIs
[setup] Vendor: Google Inc.
[setup] Platform Version: 2.2
[setup] API level: 8
[setup] WARNING: No minSdkVersion value set. Application will install on all
Android versions.
[setup] Importing rules file: platforms\android-8\ant\ant_rules_r2.xml

clean:
[delete] Deleting directory C:\android\android-sdk_r06-windows\android-sdk-wi
ndows\tools\applics\hello\bin
...
-compile-tested-if-test:
-dirs:
[echo] Creating output directories if needed...
[mkdir] Created dir: C:\android\android-sdk_r06-windows\android-sdk-windows\
tools\applics\hello\gen
[mkdir] Created dir: C:\android\android-sdk_r06-windows\android-sdk-windows\
tools\applics\hello\bin
[mkdir] Created dir: C:\android\android-sdk_r06-windows\android-sdk-windows\
tools\applics\hello\bin\classes

-resource-src:
[echo] Generating R.java / Manifest.java from the resources...
[null] (skipping index file 'C:\android\android-sdk_r06-windows\android
-sdk-windows\tools\applics\hello\res\drawable-hdpi\Thumbs.db')
[null] (skipping index file 'C:\android\android-sdk_r06-windows\android
-sdk-windows\tools\applics\hello\res\drawable-ldpi\Thumbs.db')
[null] (skipping index file 'C:\android\android-sdk_r06-windows\android
-sdk-windows\tools\applics\hello\res\drawable-mdpi\Thumbs.db')

-aidl:
[echo] Compiling aidl files into Java classes...

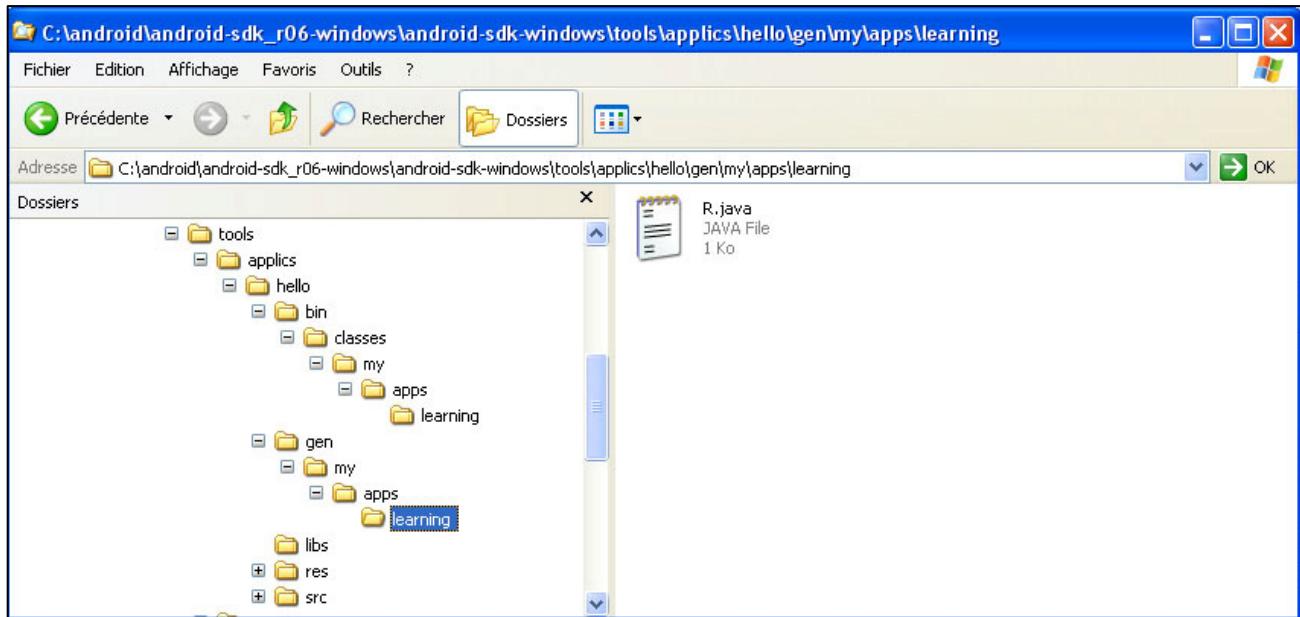
compile:
[javac] C:\android\android-sdk_r06-windows\android-sdk-windows\platforms\and
roid-8\ant\ant_rules_r2.xml:255: warning: 'includeanruntime' was not set, defau
lting to build.sysclasspath=last; set to false for repeatable builds
[javac] Compiling 2 source files to C:\android\android-sdk_r06-windows\andro
id-sdk-windows\tools\applics\hello\bin\classes

-dex:
[echo] Converting compiled files and external libraries into C:\android\and
roid-sdk_r06-windows\android-sdk-windows\tools\applics\hello\bin\classes.dex...

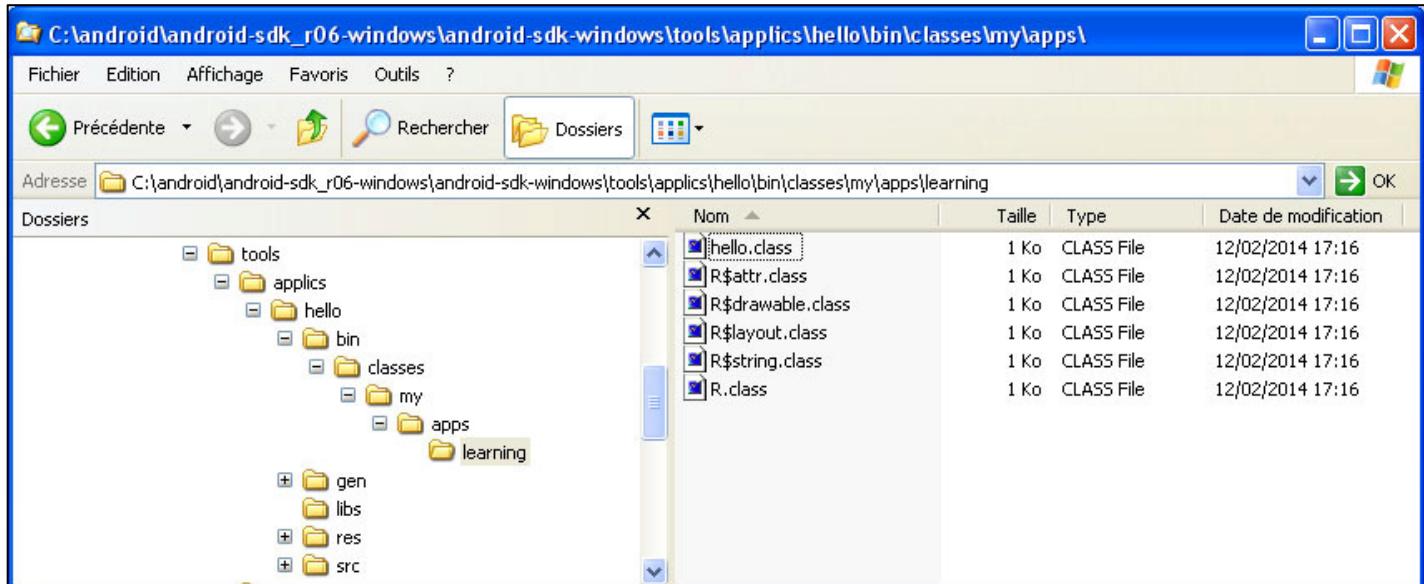
-package-resources:
[echo] Packaging resources
[aaptexec] Creating full resource package...
[null] (skipping index file 'C:\android\android-sdk_r06-windows\andro
id-sdk-windows\tools\applics\hello\res\drawable-hdpi\Thumbs.db')
```

```
...  
-package-debug-sign:  
[apkbuilder] Creating hello-debug-unaligned.apk and signing it with a debug key.  
...  
[apkbuilder] Using keystore: C:\Documents and Settings\Utilisateur\.android\debu  
g.keystore  
  
debug:  
[echo] Running zip align on final apk...  
[echo] Debug Package: C:\android\android-sdk_r06-windows\android-sdk-window  
s\tools\apps\hello\bin\hello-debug.apk  
  
install:  
[echo] Installing C:\android\android-sdk_r06-windows\android-sdk-windows\to  
ols\apps\hello\bin\hello-debug.apk onto default emulator or device...  
[exec] pkg: /data/local/tmp/hello-debug.apk  
[exec] Success  
[exec] 852 KB/s (0 bytes in 13632.000s)  
  
BUILD SUCCESSFUL  
Total time: 9 seconds  
C:\android\android-sdk_r06-windows\android-sdk-windows\tools\apps\hello>
```

Que ce soit en version pas à pas ou en version compacte (ant), on peut vérifier que de nouveaux fichiers sont apparus :



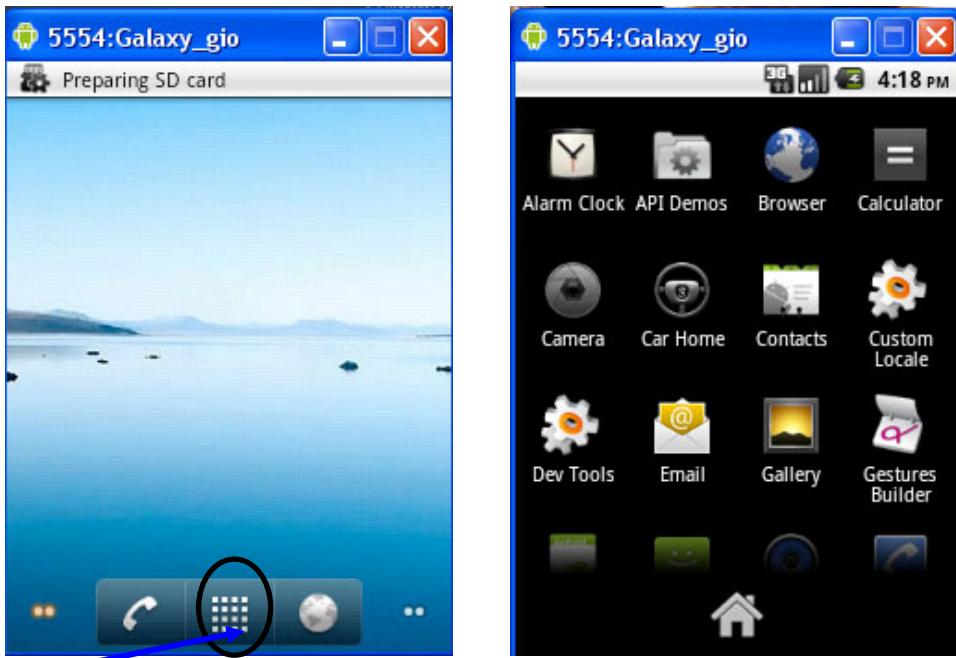
ainsi que les fichiers compilés :



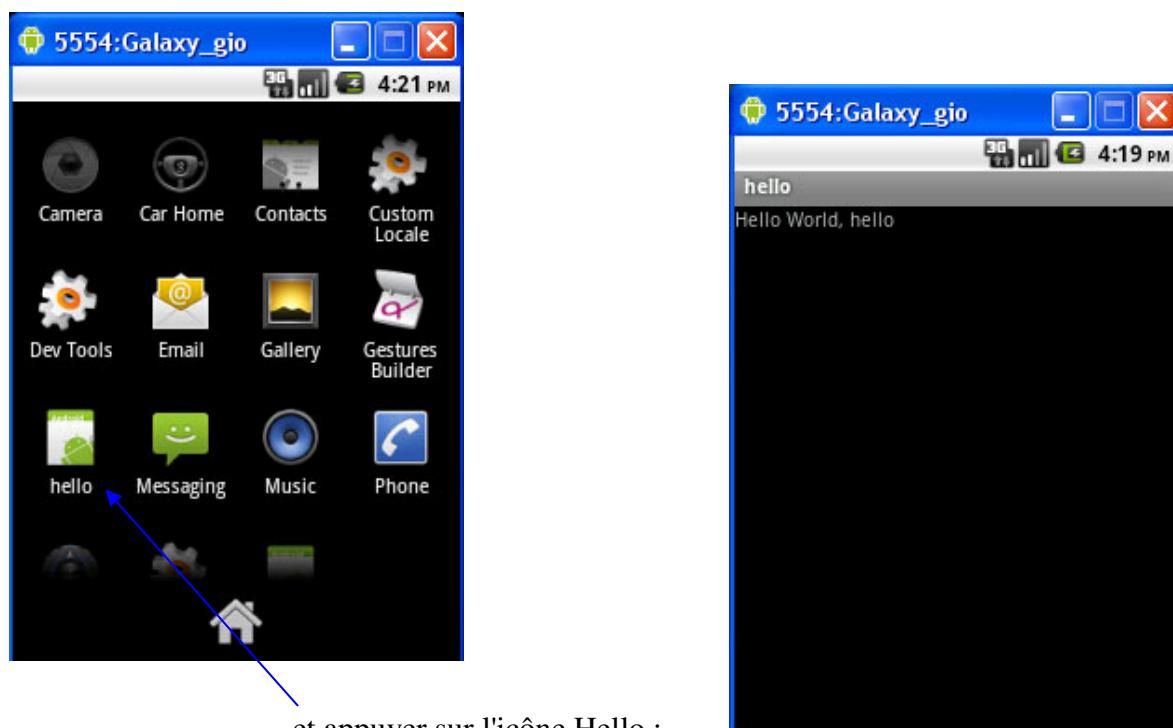
et surtout :

9.7 L'exécution de l'application

Le résultat, après avoir éventuellement déverrouillé le mobile, sera :



Appuyer sur la touche "Menu" :
Faire glisser pour déplacer les icônes vers le haut :



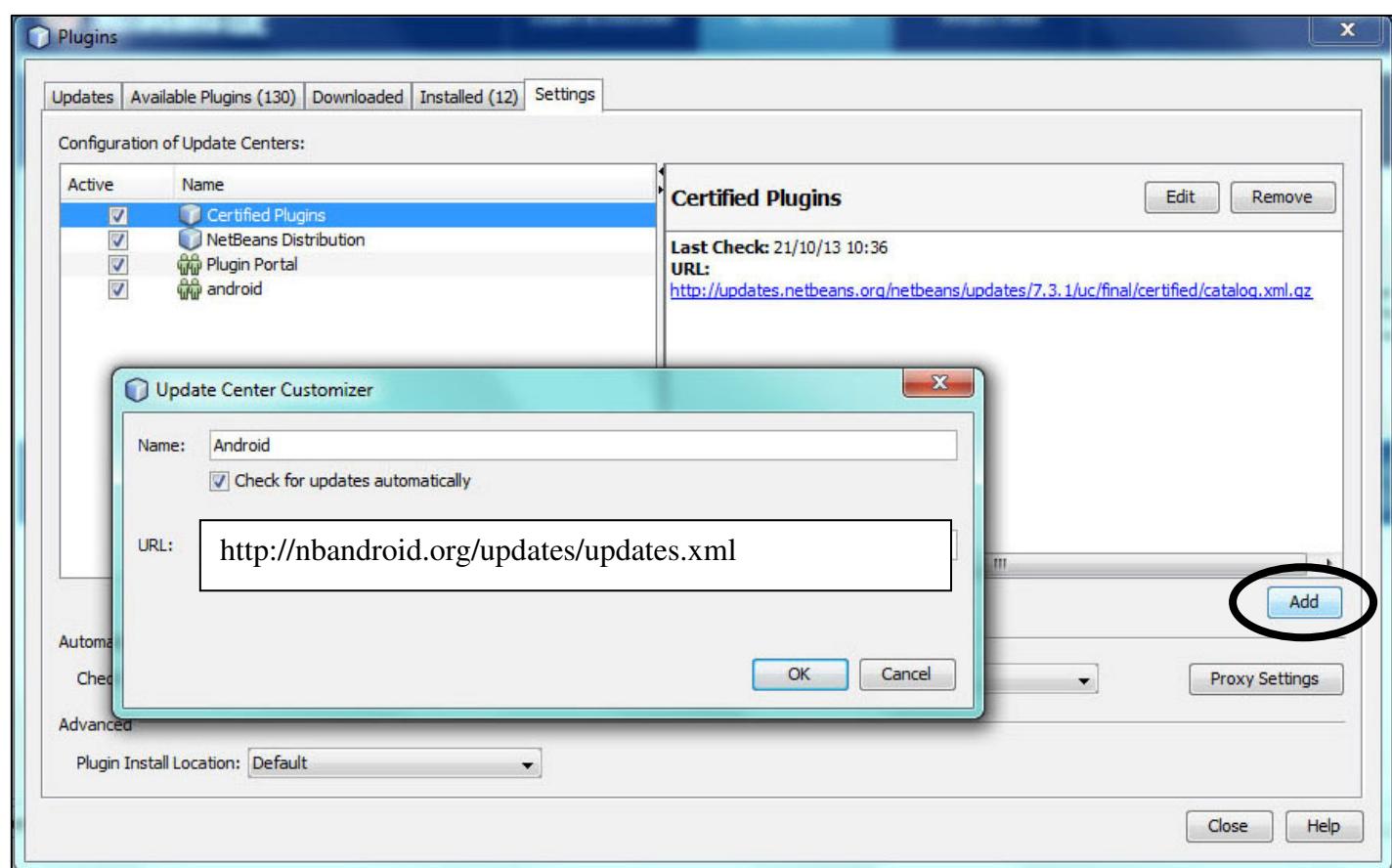
et finir par Esc pour revenir en arrière (avec ce genre d'émulateur).

Bon, passons à présent à une EDI moins revêche ... Une large communauté de développeurs Android utilise Eclipse (= "ni Dieu ni Maître" ;-)), mais comme nous avons toujours utilisé NetBeans ...

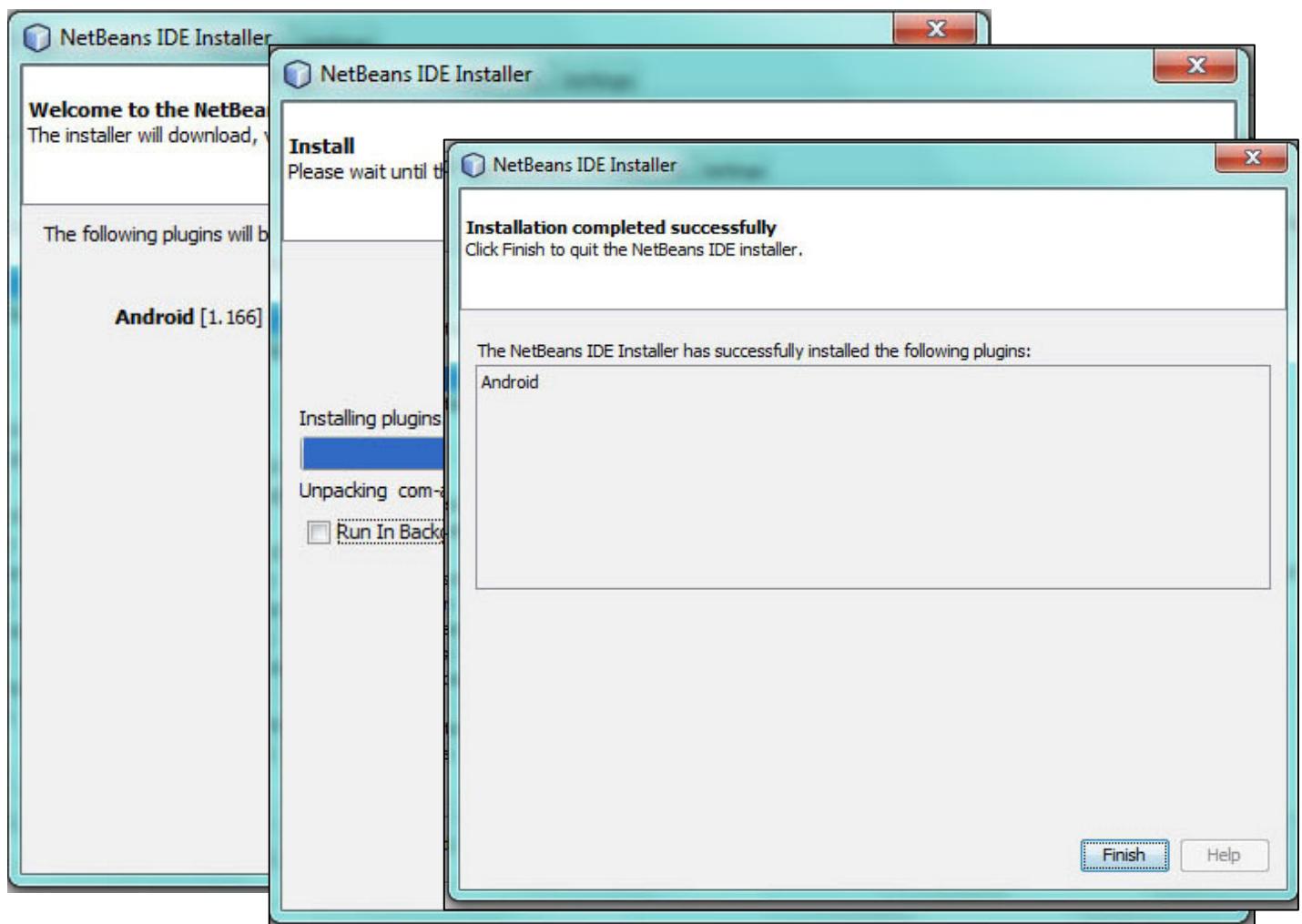
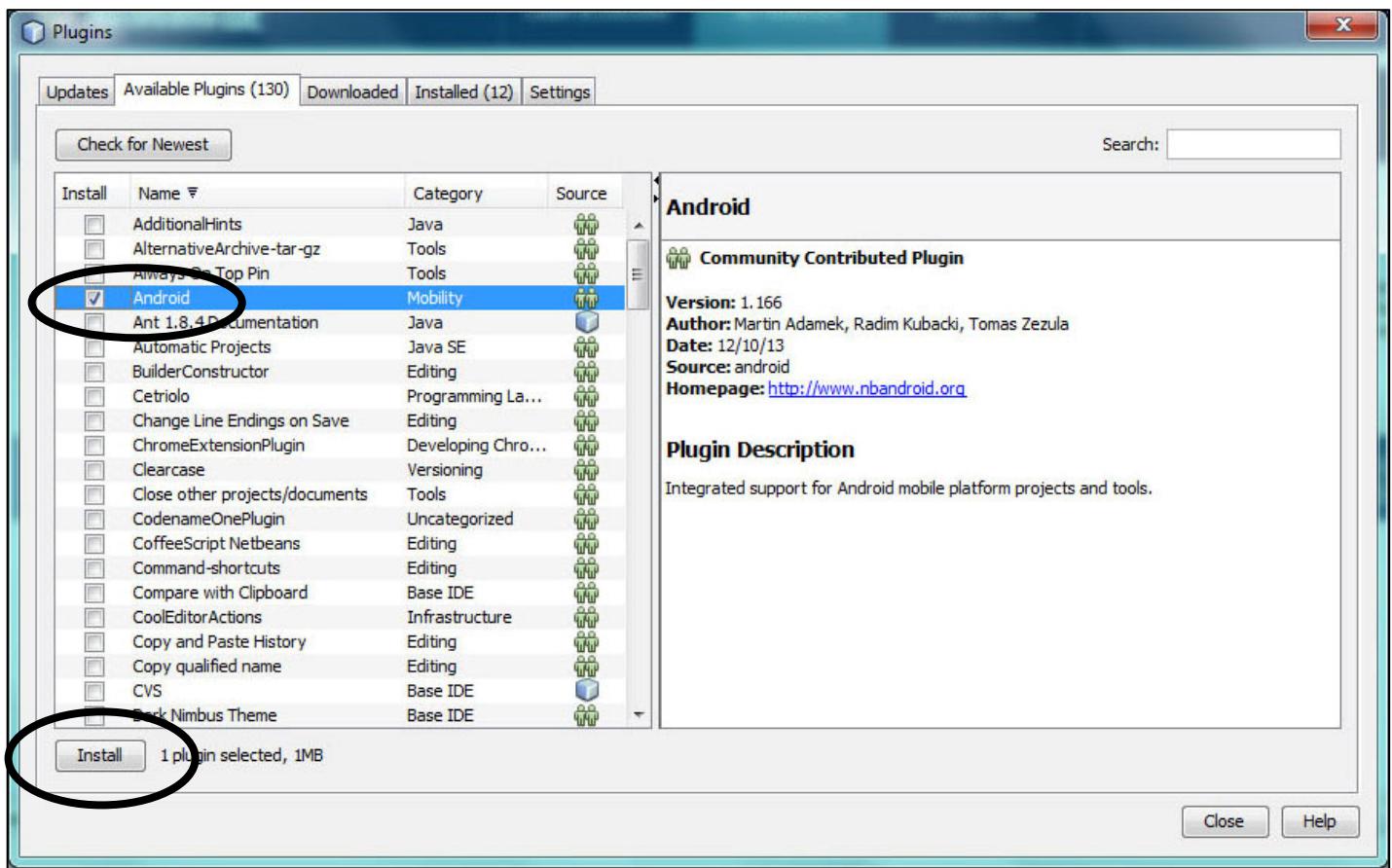
10. Le plugin Android pour NetBeans

Android Studio est clairement l'IDE de développement Android recommandé par Android.

Mais, même si ce n'est pas parfait, NetBeans peut aussi être utilisé pour développer en Android. Il existe en effet un plugin à installer dans l'IDE fétiche de Java, plugin que l'on peut trouver sur le site de NBandroid, un projet OpenSource (mais avec une extension payante). Pour installer ce plugin, il suffit d'aller dans le menu Tools-->Plugins de Netbeans et d'ajouter une nouvelle source dans l'onglet Settings :

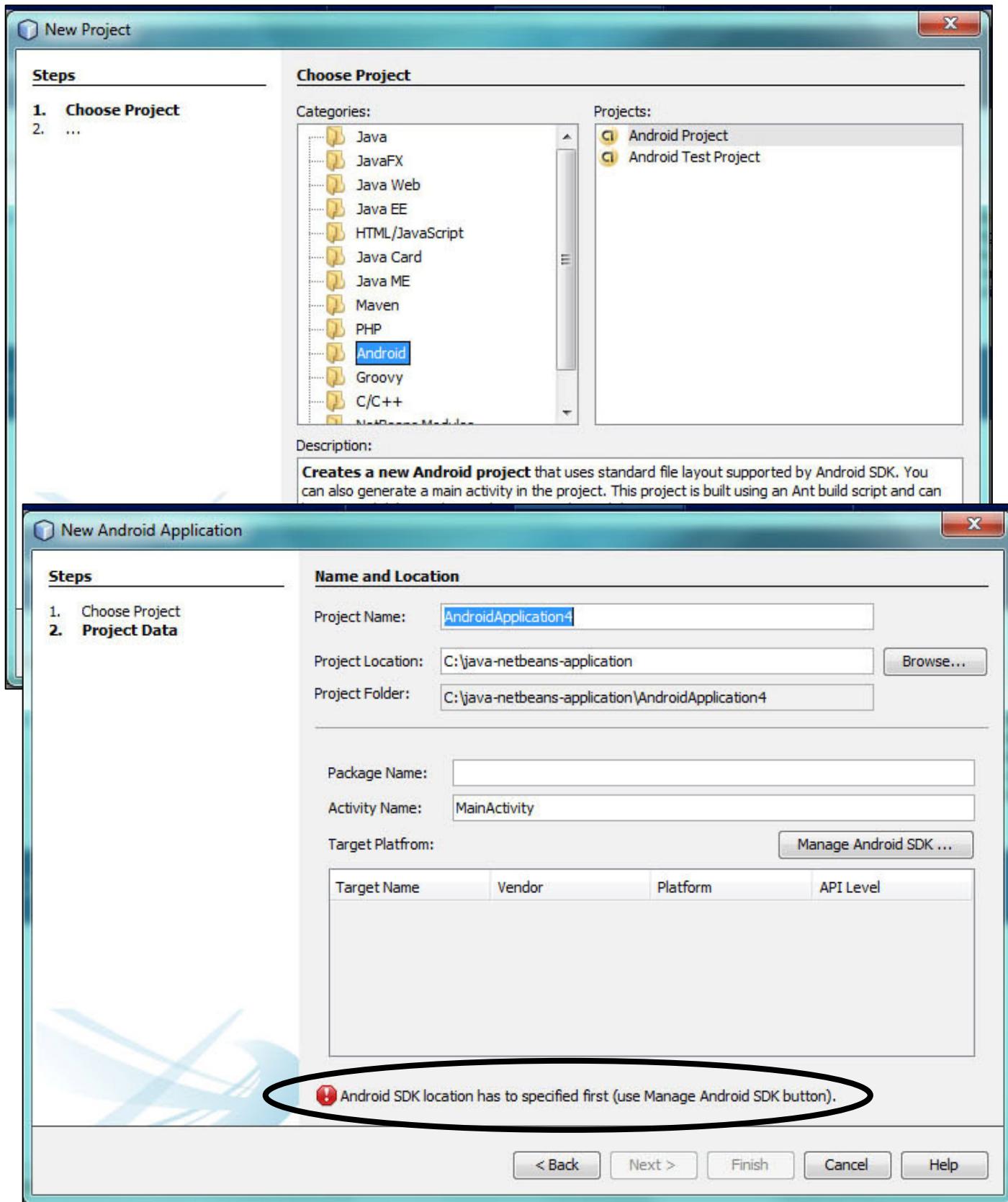


A remarquer que l'URL est bel et bien vérifiée et utilisée (il peut être nécessaire de couper temporairement le firewall). Le plugin devient alors utilisable et on peut l'installer :

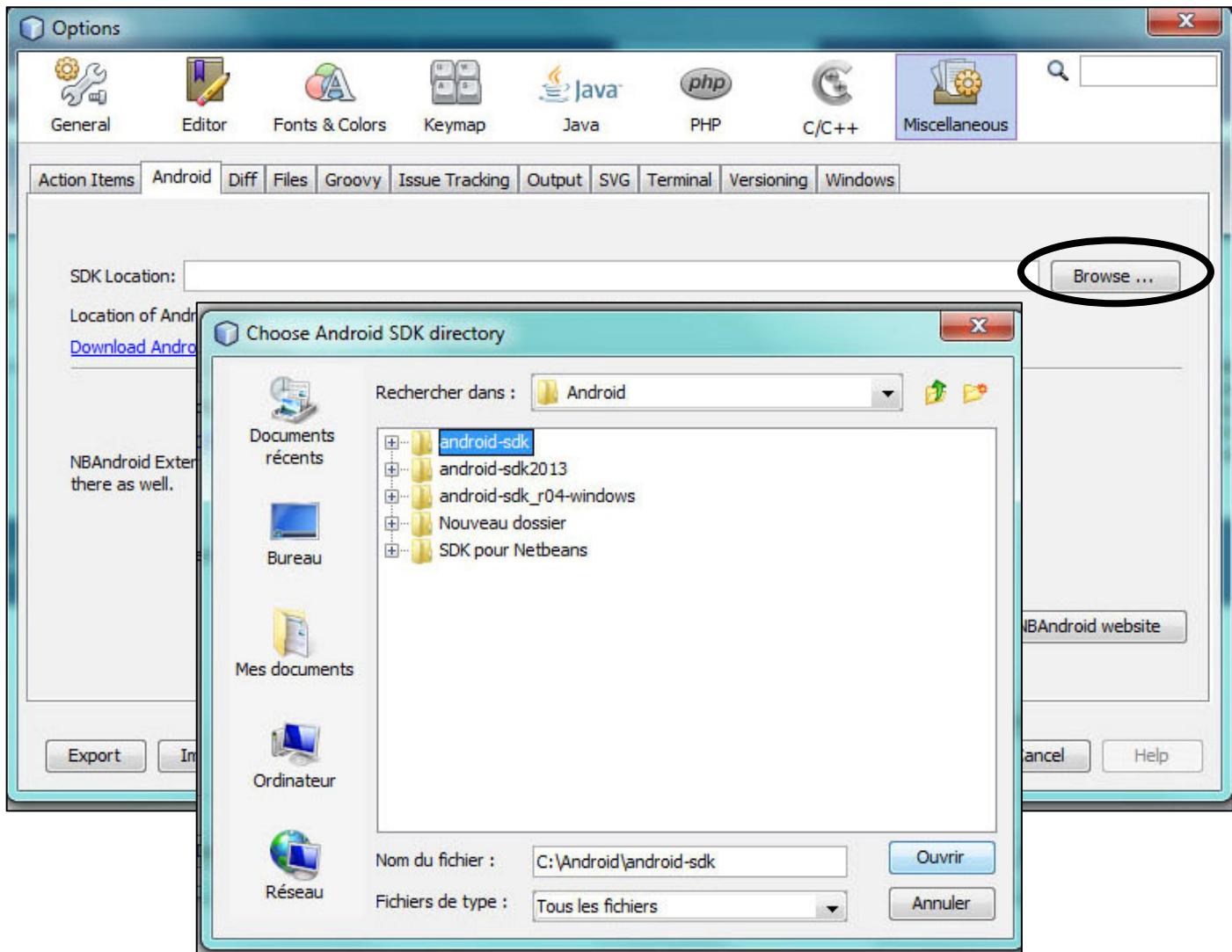


11. Crédation et développement d'un projet Netbeans

Les choses se passent comme pour un projet de Java standard JSE ou JEE :



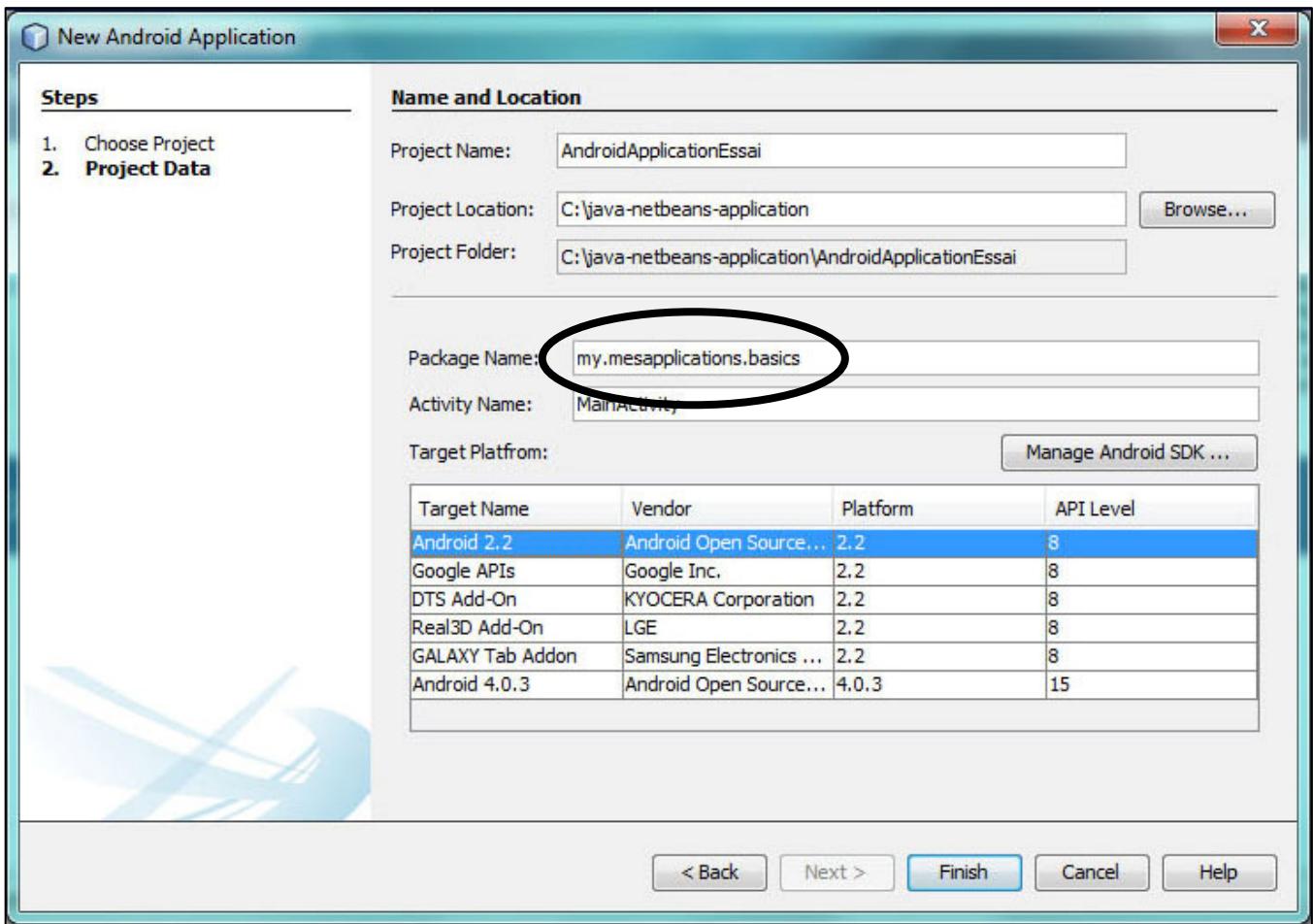
Il apparaît donc un petit problème : en fait, NetBeans ne sait pas où trouver le SDK Android et il faut donc lui indiquer où il a été déployé : un appui sur le bouton "Manage Android SDK ..." donne :



Reste alors à choisir une plate-forme dans le tableau proposé et aussi le nom du package, qui doit respecter un certain format : en effet, les packages d'une application doivent être uniques parmi toutes les autres applications que le smartphone pourrait héberger et donc, pour éviter les confusions, il est d'usage de fournir le nom du package sous la forme :

"my.NomApplication.nomPackage"

Pour nous, disons que ce sera : my.mesapplications.basics

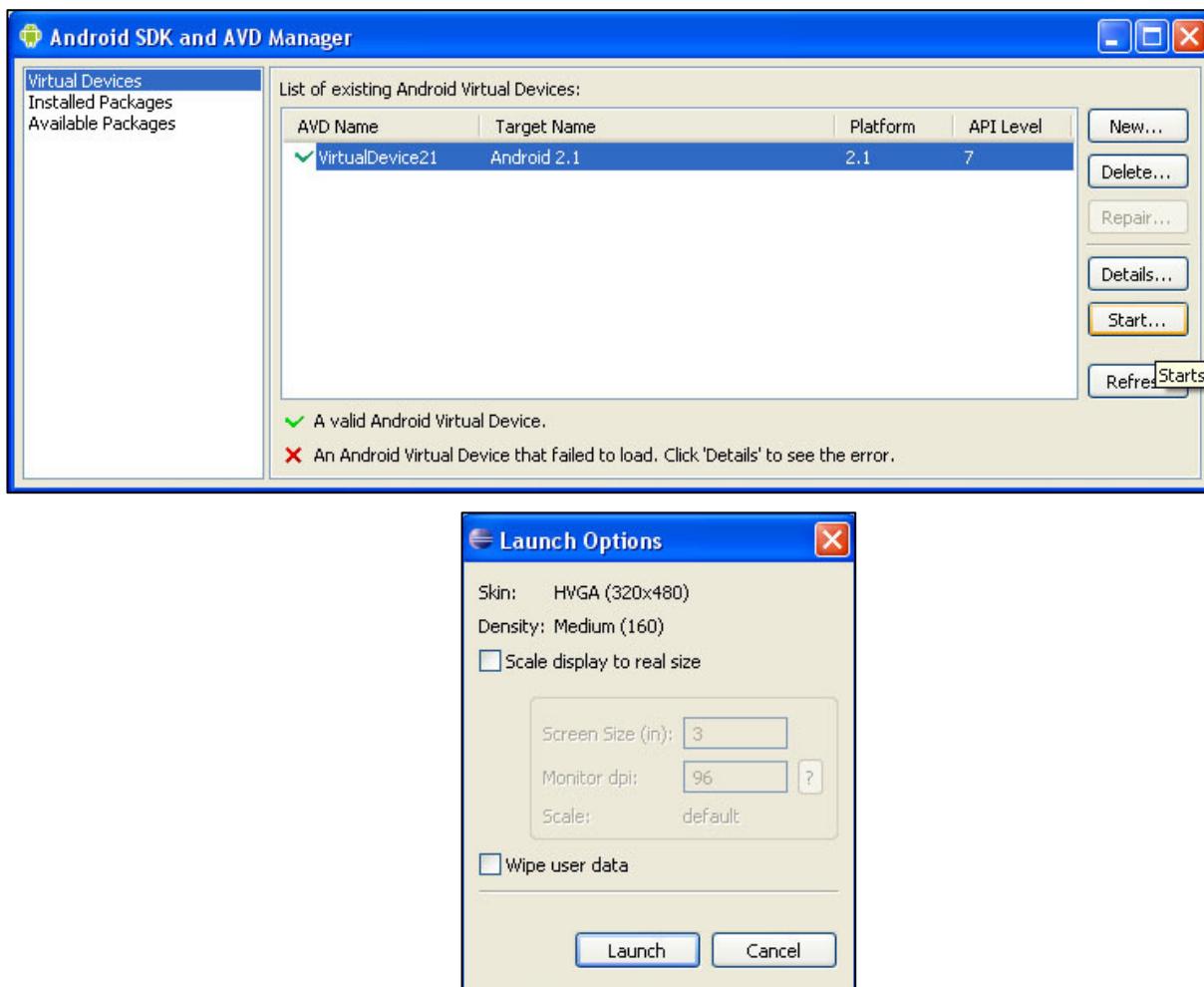


Quand on sollicite l'exécution d'une application Android déployée comme ci-dessus, on obtient :

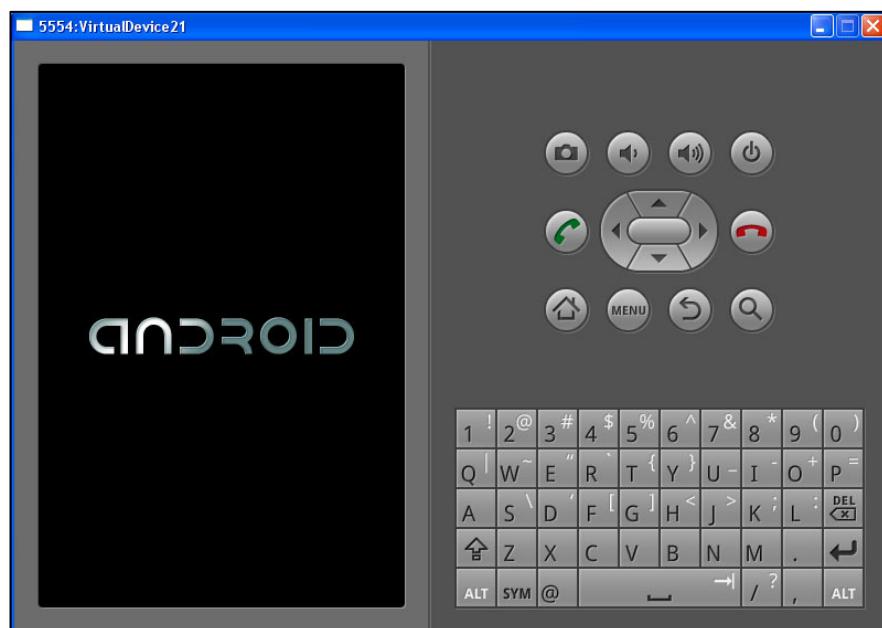
- ◆ soit le lancement de l'exécution de l'application;
- ◆ soit d'abord le manager d'AVD si on n'a jamais défini un AVD - il nous est donc alors demandé de le faire comme décrit ci-dessus :



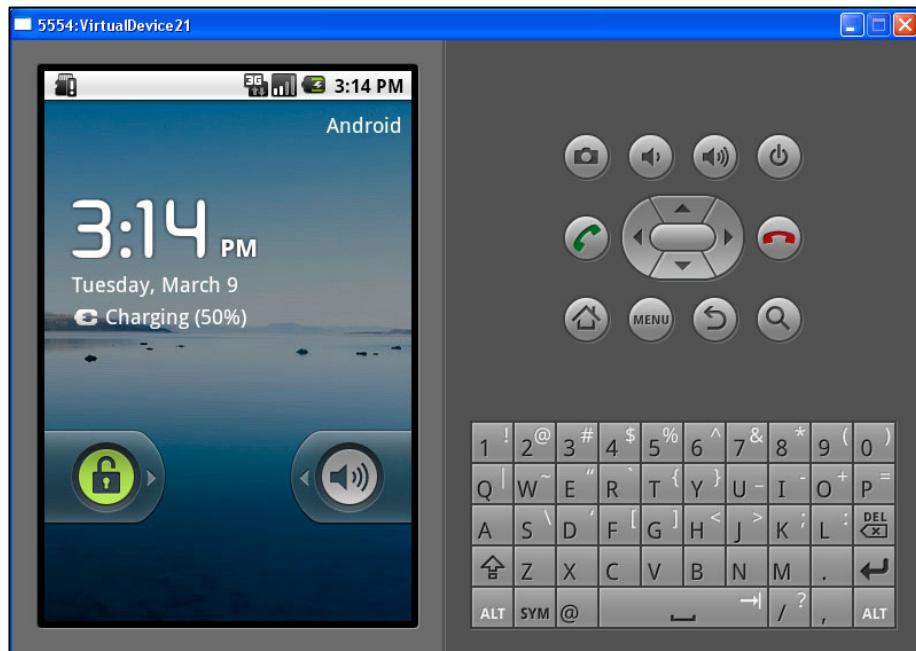
Après quoi il reste à sélectionner l'AVD à utiliser :



et enfin le même résultat que si on avait défini l'AVD au préalable, c'est-à-dire d'abord :



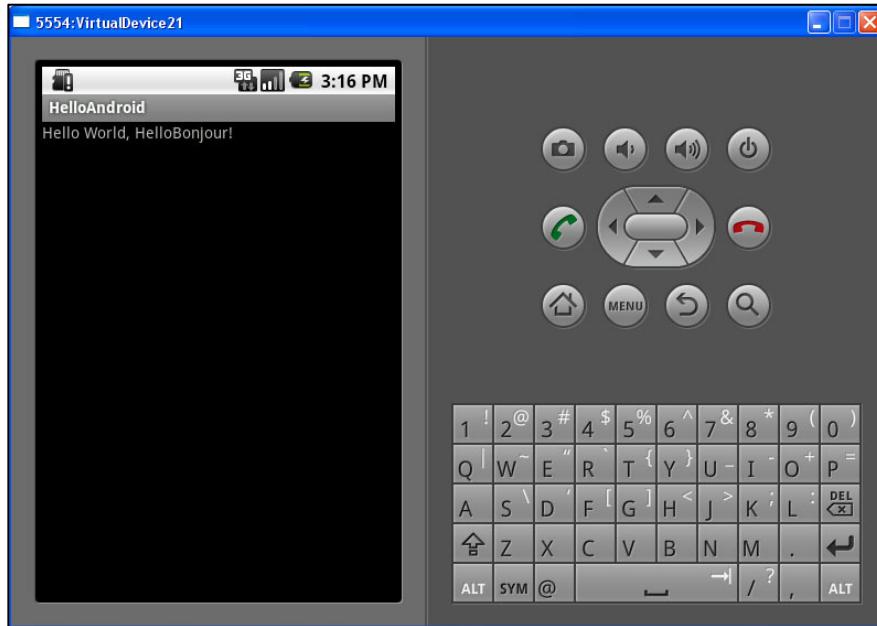
puis, après un certain temps :



- pour lancer l'application, il peut être nécessaire d'appuyer sur le bouton Menu de l'interface



pour obtenir finalement :



A remarquer que l'appui sur le bouton "Home-Maison" rend l'interface obtenu au démarrage d'Android. A remarquer aussi le verbiage dans la console :

```
[2010-04-28 17:57:04 - BonjourAndroid]-----
[2010-04-28 17:57:04 - BonjourAndroid]Android Launch!
[2010-04-28 17:57:04 - BonjourAndroid]adb is running normally.
[2010-04-28 17:57:04 - BonjourAndroid]Performing aplics.basics.HelloBonjour activity
launch
[2010-04-28 17:57:04 - BonjourAndroid]Automatic Target Mode: launching new emulator
with compatible AVD 'VirtualDevice21'
[2010-04-28 17:57:04 - BonjourAndroid]Launching a new emulator with Virtual Device
'VirtualDevice21'
[2010-04-28 17:57:06 - BonjourAndroid]New emulator found: emulator-5554
[2010-04-28 17:57:06 - BonjourAndroid]Waiting for HOME ('android.process.acore') to be
launched...
[2010-04-28 17:57:35 - BonjourAndroid]HOME is up on device 'emulator-5554'
[2010-04-28 17:57:35 - BonjourAndroid]Uploading BonjourAndroid.apk onto device
'emulator-5554'
[2010-04-28 17:57:35 - BonjourAndroid]Installing BonjourAndroid.apk...
[2010-04-28 17:57:49 - BonjourAndroid]Success!
[2010-04-28 17:57:49 - BonjourAndroid]Starting activity aplics.basics.HelloBonjour on
device
[2010-04-28 17:57:56 - BonjourAndroid]ActivityManager: Starting: Intent {
    cmp=aplics.basics/.HelloBonjour }
```

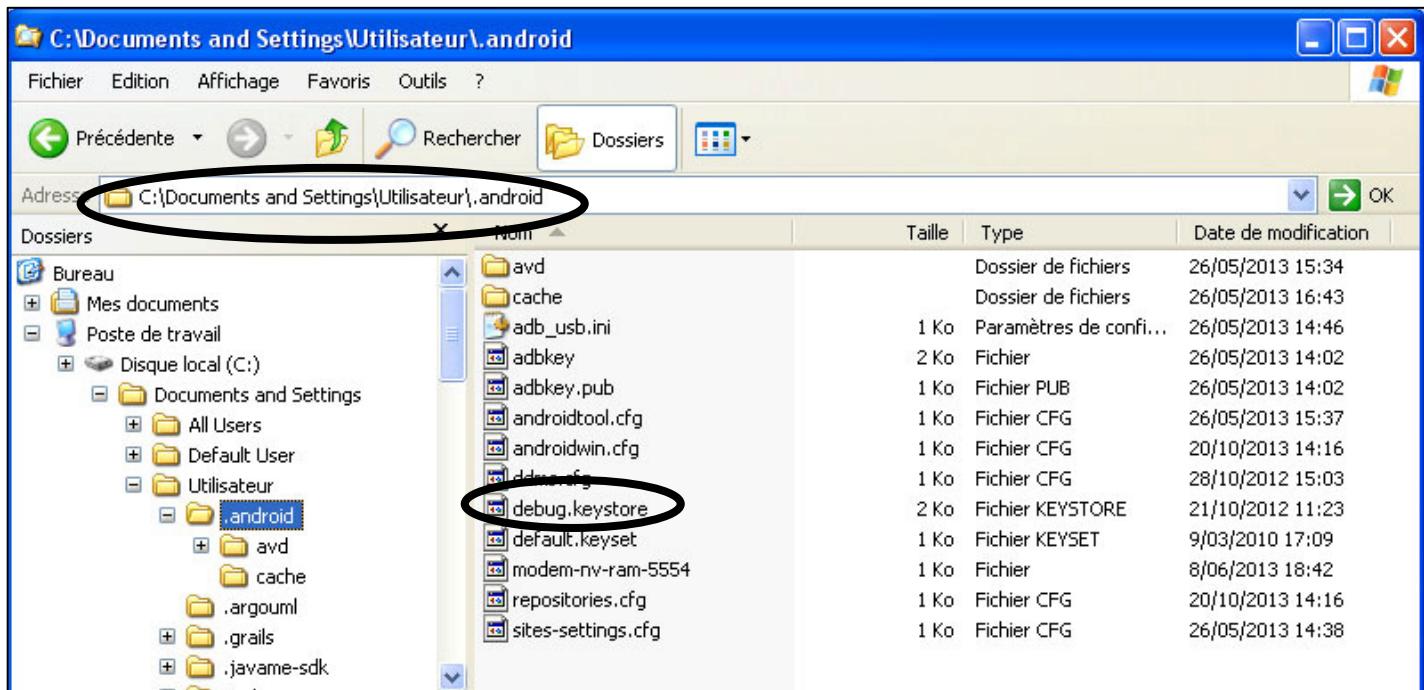
ADB (Android Debug Bridge) est un outil du SDK d'Android qui permet d'interagir sur le mobile depuis l'ordinateur de développement (voir plus loin – 11.3). On remarquera les opérations de "téléchargement" et d'"installation" sur le mobile virtuel – mais précisément, comment faire pour "vrai" mobile ?

Remarque

Le problème suivant peut survenir :

Creating AndroidApplication2-debug-unaligned.apk and signing it with a debug key...
C:\android\android-sdk\tools\ant\build.xml:955: The following error occurred while executing this line:
C:\android\android-sdk\tools\ant\build.xml:310:
com.android.sdklib.build.ApkCreationException: **Debug Certificate expired on 21/10/13 11:23**
at com.android.sdklib.build.ApkBuilder.getDebugKey(ApkBuilder.java:285)
at com.android.sdklib.build.ApkBuilder.<init>(ApkBuilder.java:392)
at com.android.ant.ApkBuliderTask.execute(ApkBuilderTask.java:334)
at org.apache.tools.ant.UnknownElement.execute(UnknownElement.java:291)
at sun.reflect.GeneratedMethodAccessor118.invoke(Unknown Source)
at ...

L'application doit en effet être signée et elle l'est ici au moyen d'une clé privée associée à une clé publique : c'est un couple de clés "de travail", de "debug". Le problème qui apparaît ici, se résout en supprimant le fichier debug.keystore – une nouvelle compilation en récréera un nouveau :



On peut visualiser le contenu de ce keystore comme pour tout keystore :

```
C:\ Invité de commandes
C:\Documents and Settings\Utilisateur\.android>keytool -list -keystore debug.keystore -v
Entrez le mot de passe du fichier de clés :

***** WARNING WARNING WARNING *****
* L'intégrité des informations stockées dans votre fichier de clés *
* n'a PAS été vérifiée. Pour cela, *
* vous devez fournir le mot de passe de votre fichier de clés.
*
***** WARNING WARNING WARNING *****

Type de fichier de clés : JKS
Fournisseur de fichier de clés : SUN

Votre fichier de clés d'accès contient 1 entrée

Nom d'alias : androiddebugkey
Date de création : 26-oct.-2013
Type d'entrée: PrivateKeyEntry
Longueur de chaîne du certificat : 1
Certificat[1]:
Propriétaire : CN=Android Debug, O=Android, C=US
Emetteur : CN=Android Debug, O=Android, C=US
Numéro de série : 67aeb16b
Valide du : Sat Oct 26 12:51:11 CEST 2013 au : Mon Oct 19 12:51:11 CEST 2043
Empreintes du certificat :
    MD5: E9:1E:59:FE:F0:5B:C6:1D:2C:88:94:C5:6E:93:FC:68
    SHA1 : C4:B5:39:DD:4B:8D:A7:5F:6B:8B:B3:EB:71:8D:D1:6D:AF:B0:07:BD
    SHA256 : 47:19:E9:53:28:2F:6F:5D:1C:D6:88:B8:1E:D2:0D:AE:E4:E6:18:F1:58
    :C6:E2:3F:AF:D7:F6:4B:2E:0B:10:B1
    Nom de l'algorithme de signature : SHA256withRSA
    Version : 3
```

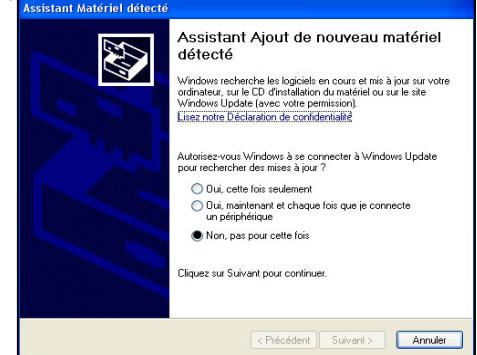
12. Installation et exécution d'une application sur un mobile

Nous allons à présent utiliser un véritable mobile qui possède le framework Android (dans notre cas, par exemple, un Samsung Galaxy). A l'heure actuelle, les systèmes d'exploitation comme Windows7 ont largement simplifié les manipulations.

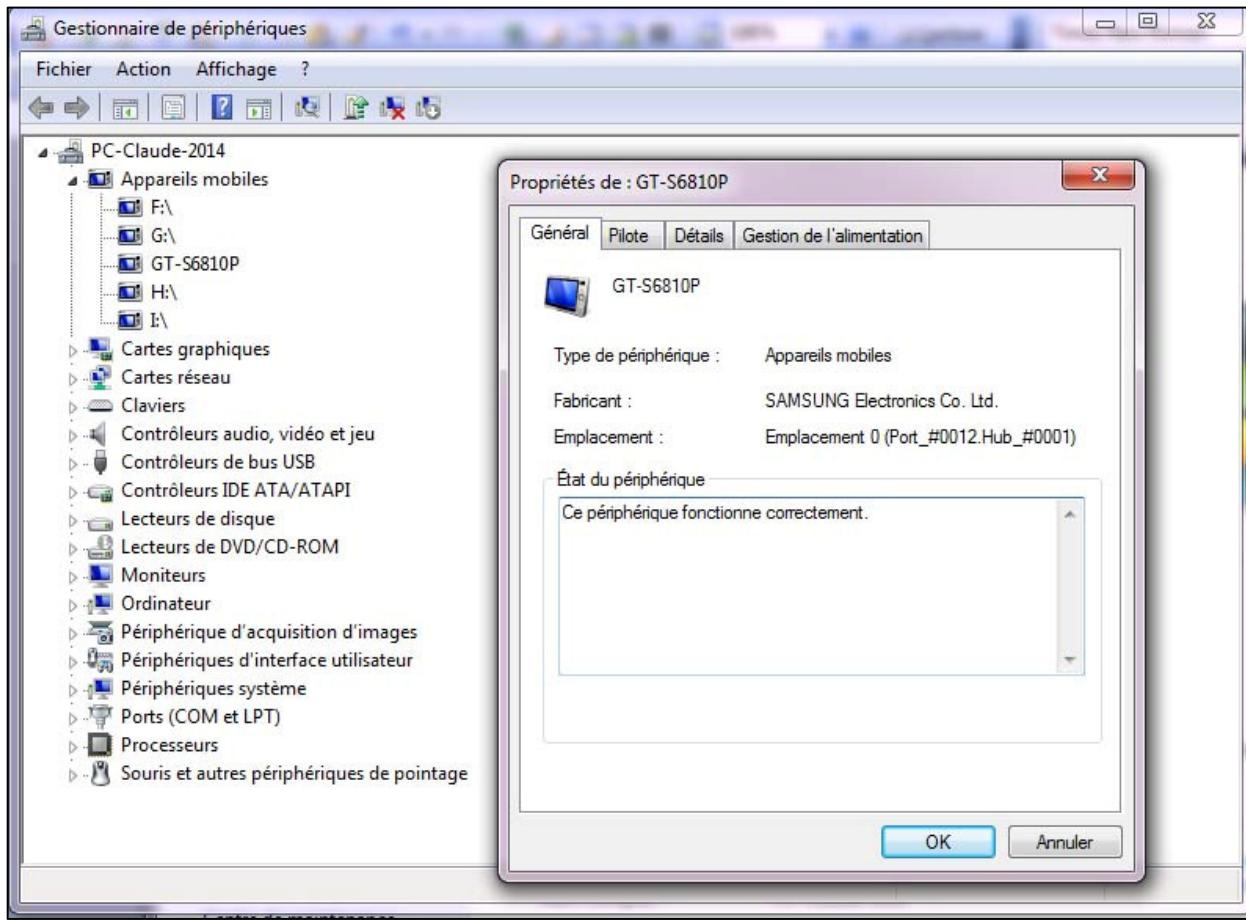
Le plus simple est de connecter le smartphone au PC de développement sur un port USB. Le mobile est alors reconnu comme un périphérique dont le driver est normalement installé automatiquement :



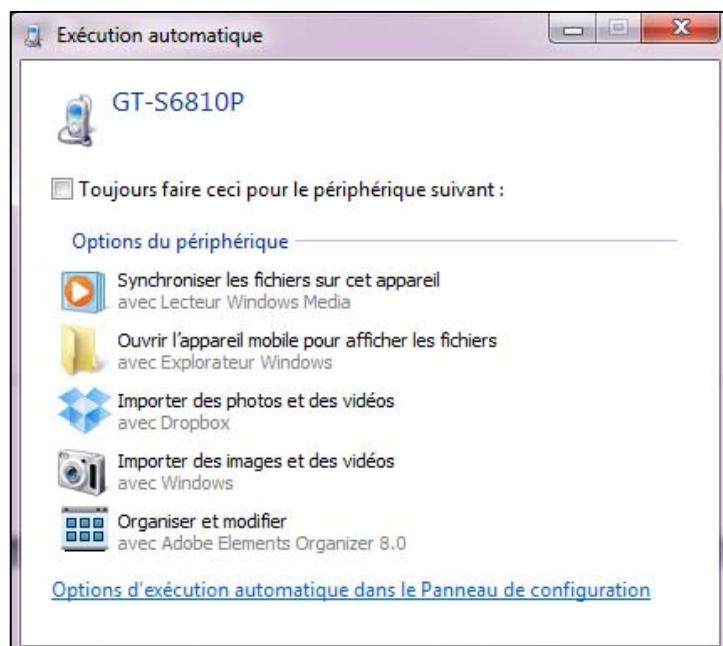
Sur des O.S. plus anciens, l'installation devra se faire manuellement :



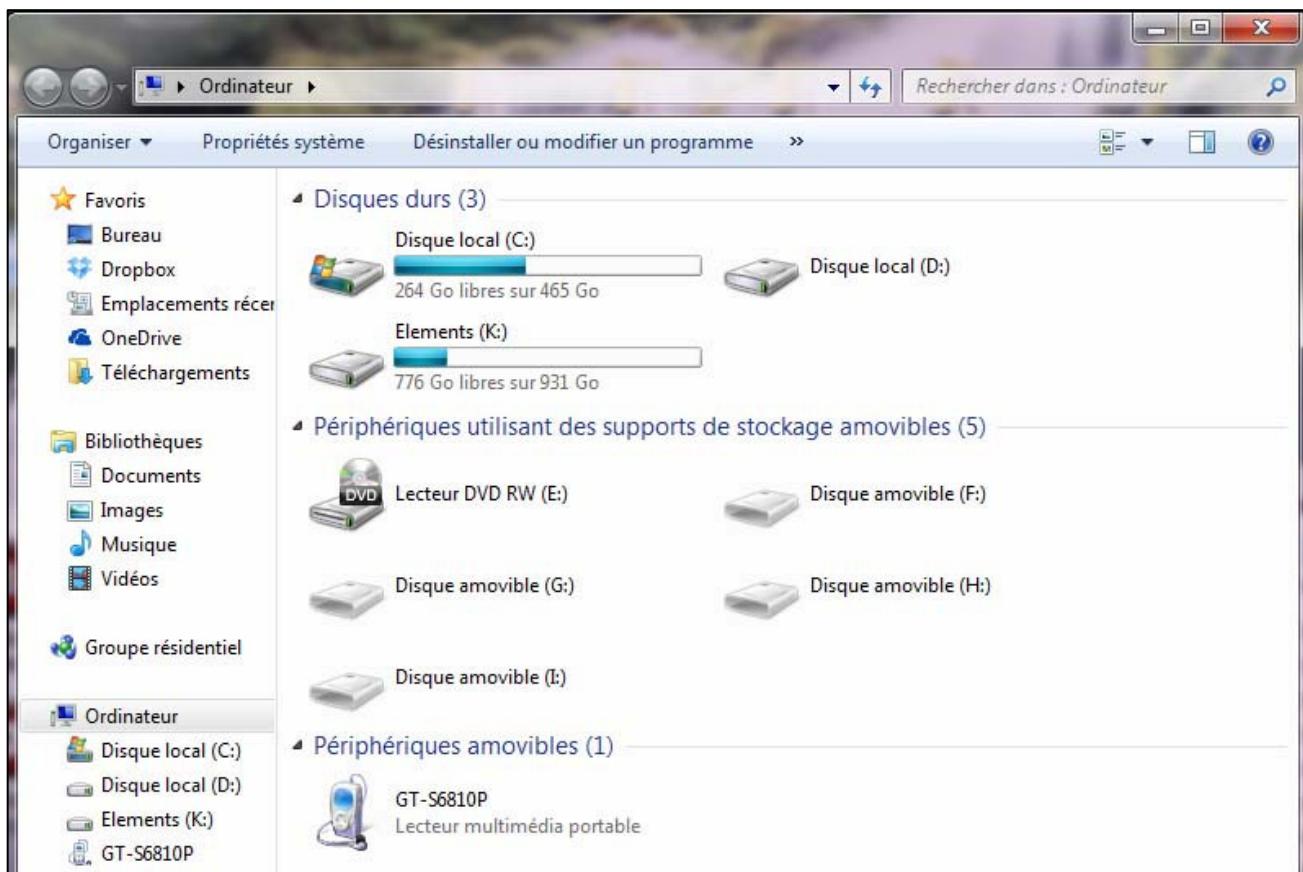
On peut vérifier par le panneau de configuration (nœud Système) que tout est en ordre :



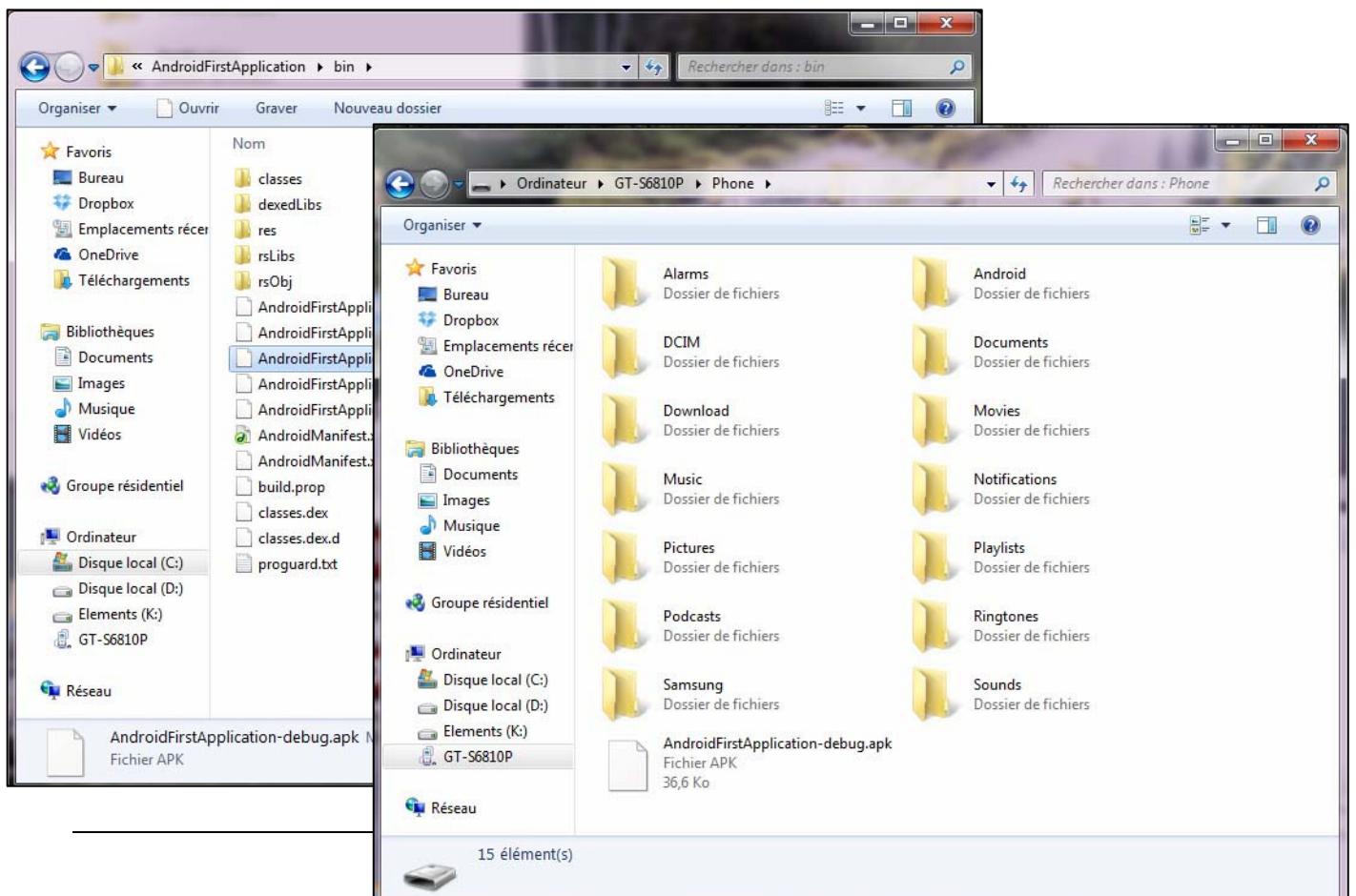
En même temps, on nous invite à poursuivre :



Notre smartphone indique qu'il est connecté en tant que périphérique et, de fait, l'explorateur de fichiers le voit comme tel :

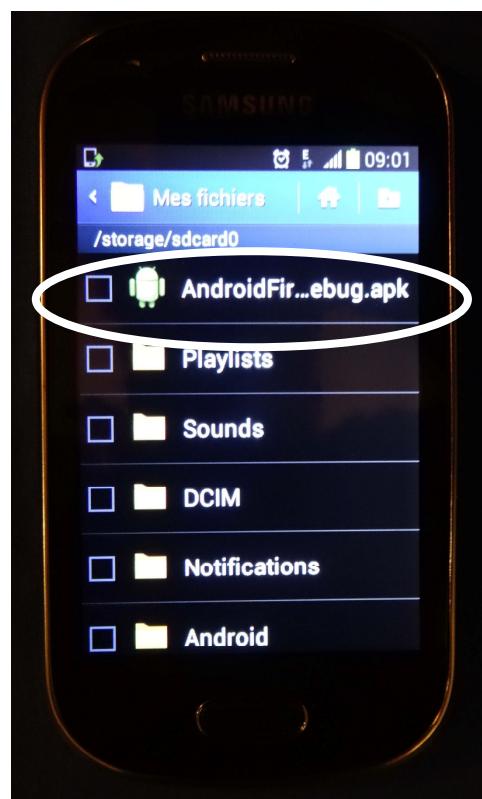


Il reste dès lors à recopier notre application dans le répertoire de notre choix sur le smartphone :

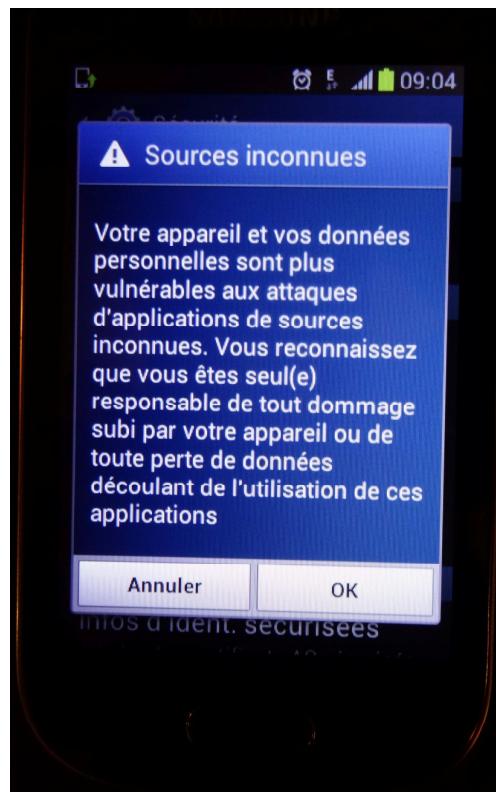
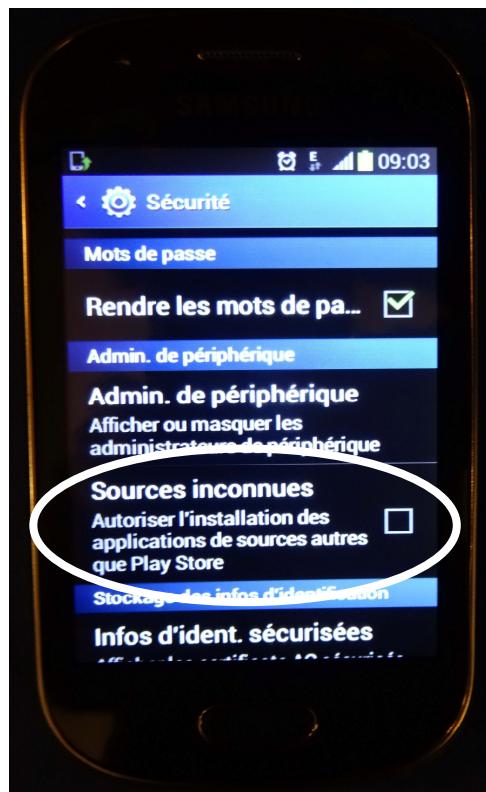


Le smartphone montre que l'application apk a bien été transférée :

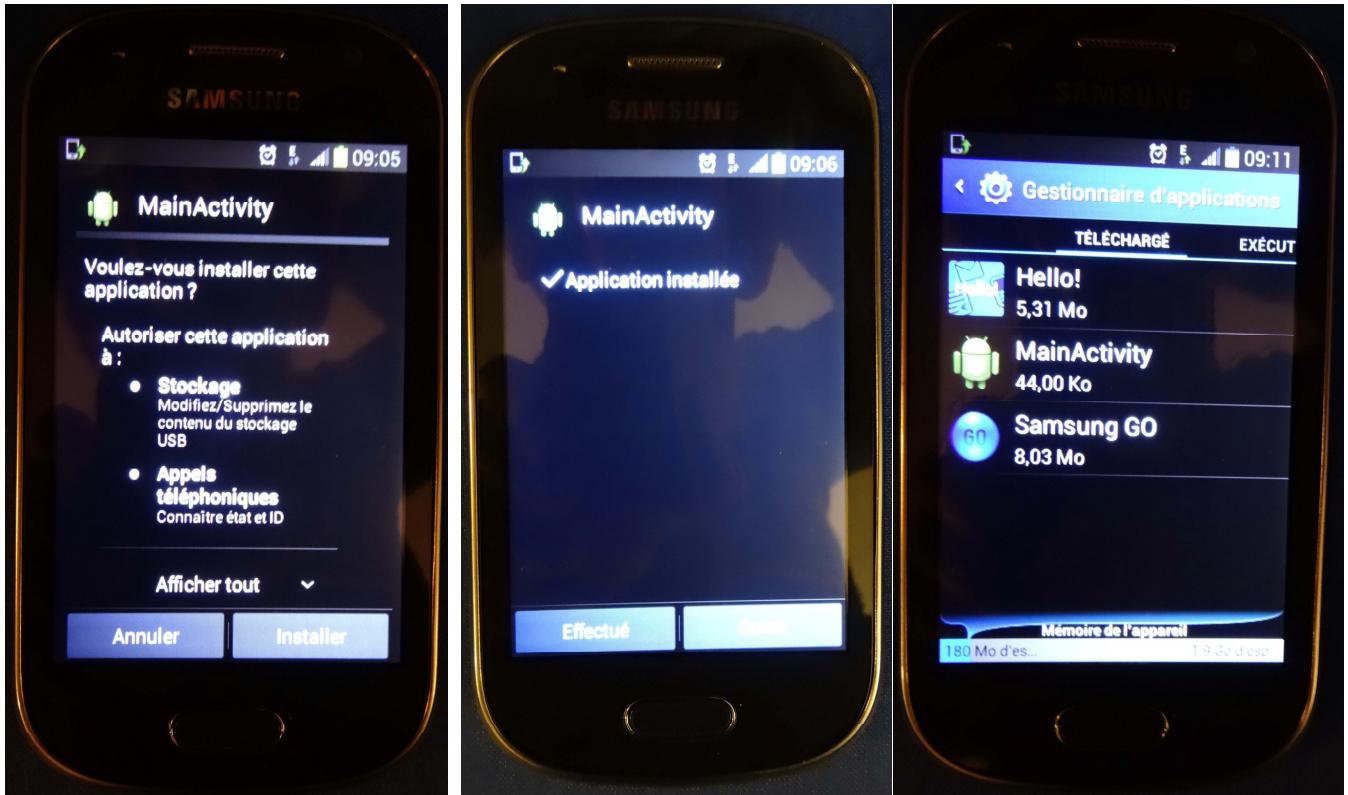
mais si on veut lancer cette application :



Nous allons modifier les paramètres nécessaires pour ne pas devoir passer par Google Play :



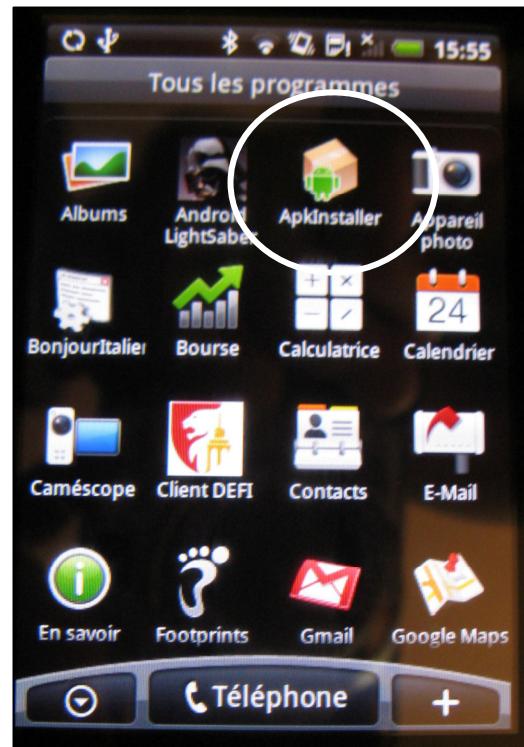
Ceci étant fait, on peut tenter de lancer l'application, ce qui lance le processus d'installation préalable, puis l'exécution :



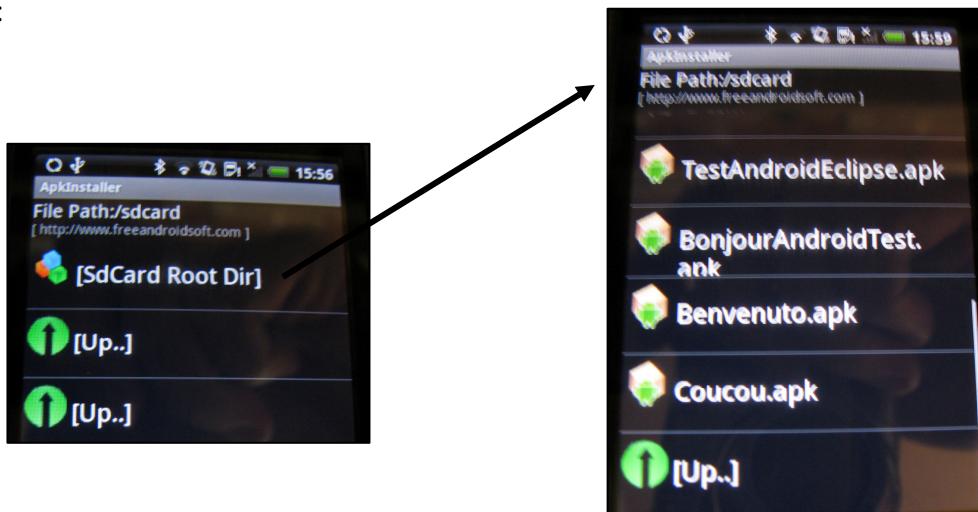
et on peut vérifier que notre application a rejoint les autres applications présentes au moyen du gestionnaire d'application.

Remarque

Dans les versions plus anciennes d'Android, il faut faire appel à une application très logiquement appelée "ApkInstaller" :



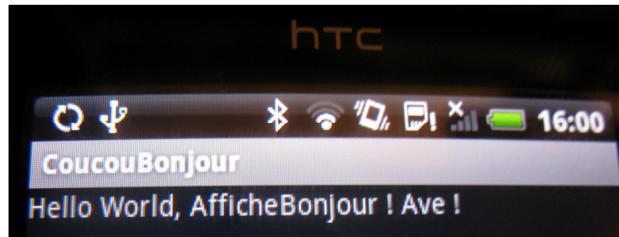
On se retrouve ainsi à choisir l'application à installer parmi celles qui se trouvent sur la carte SD :



Nous choisissons donc notre application "Coucou.apk" :



Et donc finalement ☺ :



13. L'Android Debug Bridge (ADB)

Au cours des multiples essais du développeur qui explore le monde Android, une situation classique de blocage est celle de multiples déploiements de la même application avec des signatures (automatiques ou non) différentes. Le syndrome se manifeste avec des messages Android du type suivant :

```
[2010-08-26 22:37:38 - Coucou]Android Launch!  
[2010-08-26 22:37:38 - Coucou]adb is running normally.  
[2010-08-26 22:37:38 - Coucou]Performing applics.essais.AfficheBonjour activity launch  
[2010-08-26 22:37:38 - Coucou]Automatic Target Mode: Several compatible targets. Please  
select a target device.  
[2010-08-26 22:37:42 - Coucou]Uploading Coucou.apk onto device 'HT9CXL900982'  
[2010-08-26 22:37:42 - Coucou]Installing Coucou.apk...  
[2010-08-26 22:37:43 - Coucou]Re-installation failed due to different application  
signatures.
```

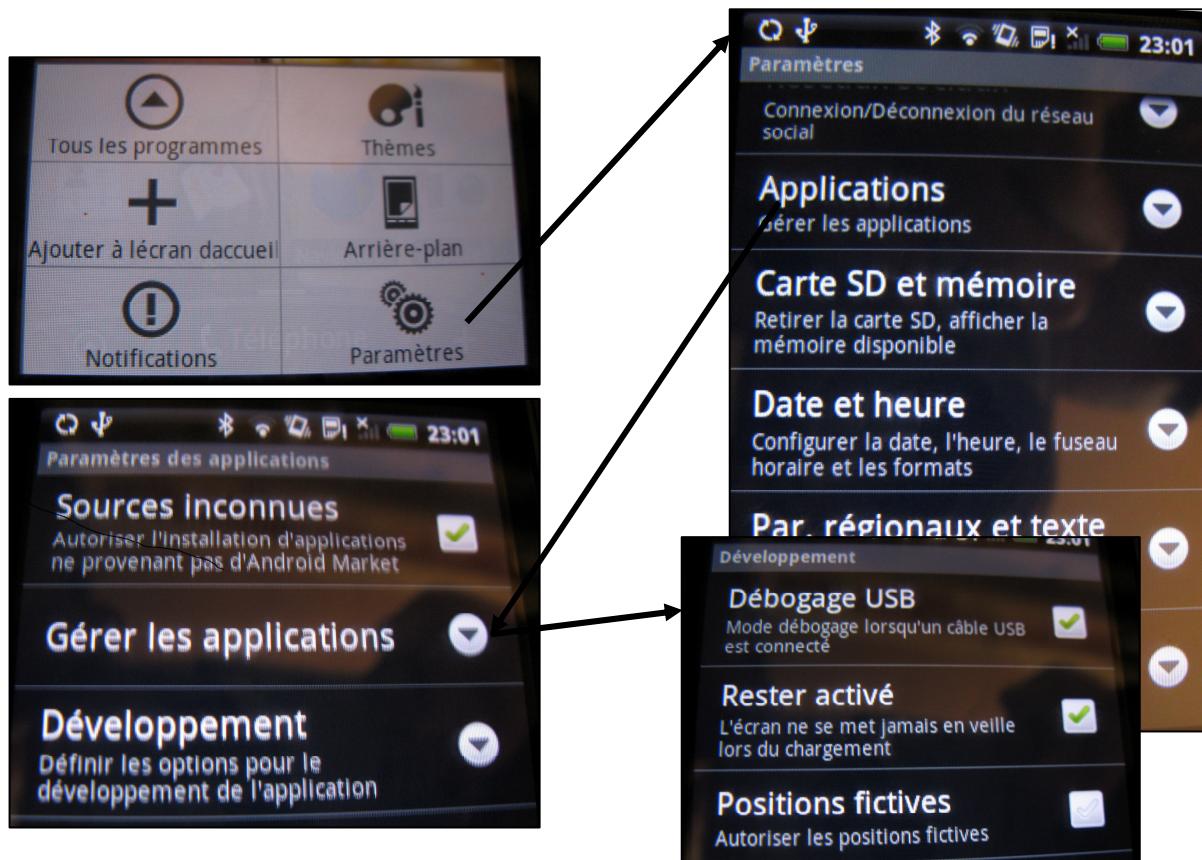
[2010-08-26 22:37:43 - Coucou] You must perform a full uninstall of the application.

WARNING: This will remove the application data!

[2010-08-26 22:37:43 - Coucou] Please execute 'adb uninstall apps.essais' in a shell.

[2010-08-26 22:37:43 - Coucou] Launch canceled!

Il faut donc interagir sur le mobile depuis l'ordinateur de développement. Ceci peut se faire avec un outil du SDK d'Android, l'**ADB** (Android Debug Bridge). Il faut cependant que le mobile soit en état de recevoir les commandes de cet outil, ce dont on peut s'assurer en vérifiant que le mode "USB debugging est activé" :



Dans ce cas, on peut alors utiliser cet outil adb depuis une fenêtre de commande DOS de l'ordinateur de développement :

```
C:\Android\android-sdk\platform-tools
```

```
C:\Android>cd android-sdk_r06-windows  
C:\Android\android-sdk_r06-windows>cd android-sdk-windows  
C:\Android\android-sdk_r06-windows\android-sdk-windows>cd tools  
C:\Android\android-sdk\platform-tools>adb shell  
error: more than one device and emulator
```

Le problème vient ici du fait que, outre le mobile physique, l'émulateur est également actif :

```
C:\Android\android-sdk\platform-tools>adb devices
List of devices attached
emulator-5554 device
HT9CXL900982 device
```

Nous pouvons donc envisager de 'nettoyer' le répertoire des packages du package qui pose des problèmes de signature :

```
C:\Android\android-sdk\platform-tools>adb uninstall applics.essais
adb server is out of date. killing...
* daemon started successfully *
- waiting for device -
error: more than one device and emulator
...
- waiting for device -
Success
```

- l'aboutissement de nos commandes étant du au fait que l'émulateur a été arrêté.

L'outil adb permet en fait de *dialoguer avec le noyau linux* au moyen d'un shell :

```
C:\Android\android-sdk\platform-tools >adb shell
$ ls
ls
sqlite_stmt_journals
cache
sdcards
etc
system
sys
sbin
proc
...
default.prop
data
root
dev

$ cd app
cd app

$ ls
ls
PDFViewer.apk
DownloadProvider.odex
DCSUtility.apk
HTCSetupWizard.apk
FilePicker.odex
CertificateService.odex
...
```

```
UploadProvider.apk
GoogleSearch.odex
HtcSettingsProvider.apk
CustomizationSettingsProvider.odex
...
WorldClock.apk
WeatherProvider.apk
...
PackageInstaller.odex
Maps.apk
...
WeatherProvider.odex
...
PDFViewer.odex
...
Weather.odex
Launcher.apk
...
HTMLViewer.apk
Browser.odex
...
$ exit
C:\Android\android-sdk\platform-tools >
```

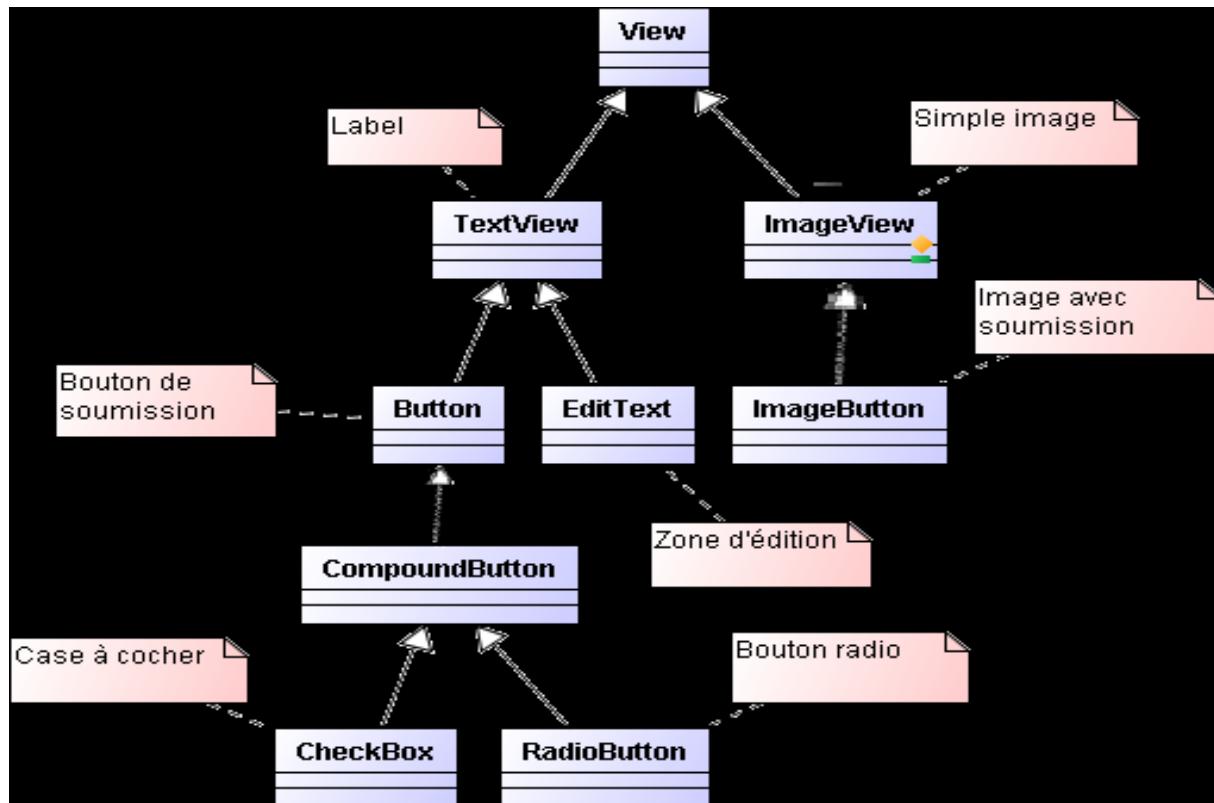
Les fichiers 'odex' sont en fait des fichiers 'dex' optimisés (**Optimized DEX** files). L'objectif est évidemment que les fichiers apk qui les utilisent sont plus rapides – mais ils dépendent alors de la présence de ces fichiers optimisés.

Nous avons vu apparaître furtivement dans les tags XML des termes tels que "TextView" ou "LinearLayout" : de quoi s'agit-il vraiment ?

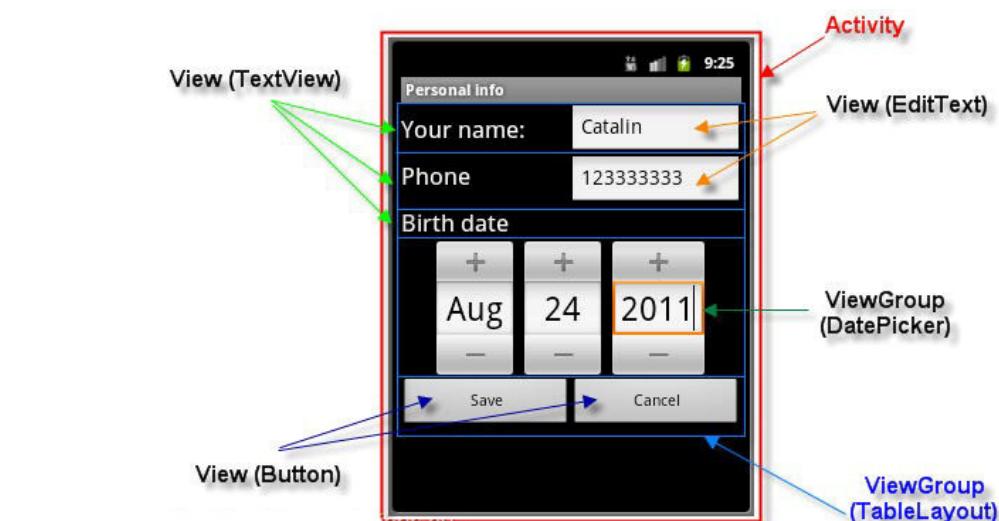
14. Les interfaces et les événements graphiques

14.1 View et ViewGroup

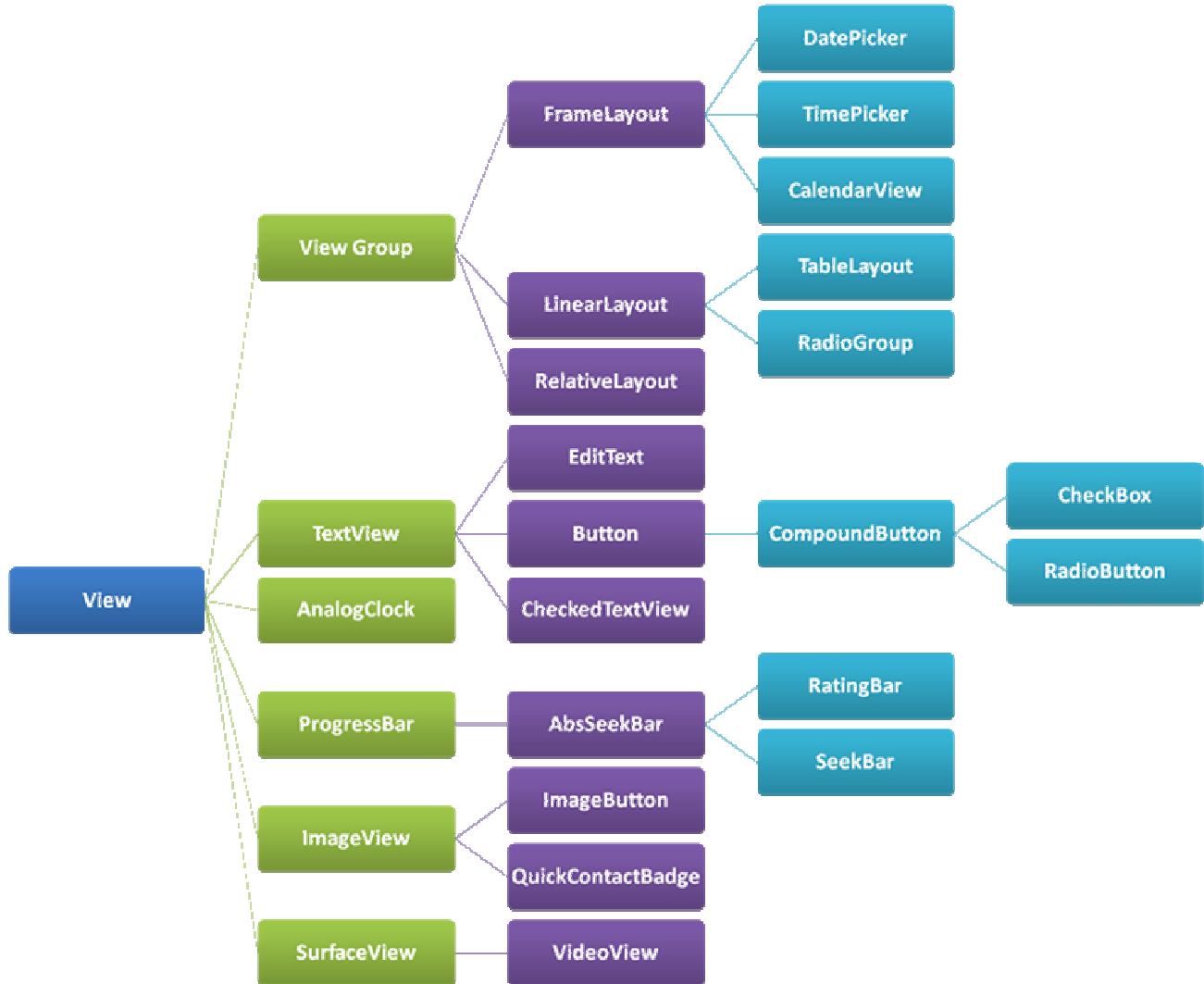
Tous les composants graphiques d'Android héritent de la classe **View** (package android.view). Cette dernière sert de base aux sous-classes appelées **widgets**, qui sont des éléments de l'interface graphique (GUI) prêts à être utilisés comme les champs de texte, les boutons ou cases à cocher, etc. De manière simplifiée :



La classe **ViewGroup** sert de base aux sous-classes appelées **Layouts**, qui offrent différentes sortes de structure de page comme linéaire, tabulaire et relative. L'interface utilisateur se définit donc en utilisant un arbre d'objets **View** et **ViewGroup** : les composants s'imbriquent les uns dans les autres selon une arborescence hiérarchique. Les objets de héritant de **View** sont les feuilles de l'arbre, ceux héritant de **ViewGroup** sont les branches. Typiquement, un GUI est donc comme sur cet exemple :



La hiérarchie des composants graphiques est donc plus riche qu'imaginé (schéma fourni, comme l'image précédente, par <http://www.itcsolutions.eu/2011/08/27/android-tutorial-4-procedural-vs-declarative-design-of-user-interfaces/>) :



Chaque ViewGroup doit demander à chacun de ses enfants de s'afficher (ceux-ci peuvent demander une taille et une position à leur parent, mais c'est ce dernier qui détient la décision finale concernant la position et la taille de chacun de ses enfants).

14.2 Déclaration et propriétés des composants d'un GUI

Comme nous l'avons déjà vu, bien qu'il soit théoriquement possible de définir un GUI par programmation Java pure, il est pratique d'utiliser pour cela des déclarations XML du type :

```
<element attribut1="valeur" attribut2="valeur"> ... </element>
```

Chaque Vue Android peut disposer d'un identifiant défini grâce à l'attribut **android:id**. La convention consiste à utiliser le format

@+id/nom_unique

où **nom_unique** représente le nom de l'objet à utiliser dans le code Java. En fait :

- ◆ @ : indique au système que la ressource pointée par l'identifiant n'est pas définie dans le fichier Java courant, mais dans un fichier externe;
- ◆ + : indiquer à Android que l'identifiant peut être ajouté à la liste des identifiants de l'application si cela n'est pas déjà fait;
- ◆ id/ : combiné avec le "@", ce préfixe oblige Android à regarder dans la liste des identifiant déjà déclarés (par +);
- ◆ nom_unique: nom (unique) de l'identifiant.

Chaque type de vue possède des attributs spécifiques, et d'autres communs à tous les widgets (mais aussi aux layouts).

Parmi les propriétés communes, **layout_width** et **layout_height** sont obligatoires et existent pour toutes les vues. Ces deux propriétés peuvent prendre trois types de valeur : taille fixe (50px), **fill_parent** (le composant prend automatiquement la même taille que son conteneur parent après le placement des autres widgets) et **wrap_content** (le composant prend la taille de son contenu et la vue ne prend que la place qui lui est juste nécessaire). A noter que **fill_parent** est à présent remplacé dans les versions récentes par **match_parent**.

Parmi les propriétés propres aux widgets, on trouve **layout_x** et **layout_y** – par exemple :

```
<TextView  
    android:id="@+id/widget33"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:background="@color/yellow"  
    android:text="Bonjour jeune patricien !"  
    android:textColor="@color/black"  
    android:layout_x="66dp"  
    android:layout_y="14dp" />  
  
<TextView  
    android:id="@+id/widget35"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Toi aussi tu veux du data mining ?"  
    android:layout_x="41dp"  
    android:layout_y="38dp" />
```

L'unité classique des tailles des composants est le **dp** ou **dip** (Density independent Pixel) : elle se base sur la densité de pixels d'un écran pour définir la taille du composant considéré et correspond plus ou moins à 1 pixel sur un écran de 160 dpi (Dot Per Inch = pixel par pouce). Si px désigne la taille en pixels et dpi la taille d'une image, alors :

$$\text{px} = \text{dp} * \text{dpi} / 160$$

ou

$$\text{dp} = \text{px} * 160 / \text{dpi}$$

On parle encore d'"unité virtuelle" car elle n'est pas mesurable directement, mais plutôt concrétisée sur un matériel de densité donnée. L'intérêt : en fonction de la densité de l'écran

sur lequel s'exécute effectivement l'application, le système appliquera les transformations nécessaires pour augmenter ou diminuer la taille effective d'1 dp.

Jusqu'à Android 3.2, une possibilité envisageable est de choisir pour l'écran l'une des 3 densités prédéfinies ldpi (120 dpi), mdpi (160 dpi) et hdpi (240 dpi), éventuellement complétées par xhdpi (320 dpi) ou xxhdpi (480 dpi). Une fois cette densité choisie, elle sera appliquée partout, quelle que soit la véritable densité du téléphone. Un designer pourra donc faire une image, et au pire, voire cette image avec des pixels doublés, ou divisés par deux. Bien sûr, le designer peut fournir une version pour chacune des densités et ainsi être sûr qu'un pixel d'un bitmap correspondra toujours à un pixel à l'écran, quel que soit le terminal.

La classe base View comporte à elle seule une armée d'attributs possibles. Citons :

- ◆ android:clickable || setClickable(boolean) : l'élément est réactif à un clic de souris;
- ◆ android:minHeight / android:minWidth: hauteur/largeur minimale du composant;
- ◆ android:onClic : spécification de la méthode à appeler automatiquement lors d'un clic de la souris sur le composant.

Mais nous allons y revenir ...

14.3 Les composants spécifiques

Bien sûr, chaque composant graphique, matérialisé par une par une classe héritée de View, possède ses propres caractéristiques. Sans être exhaustif, on peut mentionner (package android.widget) :

1) **TextView** : "label", le composant graphique le plus simple. Parmi ces très nombreux attributs, citons :

- ◆ android:autoLink || setAutoLinkMask(int) : contrôle si le texte correspond à un lien quelconque (adresse email ou URL) qui, le cas échéant, est converti en élément cliquable;
- ◆ android:autoText || setKeyListener(KeyListener) : contrôle si le champ doit fournir une correction automatique de l'orthographe – il existe de nombreuses autres utilisations d'un KeyListener (nombres, majuscules, etc)
- ◆ android:editable : permet de définir cet élément comme champ de saisie.
- ◆ android:hint || setHint(int) : propose un texte par défaut lorsque le champ est vierge.
- ◆ android:maxLength || setMaxLength(int) : définit la longueur maximale du texte en terme de nombre de caractères.

On peut donc créer un TextView soit par XML :

```
<TextView ...      .../>
```

soit par programmation

```
TextView tv = new TextView(context);
tv.setText("Bonjour");
tv.setTextColor(0x112233);
```

2) **Button** : le "bouton" classique, hérité de TextView

3) **ImageView et ImageButton** : équivalent en image de TextView et Button. L'attribut android:src permettant de préciser l'image utilisée (une ressource graphique, un "drawable").

4) **TextView** : classe dérivée de TextView, cette vue est l'habituel champ de saisie. On s'en doute, une multitude d'attributs permet de contrôler la saisie comme android:password.

A partir d'Android 1.5, la plupart des modes de saisie ont été regroupés dans un seul attribut **android:inputType** dont la valeur est composée d'une classe et de modificateurs, séparés par le caractère "!" : la classe décrit généralement ce que l'utilisateur est autorisé à saisir et détermine l'ensemble de base des touches disponibles sur le clavier logiciel. Les classes possibles sont text (par défaut), number, phone, date, time, etc. Afin de préciser ce que l'utilisateur pourra saisir, on peut ensuite enrichir les classes par un ou plusieurs modificateurs (exemple : text|textCapCharacters précise que la saisie attendue est de type texte et que ce dernier sera entièrement transformé en lettres majuscules).

5) **CheckBox et RadioButton** : les habituelles classes soeurs, dérivées de CompoundButton ; bien sûr, la classe RadioGroup peut guetter un changement d'état avec

```
public void setOnCheckedChangeListener(RadioGroup.OnCheckedChangeListener listener)
```

14.4 Les gestionnaires de mise en page

Les classes dérivées de ViewGroup sont les containers-gestionnaires de mises en page, analogues à ceux que l'on connaît en AWT/Swing : ce sont donc des "layouts" (" gabarits" en français). Citons :

- ◆ LinearLayout : toutes les vues sont placées les unes à la suite des autres - ce layout aligne tous ses enfants verticalement ou horizontalement.
- ◆ RelativeLayout : les vues sont positionnées les unes par rapport aux autres; une série d'attributs est disponible pour chaque composant (View/ViewGroup) afin de spécifier la relation avec les composants voisins (tel que android:layout_below pour spécifier l'id du composant au-dessus, android:layout_above pour spécifier l'ID du composant en-dessous, etc).
- ◆ TableLayout : le tableau en lignes-colonnes; chaque rangée est représentée par un TableRow qui peut contenir 0 à n View.
- ◆ FrameLayout : permet de réserver un espace à l'écran pour afficher une vue unique;
- ◆ AbsoluteLayout : on a compris, mais c'est rarement une bonne idée.

On peut ajouter comme outil de disposition :

- ◆ GridView : les éléments sont vus comme placés sur une grille "scrollable" dont le maillage sera l'unité; chaque élément va alors recevoir une dimension représentée par une quantité de colonne et de rangées qu'elle va occuper sur cette grille (pour cela, on utilise les attributs android:layout_rowCount et android:columnCount), les recouvrements étant spécifiés au moyen d'attributs android:layout_columnSpan et android:layout_rowSpan;
- ◆ ListView : affiche une liste d'éléments scrollable.
- ◆ ScrollView : permet le défilement d'un nombre important de vues.

Un certain nombre d'attributs (parmi d'autres) sont incontournables pour ces gestionnaires :

- ◆ alignment vertical ou horizontal : android:orientation="vertical" ou "horizontal";

- ◆ gravité, c'est-à-dire importance relative de chaque composant : android:layout_weight (en %)
- ◆ positionnement relatif : android:layout_alignParentTop, android:layout_alignParentBottom, android:layout_alignParentLeft, android:layout_alignParentRight, android:layout_centerHorizontal, android:layout_centerVertical, android:layout_centerInParent.

14.5 La gestion des événements graphiques

Il existe plusieurs manières de gérer les événements graphiques. Pour les décrire, considérons un banal bouton poussoir de type "Ok" (accompagné de son contraire "Annuler" et d'une 3^{ème} bouton "Laisser tomber"), décrit par :

main.xml (onglet layout)

```
<?xml version="1.0" encoding="utf-8"?>
<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/bouton_ok"
        android:text="OK"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content">
    </Button>
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Annuler"
        android:id="@+id/bouton_annuler">
    </Button>
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/bouton_assez"
        android:text="Laisser tomber">
    </Button>
</LinearLayout>
```

a) gestion au moyen d'un attribut

On peut utiliser l'attribut "android:onClick" :

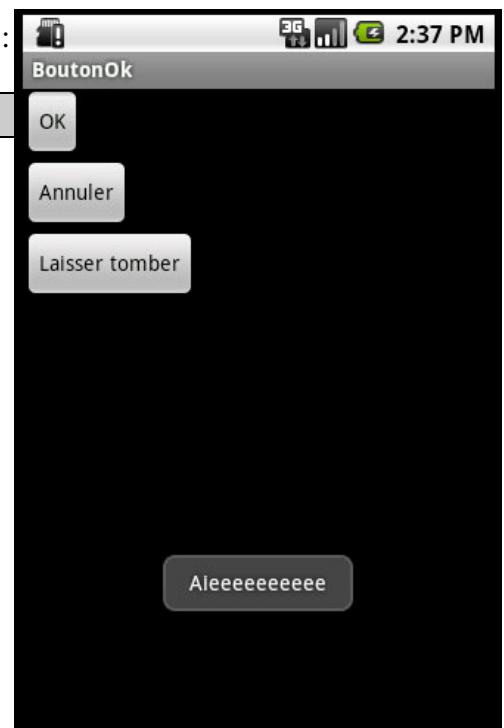
main.xml (onglet layout)

```
<?xml version="1.0" encoding="utf-8"?>
...
<Button android:id="@+id/bouton_ok"
    android:text="OK"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:onClick="methode_ok">
</Button>
...
```

La méthode en question est définie simplement dans l'activité :

OkHandler

```
package basics.gui;  
import android.app.Activity;  
import android.os.Bundle; ...  
  
public class OkHandler extends Activity  
{  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState)  
    {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
    }  
  
    public void methode_ok(View v)  
    {  
        String msg = "Aieeeeeeeeeee";  
        Toast.makeText(this,msg,Toast.LENGTH_LONG).show();  
    }  
}
```



la classe **Toast** (package android.widget) étant une simple vue destinée à délivrer un petit message à l'utilisateur. Sa méthode

public static Toast **makeText** (Context context, int resId, int duration)

construit un Toast standard avec une simple TextView

pendant un temps long ou court (configurable) fixé par :

public static final int LENGTH_LONG

public static final int LENGTH_SHORT

b) gestion au moyen d'un listener explicite

Dans la tradition classique des gestions d'événements à la Java, on utilise un listener, ici **OnClickListener** (package android.view.View) déclarant la seule méthode :

public abstract void **onClick** (View v)

Et bien sûr, et toujours dans la plus pure tradition Java, il faut encore inscrire la vue dans la mailing-list du bouton au moyen de la méthode :

public void **setOnClickListener** (View.OnClickListener l), ici, pour le bouton "Annuler" :

OkHandler (2)

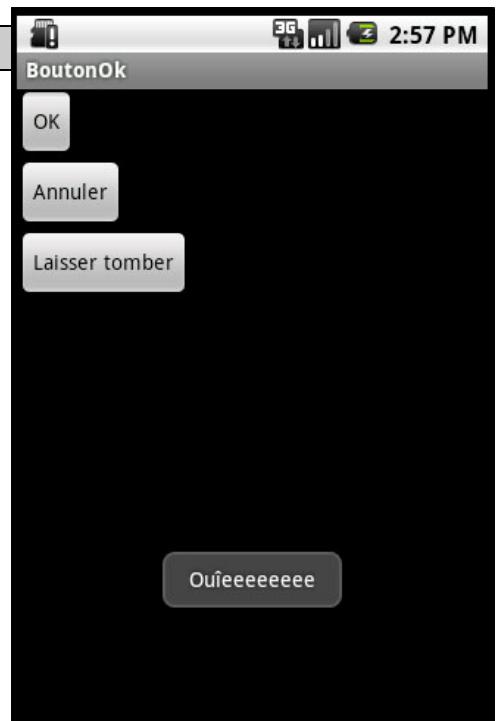
```
package basics.gui;

import android.app.Activity; ...

public class OkHandler extends Activity
implements OnClickListener
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

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

    @Override
    public void onClick(View v)
    {
        if (v == this.findViewById(R.id.bouton_annuler))
        {
            String msg = "Ouïeeeeeeee";
            Toast.makeText(this, msg, Toast.LENGTH_LONG).show();
        }
    }
}
```



la méthode

`public final View findViewById (int id)`

permettant de vérifier si la vue émettrice de l'événement est bien le bouton "Annuler".

c) gestion au moyen d'un listener implicite

Bien sûr, on peut créer un listener à la volée pour le bouton "Laisser tomber" - c'est le style préféré des développeurs Android :

OkHandler (3)

```
package basics.gui;

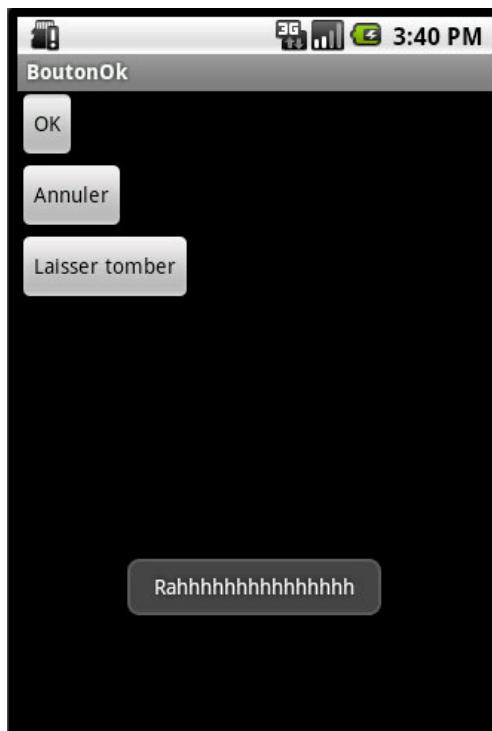
import android.app.Activity; ...

public class OkHandler extends Activity // implements OnClickListener
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
```

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

Button b = (Button) this.findViewById(R.id.bouton_annuler);
b.setOnClickListener(this);

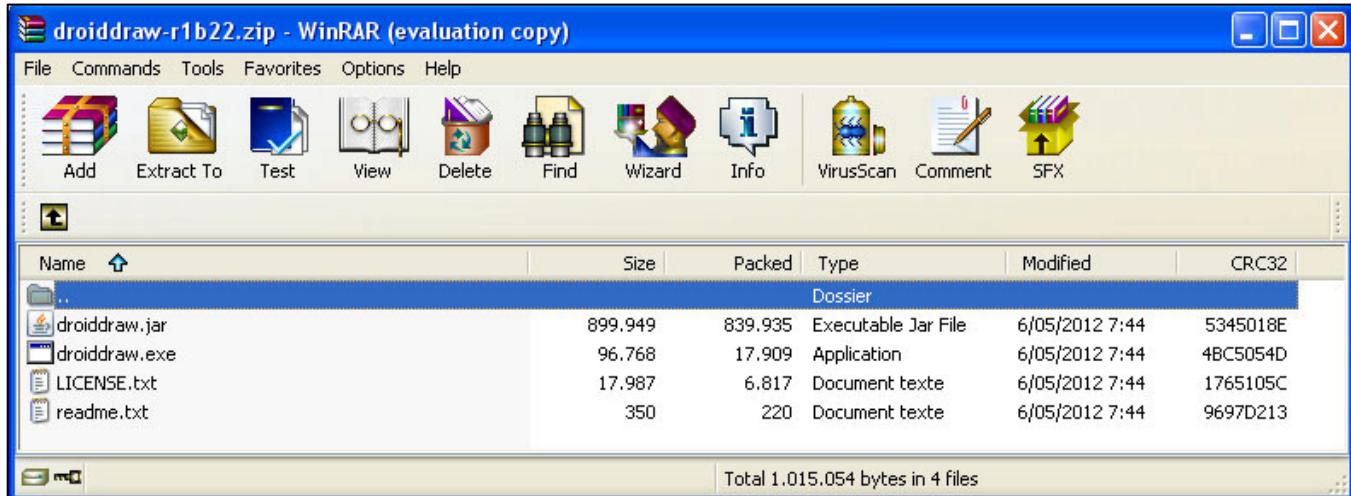
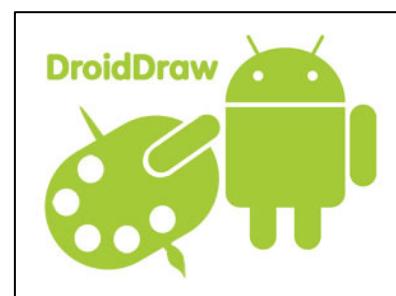
Button bfin = (Button) this.findViewById(R.id.bouton_assez);
bfin.setOnClickListener(
    new View.OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            // TODO Auto-generated method stub
            // Toast.makeText(this,"Rahhhhhhhhhhhhhh",
            // Toast.LENGTH_LONG).show();
            // this is referring to the View.OnClickListener
            // instead of the Activity
            Toast.makeText(OkHandler.this,"Rahhhhhhhhhhhhhh",
                           Toast.LENGTH_LONG).show();
        }
    });
...
}
```



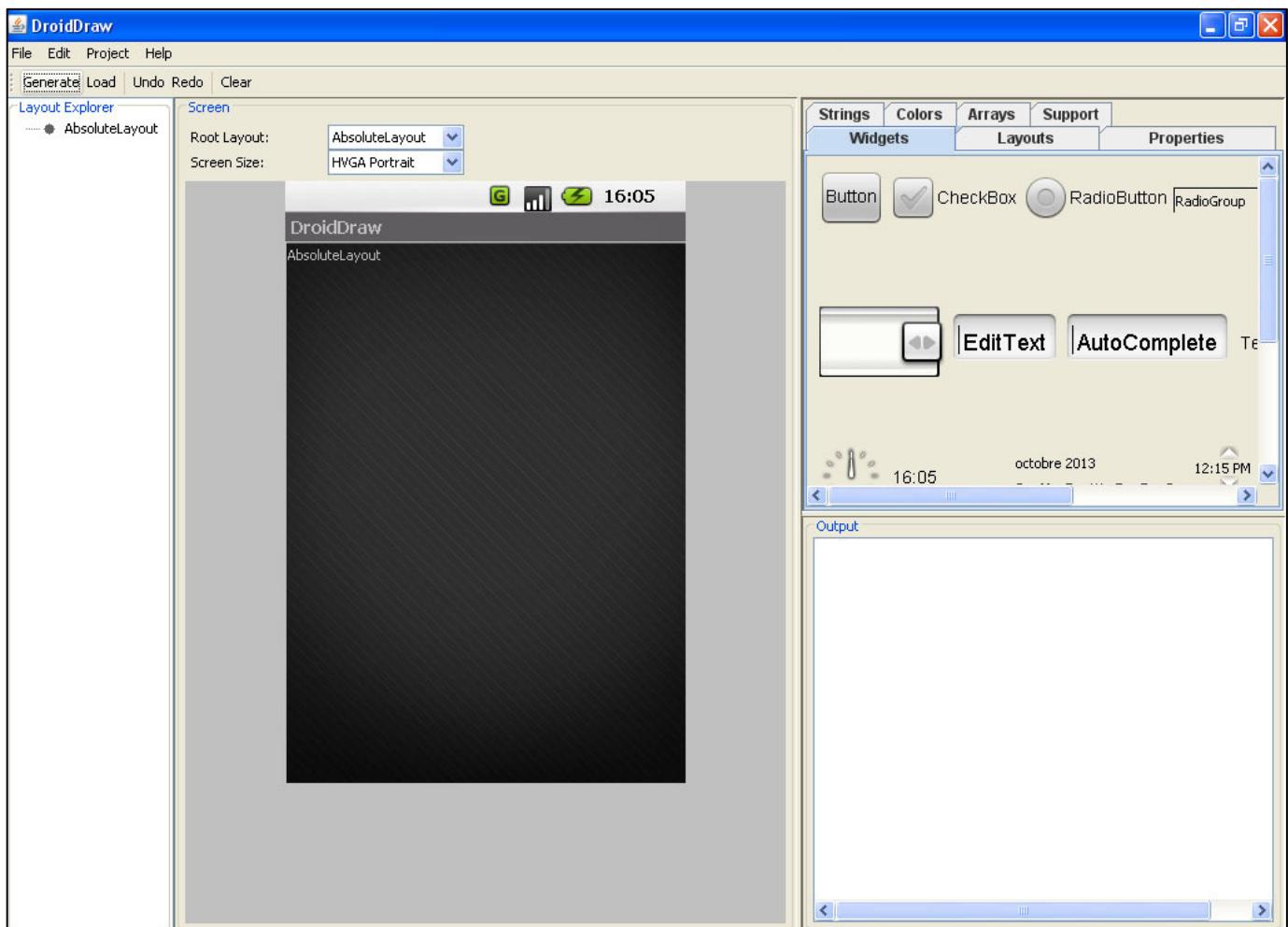
Et il faut vraiment confectionner un tel GUI à la main ?

15. L'outil DroidDraw

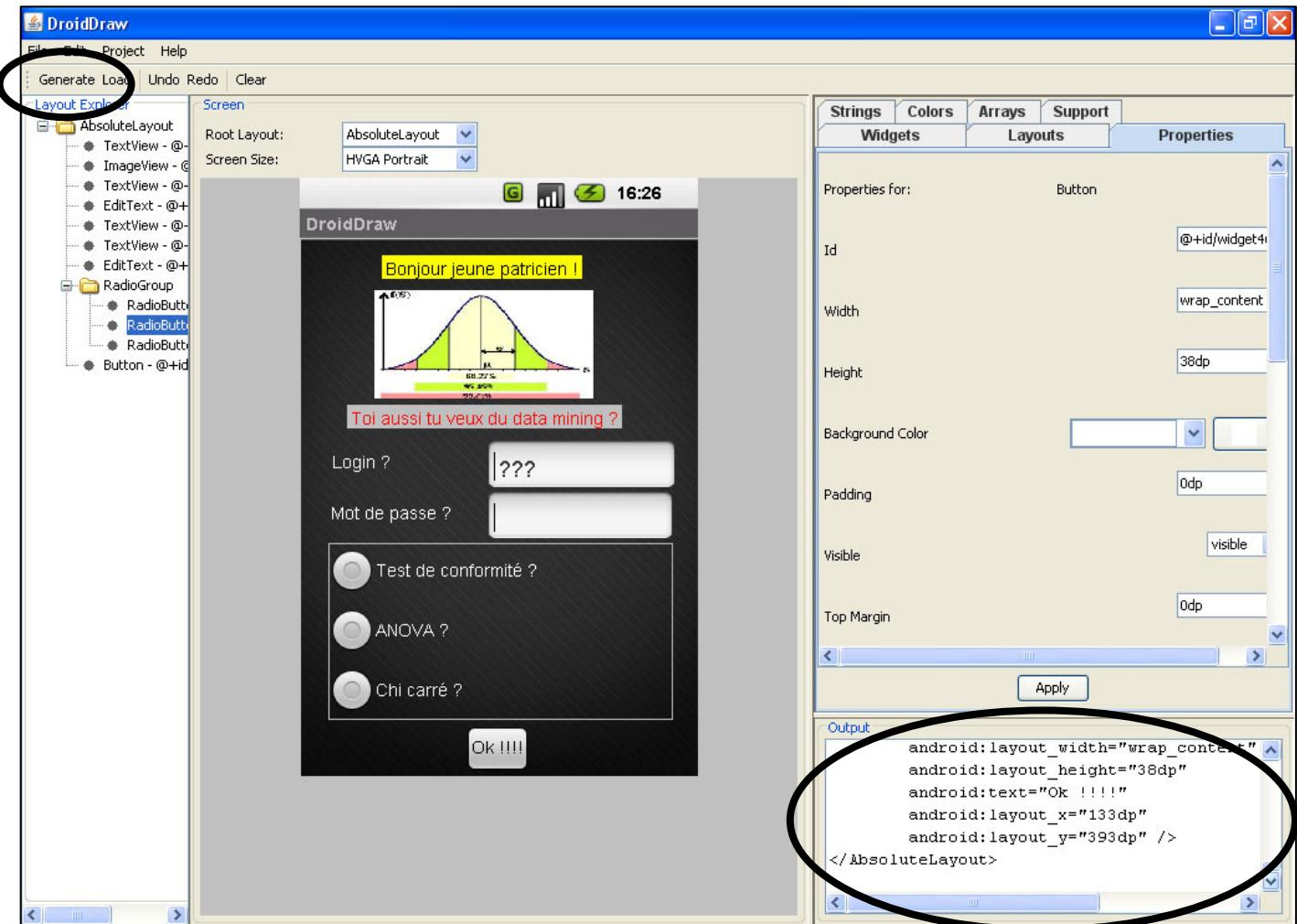
Contrairement à Eclipse, NetBeans ne possède pas (encore) de designer permettant de réaliser aisément les interfaces graphiques. Mais une solution, d'ailleurs connue des développeurs Eclipse, est l'outil DroidDraw trouvé sur <http://www.droiddraw.org/>



Une fois démarré, cet outil présente un interface de type studio graphique assez classique, avec composants à faire glisser et propriétés à paramétrier :



Dire que l'interactivité et l'ergonomie sont poussées à leur paroxysme serait sans doute un peu mentir ;-) Mais on peut tout de même parvenir à confectionner un interface du type suivant avec génération (commande "Generate" du menu) et récupération du code XML correspondant :



En fait, avec un peu d'habitude, on peut aussi travailler directement sur le code XML et demander à générer avec la commande symétrique "Load" du menu. Quoi qu'il en soit, après opérations diverses, on peut obtenir un fichier main.xml décrivant l'interface, avec notamment ceci :

main.xml (extraits)

```

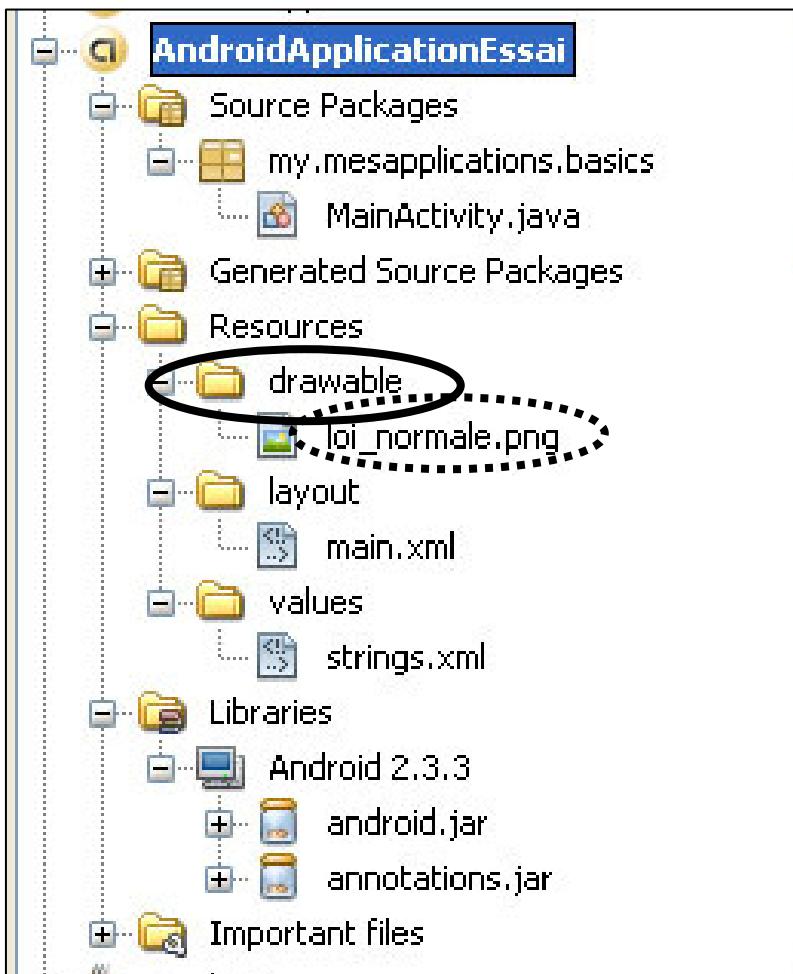
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
    android:id="@+id/widget32"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    ...

```

```
<TextView  
    android:id="@+id/widget33" ...  
    android:background="@color/yellow"  
    android:text="Bonjour jeune patricien !"  
    android:textColor="@color/black" ...  
...  
<ImageView  
    android:id="@+id/widget43" ...  
    android:src="@drawable/loi_normale"  
    android:layout_x="60dp"  
    android:layout_y="277dp" />  
/>
```

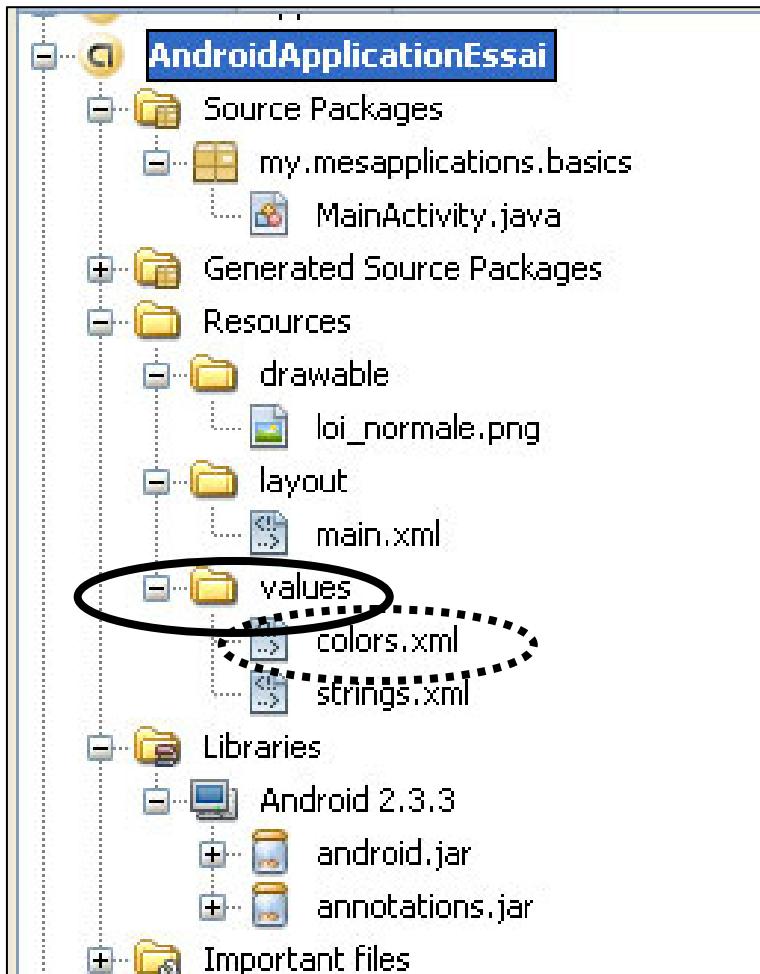
Ceci sous-entend que notre répertoire ressources s'est enrichi par rapport à celui de l'application "Hello World !" :

a) en ce qui concerne l'image :



On remarquera que le nom du fichier image respecte les règles d'Unix !

b) pour les couleurs utilisées :



où la fichier XML correspondant (qui peut être généré à partir de DroidDraw) décrit les couleurs :

colors.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
    <color name="black">#000</color>
    <color name="yellow">#0f0</color>
</resources>
```

16. Les menus

16.1 Le menu classique

Tout utilisateur d'Android est habitué à faire apparaître un menu au sein d'une application par sollicitation de la touche "menu" de son smartphone. Les deux questions qui se posent donc à nous sont



- ◆ comment définir et faire apparaître un menu ?
- ◆ comment gérer le choix d'un item de menu ?

En fait, le menu doit d'abord être défini au sein d'un fichier XML à l'allure suivante :

menu_reservation.xml

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

    <item android:id="@+id/reservation"
          android:title="Reservations">
        <!--android:icon="@drawable/option"-->

        <menu android:id="@+id/sousmenu">
            <item android:id="@+id/vol"
                  android:title="Un vol" />

            <item android:id="@+id/chambre"
                  android:title="Une chambre" />
        </menu>
    </item>

    <item android:id="@+id/quitter"
          android:title="Quitter" />
        <!--android:icon="@drawable/quit"-->
</menu>
```

On comprend sans peine l'agencement des items au sein de la balise menu, un item comportant un sous-menu le spécifiant en déclarant lui-même une nouvelle balise menu. Ce fichier XML doit se trouver dans un répertoire "menu" du répertoire res.

Le menu étant défini, comment le faire apparaître si l'utilisateur le souhaite ? En redéfinissant la méthode

```
public boolean onCreateOptionsMenu (Menu menu)
```

Cette méthode est appelée une seule fois, quand l'utilisateur le demande. Nous allons évidemment écrire dans sa redéfinition la construction du menu. Ceci se fait assez simplement en utilisant une instance de la classe **MenuItemInflater** (package android.view) dont la mission est de créer le menu à partir du fichier menu ressource au moyen de la méthode :

```
public void inflate (int menuRes, Menu menu)
```

le premier paramètre permettant de désigner la ressource menu dans le fichier R (typiquement ici "R.menu.menu_reservation"), le second ne faisant que reprendre le paramètre de

onCreateOptionsMenu désignant l'objet Menu à remplir ("gonfler") avec les items du fichier XML. Plutôt que d'instancier cet "inflater", on préfère en obtenir un au moyen de la méthode d'Activity :

```
public MenuInflater getMenuInflater()
```

qui fournit cet objet dédié au contexte qu'est l'activité.

On peut donc imaginer :

MenuActivity.java

```
package my.applications.divers;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;

public class MenuActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

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

Le résultat à l'exécution, après sollicitation de la touche "menu" à droite :



Reste à répondre aux sollicitations d'un menu ou d'un item de menu. Pour cela, il suffit de redéfinir la méthode d'Activity

public boolean **onOptionsItemSelected** (MenuItem item)

qui est appelée chaque fois qu'un item de menu est sélectionné (il est passé en paramètre). Il ne reste alors plus qu'à se demander quel est cet item, en utilisant la méthode de MenuItem :

public int **getItemId** ()

et d'enclencher l'action que l'on veut y associer (ici, de simples Toasts) :

MenuActivity.java (version 2)

```
package my.applications.divers;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.Toast;

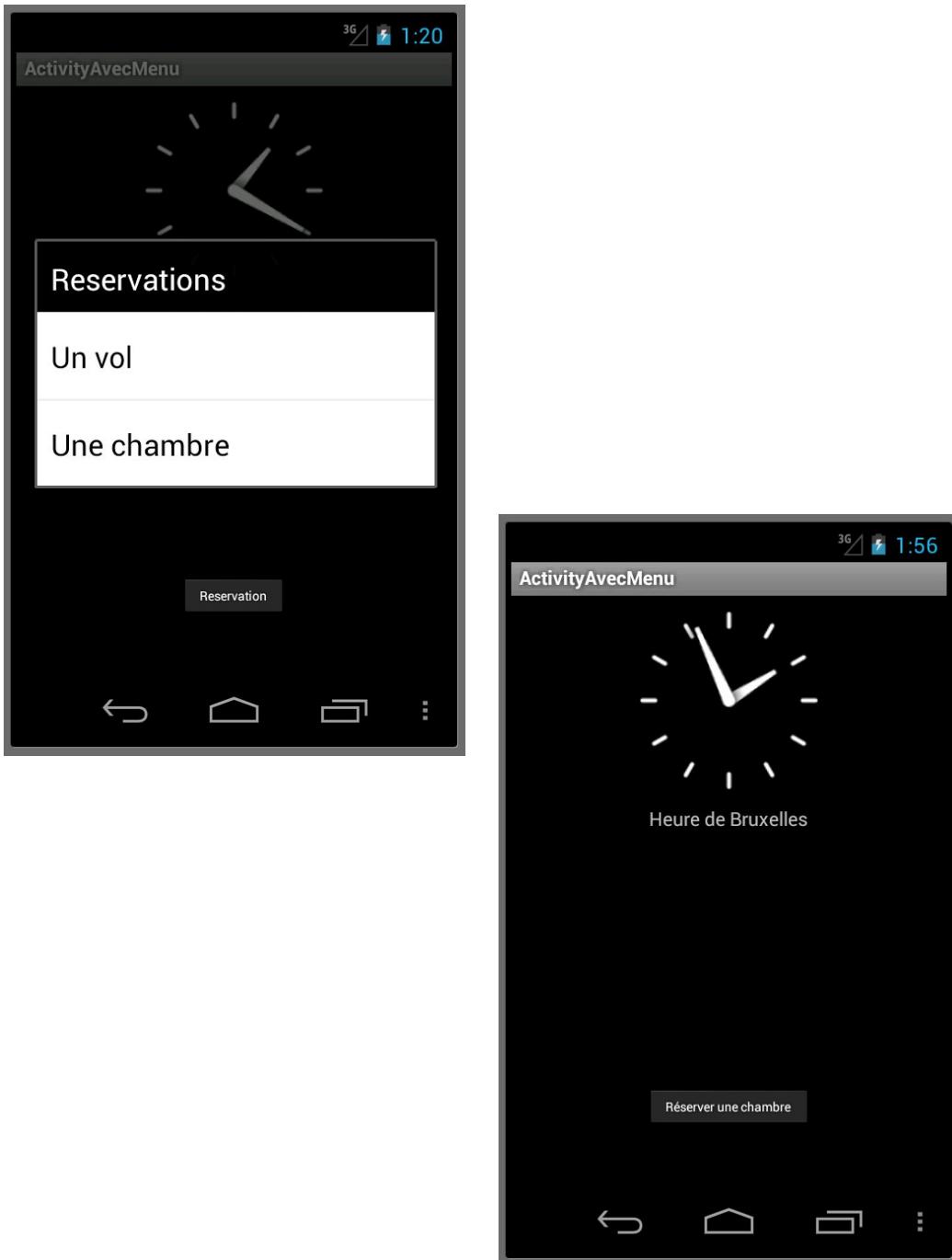
public class MenuActivity extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu_reservation, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        switch (item.getItemId())
        {
            case R.id.reservation: Toast.makeText(MenuActivity.this, "Reservation",
                Toast.LENGTH_SHORT).show();
                return true;
            case R.id.vol: Toast.makeText(MenuActivity.this, "Réserver un vol",
                Toast.LENGTH_SHORT).show();
                return true;
        }
    }
}
```

```
case R.id.chambre: Toast.makeText(MenuActivity.this, "Réserver une chambre",  
        Toast.LENGTH_SHORT).show();  
    return true;  
case R.id.quitter: finish();  
    return true;  
}  
return false;  
}  
}
```

Cela donne :



16.2 La barre des actions

La tendance actuelle est de plus en plus de remplacer le menu classique par une "**Action Bar**", soit l'équivalent du menu sous forme d'une liste d'icônes plus ou moins discrètes placées dans la coin supérieur droit de l'interface graphique.

Transformons donc notre menu avec sous menus en le menu plus "linéaire" suivant :

menu_reservation_toolbar.xml

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

<item android:id="@+id/voyage"
      android:title="Reservations"
      android:showAsAction="ifRoom"
      android:icon="@drawable/sun" >
</item>

<item android:id="@+id/vol"
      android:title="Un vol"
      android:showAsAction="ifRoom"
      android:icon="@drawable/globe" />

<item android:id="@+id/chambre"
      android:title="Une chambre"
      android:showAsAction="ifRoom"
      android:icon="@drawable/flag" />

<item android:id="@+id/quitter"
      android:title="Quitter"
      android:showAsAction="ifRoom"
      android:icon="@drawable/cloud" />

</menu>
```

On remarque deux nouveautés : chaque item se voit doté

- ◆ d'une icône, petite image png placée dans le répertoire res\drawable (on trouve ce genre d'icônes sur des sites Web, comme www.iconesgratuites.fr/categorie/Android.html - par exemple);
- ◆ d'un attribut android:showAsAction expliquant si l'icône doit apparaître si il y a assez de place ("ifRoom") ou toujours "always".

Reprendons notre application précédente :

MenuActivity.java (version 3)

```
package my.applications.divers;

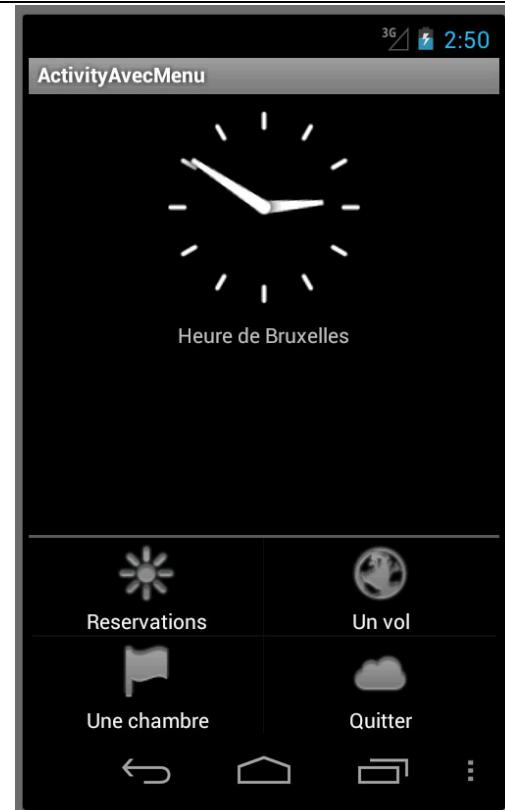
import android.app.Activity;
...
```

```
public class MenuActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState) { ... }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) { ... }

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        switch (item.getItemId())
        {
            case R.id.voyage: Toast.makeText(MenuActivity.this, "Reserver un voyage",
                Toast.LENGTH_SHORT).show();
                return true;
            case R.id.vol: Toast.makeText(MenuActivity.this, "Réserver un vol",
                Toast.LENGTH_SHORT).show();
                return true;
            case R.id.chambre: Toast.makeText(MenuActivity.this, "Réserver une chambre",
                Toast.LENGTH_SHORT).show();
                return true;
            case R.id.quitter: finish();
                return true;
        }
        return false;
    }
}
```

Bref, pas de grand changement - mais le résultat :



fait certes apparaître les icônes, mais dans un menu conventionnel ! En fait, les barres d'action n'existent que depuis la version Honeycomb d'Android (donc 3.0 = 11). Et donc, comme souvent dans les applications Android, il faut écrire dans le fichier AndroidManifest le fait que l'on travaille au moins dans cette version, soit quelque chose de ce genre :

AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="my.applications.divers"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:label="@string/app_name" android:icon="@drawable/ic_launcher">
        <activity android:name="MenuActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="11" />
</manifest>
```

Et voilà alors le résultat :



Bien sûr, si on orne les items "vol" et "chambre" d'un attribut

android:showAsAction="always",

on obtiendra :

17.Une application GUI simple

Nous allons imaginer une simplissime application Android (mais plus évoluée que le "Hello world !") destinée aux médecins effectuant leurs consultations et souhaitant garder une trace de leurs contacts avec leurs patients. A priori, on pense à une application au look suivant :



La structure du GUI est assez simple mais met en action différents layouts et contrôles : un petit passage par DroidDraw :



donne (après le plus souvent des rectifications manuelles dans le fichier XML ;-)) !) :

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:id="@+id/widget32"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android">
<LinearLayout
    android:id="@+id/widget33"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
<TextView
    android:id="@+id/widget35"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Nom :" />
<EditText
    android:id="@+id/nom"
    android:layout_width="110dp"
    android:layout_height="wrap_content"
    android:textSize="18sp" />
</LinearLayout>
<LinearLayout
    android:id="@+id/widget43"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
<TextView
    android:id="@+id/widget44"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Diagnostic :" />
<EditText
    android:id="@+id/diagnostic"
    android:layout_width="132dp"
    android:layout_height="wrap_content"
    android:textSize="18sp" />
</LinearLayout>
<Button
    android:id="@+id/sauver"
    android:layout_width="309dp"
    android:layout_height="wrap_content"
    android:text="Sauvegarder" />
<TextView
    android:id="@+id/indicateur"
    android:layout_width="303dp"
    android:layout_height="wrap_content"
    android:text="---" />
```

```
<TableLayout
    android:id="@+id/widget45"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:stretchColumns="1"
    android:shrinkColumns="1">
<TableRow
    android:id="@+id/widget91"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
<TextView
    android:id="@+id/widget60"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Lieu de la consultation : "/>
<RadioGroup
    android:id="@+id/endroit"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical">
<RadioButton
    android:id="@+id/domicile"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="domicile" />
<RadioButton
    android:id="@+id/cabinet"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="cabinet" />
<RadioButton
    android:id="@+id/hopital"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="hôpital" />
</RadioGroup>
</TableRow>
</TableLayout>
</LinearLayout>
```

Le fichier R.java qui sera généré à la compilation sera :

R.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 my.mesapplications.medecin;  
  
public final class R {  
    public static final class attr {  
    }  
    public static final class id {  
        public static final int cabinet=0x7f04000a;  
        public static final int consultations=0x7f04000e;  
        public static final int diagnostic=0x7f040005;  
        public static final int domicile=0x7f040009;  
        public static final int endroit=0x7f040008;  
        public static final int hopital=0x7f04000b;  
        public static final int indicateur=0x7f04000d;  
        public static final int nom=0x7f040003;  
        public static final int sauver=0x7f04000c;  
        public static final int uneconsultation=0x7f040001;  
        public static final int widget35=0x7f040002;  
        public static final int widget44=0x7f040004;  
        public static final int widget48=0x7f040000;  
        public static final int widget60=0x7f040007;  
        public static final int widget91=0x7f040006;  
    }  
    public static final class layout {  
        public static final int main=0x7f020000;  
    }  
    public static final class string {  
        public static final int app_name=0x7f030000;  
    }  
}
```

En termes de programmation, nous nous contenterons dans un premier temps de créer, à chaque sollicitation du bouton "sauver", une instance de la classe de classe élémentaire Consultation :

Consultation.java

```
package my.mesapplications.medecin;  
  
public class Consultation  
{  
    private String nomPatient;  
    private String diagnostic;
```

```
public Consultation()
{
    nomPatient = null; diagnostic = null;
}
public String getNomPatient() { return nomPatient; }
public void setNomPatient(String nomPatient) { this.nomPatient = nomPatient; }
public String getDiagnostic() { return diagnostic; }
public void setDiagnostic(String diagnostic) { this.diagnostic = diagnostic; }

@Override
public String toString()
{
    return getNomPatient() + getDiagnostic();
}
```

Le traitement de l'événement sera assuré par un listener à la volée style "Android-preferred" :

MainActivity.java

```
package my.mesapplications.medecin;

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

public class MainActivity extends Activity
{
    Consultation laConsultation = new Consultation();

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

        Button sauvegarde = (Button) this.findViewById(R.id.sauver);
        sauvegarde.setOnClickListener(onSauvegarde);
    }

    private View.OnClickListener onSauvegarde = new View.OnClickListener()
    {
        public void onClick(View view)
        {
            EditText nom = (EditText) findViewById(R.id.nom);
            EditText diagnostic = (EditText) findViewById(R.id.diagnostic);
        }
    }
}
```

```
        laConsultation.setNomPatient(nom.getText().toString());
        laConsultation.setDiagnostic(diagnostic.getText().toString());

        TextView sauvetageFait = (TextView) findViewById(R.id.indicateur);
        sauvetageFait.setText("Sauvetage effectué !!");

    }
};

}
```

Bien sûr, créer un objet Consultation ne présente guère d'intérêt : il faudrait à tout le moins mémoriser les consultations et les afficher dans une liste, et même aussi rendre cette liste persistante.

18. Les listes et les adapters

Nous allons à présent ajouter une boîte de liste qui contiendra les consultations. Un tel widget est classiquement une **ListView**. Ce genre de composant est en fait géré selon le framework MVC, tout comme ses homologues JList et JComboBox de Swing. Et même "mieux" puisque les trois composants MVC existent de manière distincte. En fait, pour programmer l'ajout dynamique d'éléments à une liste, il nous faudra :



- ◆ évidemment une instance de ListView, dont les caractéristiques visuelles sont toujours définies dans main.xml et qui sera la **Vue**;
- ◆ un container mémoire du type **List<?>** qui contiendra les données à afficher et/ou à sélectionner : bien sûr, il sera le **Modèle**;
- ◆ un objet "**adapter**", en fait un **Contrôleur**, responsable de la synchronisation entre les données et leur vue.

Un tel adapter implémente l'interface android.widget.**Adapter**, qui déclare des méthodes comme :

```
public abstract Object getItem (int position)
```

ou encore

```
public abstract void registerDataSetObserver (DataSetObserver observer)
```

où l'on devine qu'un **DataSetObserver** est un objet averti des modifications dans les données.

La classe usuelle utilisée comme adapter est android.widget.**ArrayAdapter<T>**, qui représente une version d'adapter plus particulièrement adaptée aux widgets de type ListView (elle implémente d'ailleurs l'interface android.widget.ListAdapter). Un constructeur classique est

```
public ArrayAdapter (Context context, int textViewResourceId, List<T> objects)
```

où le premier paramètre est habituellement l'activité hôte, le troisième désigne le modèle et le deuxième vaut classiquement

simple_list_item_1

ce qui signifie que les données du modèle, qui seront transformées par défaut par l'adapter en chaînes de caractères (par appel de `toString()`), seront utilisées pour initialiser des `TextView` qui seront en définitive ajoutées à la `ListView`.

Nous allons donc ajouter une `ListView` dans notre GUI, mais se pose alors un problème : notre liste et nos composants graphiques doivent occuper la place maximale sans occulter les autres contrôles graphiques. Pour obtenir cela, on peut utiliser un `ScrollView` pour les composants autres que la `ListView` (qui, elle, utilise un scroll natif). Cela donne :

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/gui"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivityAdapter" >

    <ListView
        android:id="@+id/consultations"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true" />

    <ScrollView
        android:id="@+id/widget32"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        xmlns:android="http://schemas.android.com/apk/res/android">
        <LinearLayout
            android:id="@+id/gui2"
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="match_parent">
            <TextView
                android:id="@+id/widget35"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Nom :" />
        
    

```

```
<EditText  
    android:id="@+id/nom"  
    android:layout_width="110dp"  
    android:layout_height="wrap_content"  
    android:textSize="18sp" />  
<TextView  
    android:id="@+id/widget44"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Diagnostic :" />  
<EditText  
    android:id="@+id/diagnostic"  
    android:layout_width="132dp"  
    android:layout_height="wrap_content"  
    android:textSize="18sp" />  
<Button android:id="@+id/sauver"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Sauver cette consultation"/>  
<TextView  
    android:id="@+id/indicateur"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="RAS"/>  
</LinearLayout>  
</ScrollView>  
</LinearLayout>
```

avec

MainActivity.java

```
package my.mesapplications.medecin;  
  
import android.app.Activity;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.ArrayAdapter;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.ListView;  
import android.widget.TextView;  
import java.util.ArrayList;  
import java.util.List;  
  
public class MainActivity extends Activity  
{  
    List<Consultation> modeleConsultations = new ArrayList<Consultation>();  
    ArrayAdapter<Consultation> controleurConsultations = null;
```

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    ListView vueConsultations = (ListView) findViewById(R.id.consultations);
    controleurConsultations = new ArrayAdapter<Consultation>
        (this, android.R.layout.simple_list_item_1, modeleConsultations );
    vueConsultations.setAdapter(controleurConsultations);

    Button sauvegarde = (Button) this.findViewById(R.id.sauver);
    sauvegarde.setOnClickListener(onSauvegarde);
}

private View.OnClickListener onSauvegarde = new View.OnClickListener()
{
    public void onClick(View view)
    {
        EditText nom = (EditText) findViewById(R.id.nom);
        EditText diagnostic = (EditText) findViewById(R.id.diagnostic);

        Consultation laConsultation = new Consultation();
        laConsultation.setNomPatient(nom.getText().toString());
        laConsultation.setDiagnostic(diagnostic.getText().toString());

        controleurConsultations.add(laConsultation);
        controleurConsultations.notifyDataSetChanged();

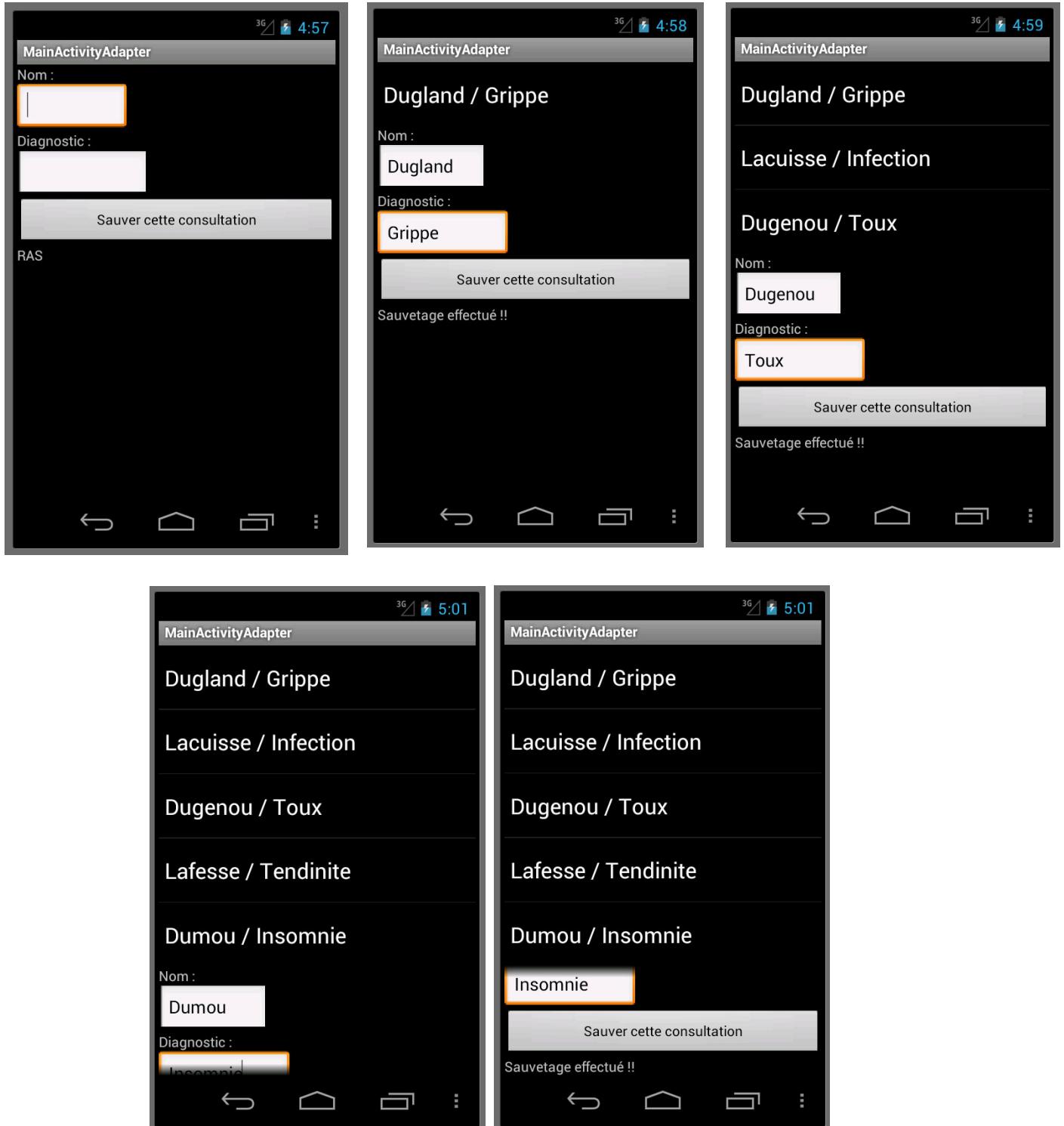
        TextView sauvetageFait = (TextView) findViewById(R.id.indicateur);
        sauvetageFait.setText("Sauvetage effectué !!");
    }
};
```

On remarquera l'utilisation de la méthode de l'adapter :

```
public void notifyDataSetChanged ()
```

qui signale aux observers que les data ont été modifiées et que toutes les vues intéressées doivent se mettre à jour.

A l'exécution :



19. Les communications réseaux

Les communications réseaux sous Android nous sont bien connues ! En effet, elles utilisent les classes sockets de la programmation réseau habituelle sous Java, comme la classe Socket par exemple. Une petite condition pour que notre application puisque accéder au réseau : il faut ajouter dans le fichier manifeste une clause autorisant cet accès, ce qui se fait au moyen de



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

Donc :

AndroidManifest.xml

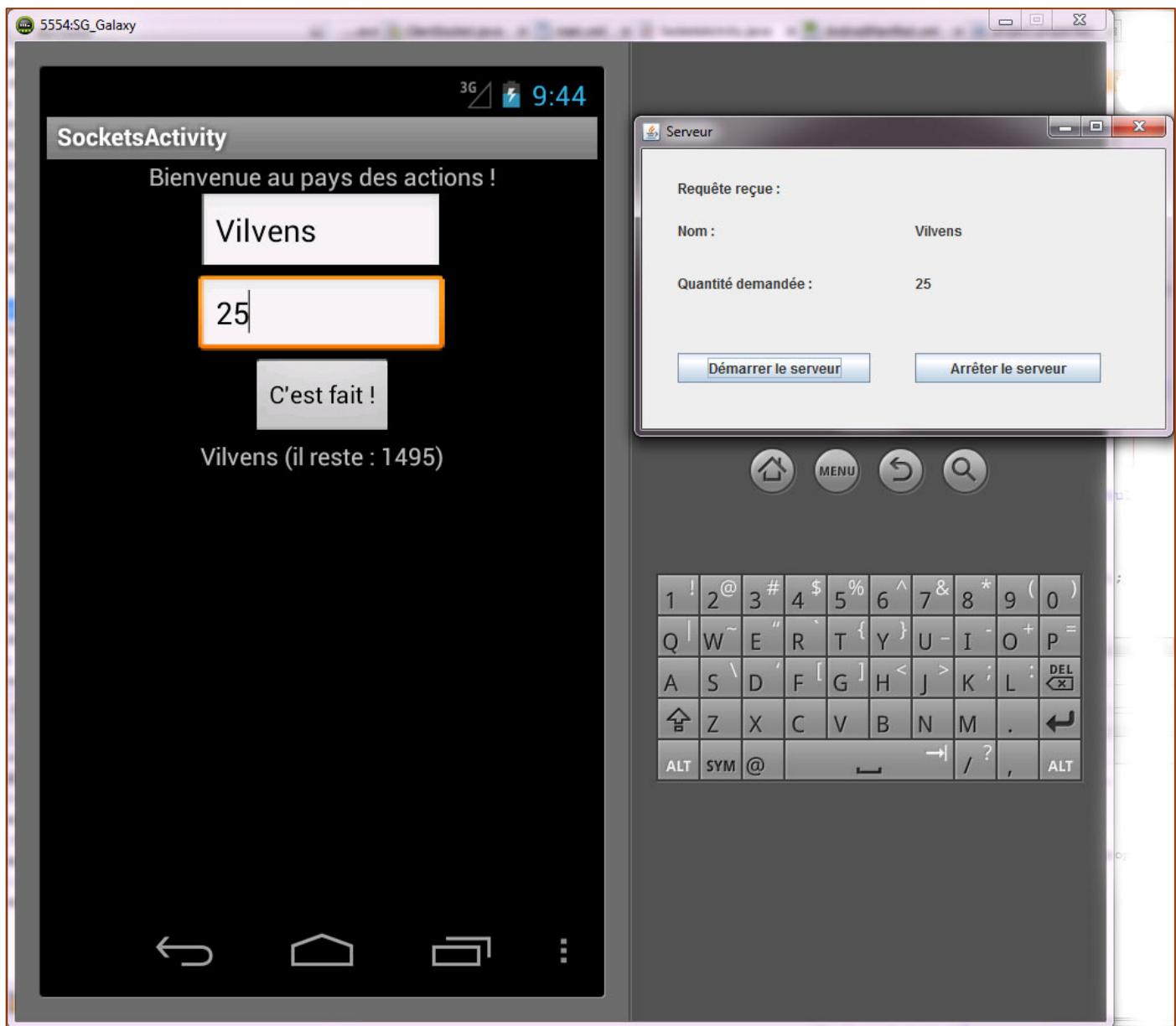
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="applics.basics"
    ...
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".HelloBonjour"
            android:label="@string/app_name">
            ...
        </activity>
    </application>
    ...
    <uses-permission android:name="android.permission.INTERNET"></uses-
    permission>
</manifest>
```

On peut donc imaginer modifier notre application pour qu'elle puisse atteindre un serveur réseau tout à fait conventionnel : il s'agit d'un serveur FenServeurSocket dont l'air devrait dire quelque chose au lecteur qui a lu Java II ;-). Attention toutefois : il faut éventuellement harmoniser les JDK. Pour cela, dans le fichier properties du projet (répertoire nbprojetc), il faut rectifier

```
javac.source=1.7
javac.target=1.7
```

en lieu et place (par exemple) de :

```
javac.source=1.5
javac.target=1.5
```



SocketsActivity.java

```
package my.mesapplications.network;

import android.app.Activity;
import android.view.View.OnClickListener;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
```

```

public class SocketsActivity extends Activity implements OnClickListener
{
    Button BLogin;
    EditText ETLogin;
    EditText ETCombien;
    TextView TVReponse;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        BLogin = (Button)this.findViewById(R.id.ButtonEnvoi);
        BLogin.setOnClickListener(this);
        ETLogin = (EditText)this.findViewById(R.id.EditTextUser);
        ETCombien = (EditText)this.findViewById(R.id.EditTextCombien);
        TVReponse = (TextView)this.findViewById(R.id.TextViewReponse);
    }
    @Override
    public void onClick(View arg0)
    {
        String nom = (String) ETLogin.getText().toString();
        int quantite = Integer.parseInt((String) ETCombien.getText().toString());
        contacteServeur(nom, quantite);
        BLogin.setText("C'est fait !");
    }

    private void contacteServeur(String n, int q)
    {
        Socket cliSock=null;
        DataInputStream dis = null; DataOutputStream dos = null;
        try
        {
            cliSock = new Socket(InetAddress.getByName("192.168.1.5"),
                50000);
        }
        catch (UnknownHostException e)
        { System.err.println("Erreur ! Host non trouvé [" + e + "]"); }
        catch (IOException e)
        { System.err.println("Erreur ! Pas de connexion ? [" + e + "]"); }
        try
        {
            dis = new DataInputStream(cliSock.getInputStream());
            dos = new DataOutputStream(cliSock.getOutputStream());
        }
        catch (IOException e){ }

        String outNom = n; int outQuantite = q;
        String reponse = null; int inQuantiteRestante = 0;
    }
}

```

```
if (!outNom.equals("")) && outQuantite>0 && dos!=null && dis!=null)
{
    try
    {
        dos.writeUTF(outNom);dos.writeInt(outQuantite);
        reponse = dis.readUTF();
        inQuantiteRestante = dis.readInt();
        TVReponse.setText(reponse + " (il reste : " +
                           inQuantiteRestante+ ")");
    }
    catch (IOException e) { }
    finally
    {
        try { dis.close();dos.close(); }
        catch (IOException e) { e.printStackTrace(); }
    }
}
}
```

avec

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:id="@+id/widget32"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <TextView
        android:id="@+id/widget33"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bienvenue au pays des actions !"
        android:layout_gravity="center_horizontal" />
    <EditText
        android:id="@+id/EditTextUser"
        android:layout_width="143dp"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:layout_gravity="center_horizontal" />
    <EditText
        android:id="@+id/EditTextCombien"
        android:layout_width="143dp"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:layout_gravity="center_horizontal" />
```

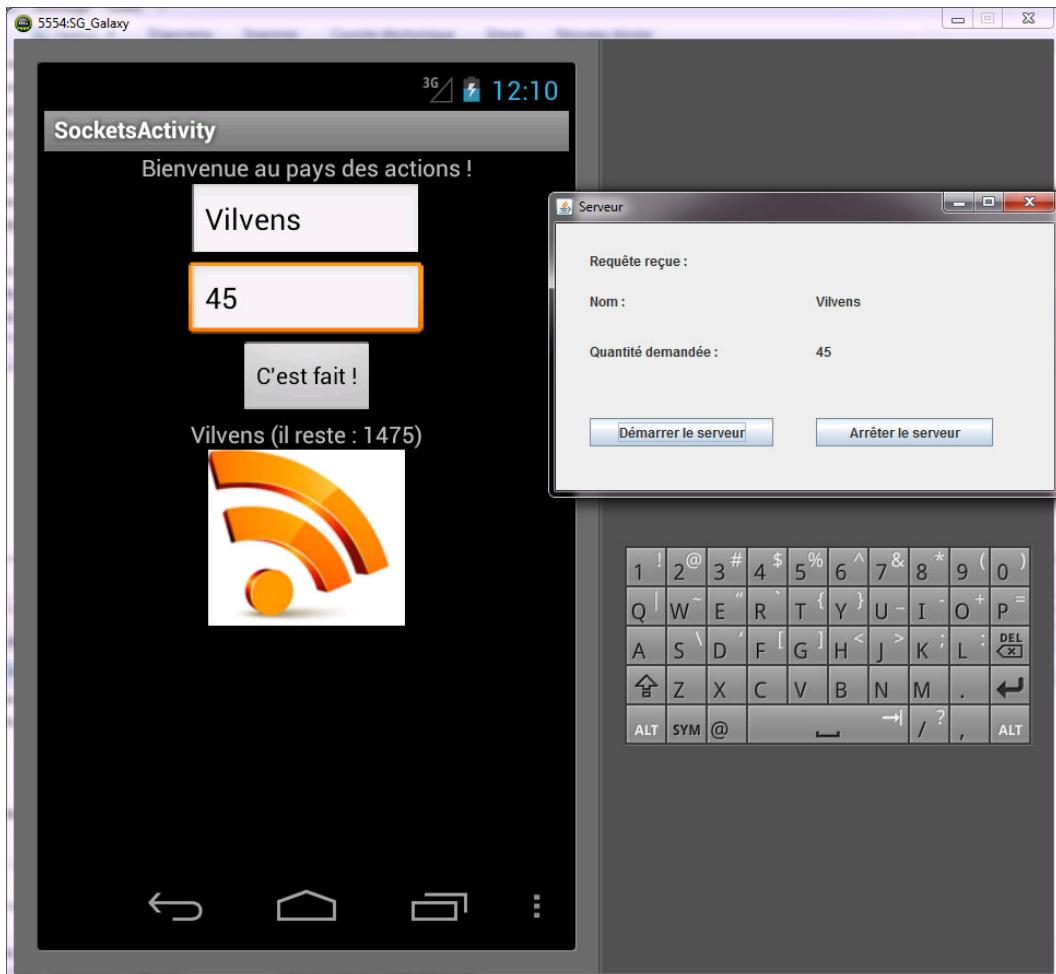
```

<Button
    android:id="@+id/ButtonEnvoi"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="C'est fait !"
    android:layout_gravity="center_horizontal" />
<TextView
    android:id="@+id/TextViewReponse"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Voyons voir ..."
    android:layout_gravity="center_horizontal" />
<ImageView
    android:id="@+id/IVOk"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:visibility="invisible"
    android:layout_gravity="center_horizontal" />
</LinearLayout>

```

Le serveur J2SE n'a donc absolument pas été modifié ☺ !

On peut même imaginer de faire apparaître une image illustrant le fait que la transaction réseau a réussi et qui n'apparaîtrait qu'après la réception de la réponse du serveur :



Nous allons pour cela :

- ◆ ajouter l'image en question (reseau.png) dans la répertoire drawable du répertoire res de notre projet :
- ◆ modifier le layout avec une ImageView, au départ invisible :

main.xml (2)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:id="@+id/widget32"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android">
    ...
<TextView
    android:id="@+id/TextViewReponse"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Voyons voir ..."
    android:layout_gravity="center_horizontal" />
<ImageView
    android:id="@+id/IVOk"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:visibility="visible"
    android:layout_gravity="center_horizontal" />
</LinearLayout>
```

- ◆ ajouter au code de la méthode contacteServeur() les instructions d'affichage :

SocketsActivity.java (2)

```
package my.mesapplications.network;

import android.app.Activity;
...

public class SocketsActivity extends Activity implements OnClickListener
{
    Button BLogin;
    EditText ETLogin;
    EditText ETCombien;
    TextView TVReponse;

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

```
@Override
public void onClick(View arg0)
{
    ...
    contacteServeur(nom, quantite);
    BLogin.setText("C'est fait !");
}

private void contacteServeur(String n, int q)
{
    ...
    if (!outNom.equals("") && outQuantite>0 && dos!=null && dis!=null)
    {
        try
        {
            dos.writeUTF(outNom);dos.writeInt(outQuantite);
            reponse = dis.readUTF();
            inQuantiteRestante = dis.readInt();
            TVReponse.setText(reponse + " (il reste : " +
                inQuantiteRestante+ ")");
            Resources res =this.getResources();
            Drawable draw = res.getDrawable(R.drawable.reseau);
            ImageView img = (ImageView) this.findViewById(R.id.IVOK);
            img.setImageDrawable(draw);
            img.setVisibility(View.VISIBLE);
        }
        catch (IOException e) { }
        ...
    }
}
```

20. Threads asynchrones et GUIs

Notre pratique des applications installées nous rend bien conscients que l'on a intérêt à threader les opérations réseau afin de ne pas bloquer l'interface graphique. Ainsi, on décharge le thread principal (appelé **UI Thread**) de la surveillance du réseau (par exemple), car celui-ci exécute déjà le code de l'Activity, gère les interactions avec l'utilisateur et régit l'affichage.



Si il est vrai que l'on peut utiliser la traditionnelle classe Thread, il existe une classe dédiée à ce genre de travail : **AsyncTask** (package android.os). Cette classe est un "helper", une classe fournie pour faciliter la manipulation de threads au comportement stéréotypé. Elle permet de créer une tâche qui va s'effectuer en arrière-plan et qui sera capable de communiquer sur son état d'avancement. Clairement, la tâche visée est une opération modérément longue, comme un téléchargement ou un algorithme nécessitant de nombreux calculs, le tout accompagné d'une barre de progression.

La classe est paramétrable :

AsyncTask<Params, Progress, Result>

avec

- ◆ Params : paramètres fournis au thread;
- ◆ Progress : le type de l'information envoyée par le thread au UI thread pour que ce dernier puisse rendre compte de l'état d'avancement;
- ◆ Result : le résultat retourné par le thread.

L'un ou l'autre de ces paramètres peut être Void si il n'a pas d'usage.

Le cycle de vie d'un AsyncTask comporte 4 états, associés à 4 méthodes. Concrètement, un tel objet doit toujours implémenter la méthode **doInBackground()**. C'est là que sera réalisé le traitement lourd de manière asynchrone dans un thread implicite séparé. C'est là aussi que peut être appelée la méthode **publishProgress()** qui permet la mise à jour de la progression.

Les méthodes **onPreExecute()** (appelée avant le traitement - exemple typique : afficher une barre de progression), **onProgressUpdate()** (appelée en réponse à publishProgress() pour afficher la progression par le UI thread - on ne doit donc pas appeler la méthode onProgressUpdate() directement) et **onPostExecute()** (appelée après le traitement) sont optionnelles. Ces trois méthodes s'exécutent depuis l'UI Thread : elles doivent en effet pouvoir modifier l'interface. Il faudra donc éviter d'y programmer des traitements lourds.

Supposons que nous voulions réaliser l'exemple suivant :



Nous allons donc logiquement utiliser le widget **ProgressBar** (package android.widget) dont une méthode utilisée ici sera :

```
public synchronized void setProgress (int progress)
```

pour mettre à jour la progression affichée. Le composant est bien sûr défini dans main.xml :

main.xml

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

    <Button
        android:layout_marginTop="10dp"
        android:id="@+id/lancer"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Lancer la tâche" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text="Progression de la tâche asynchrone:" />

<ProgressBar
    android:id="@+id/barre_progression"
    style="?android:attr/progressBarStyleHorizontal"
    android:layout_width="fill_parent"
    android:layout_margin="10dp"
    android:layout_height="wrap_content" />

<TextView
    android:id="@+id/widget33"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Et à part ça ? Quoi de neuf ?"
    android:layout_gravity="center_horizontal" />

<EditText
    android:id="@+id/EditTextUser"
    android:layout_width="143dp"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:layout_gravity="center_horizontal" />

<Button
    android:id="@+id/ButtonEnvoi"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Noter"
    android:layout_gravity="center_horizontal" />
</LinearLayout>
```

Notre activité sera alors simple à programmer : elle utilisera une classe privée imbriquée dérivée d'Asynctask dont elle fera démarrer une instance avec la méthode :

```
public final AsyncTask<Params, Progress, Result> execute (Params... params)
```

ActivityAvecAsynctask.java

```
package my.mesapplications.async;
```

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

```
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.Toast;

public class ActivityAvecAsynctask extends Activity
{
    private ProgressBar barre;
    private Button boutonDemarre;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        barre = (ProgressBar) findViewById(R.id.barre_progression);
        boutonDemarre = (Button) findViewById(R.id.lancer);

        boutonDemarre.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                TraitementLourd calcul=new TraitementLourd();
                calcul.execute();
            }
        });
    }

    /**
     * -----
     */
    private class TraitementLourd extends AsyncTask<Void, Integer, Void>
    {
        @Override
        protected void onPreExecute()
        {
            super.onPreExecute();
            Toast.makeText(getApplicationContext(), "Début du traitement asynchrone",
Toast.LENGTH_LONG).show();
        }

        @Override
        protected void onProgressUpdate(Integer... values)
        {
            super.onProgressUpdate(values);
            barre.setProgress(values[0]);
        }
    }
}
```

```

@Override
protected void doInBackground(Void... arg0)
{
    int progress;
    for (progress=0;progress<=100;progress++)
    {
        for (int i=0; i<1000000; i++){ } // ex: résolution d'un grand système d'équations
        publishProgress(progress);
        progress++;
    }
    return null;
}

@Override
protected void onPostExecute(Void result)
{
    Toast.makeText(getApplicationContext(), "Traitement terminé !",
        Toast.LENGTH_LONG).show();
}
}

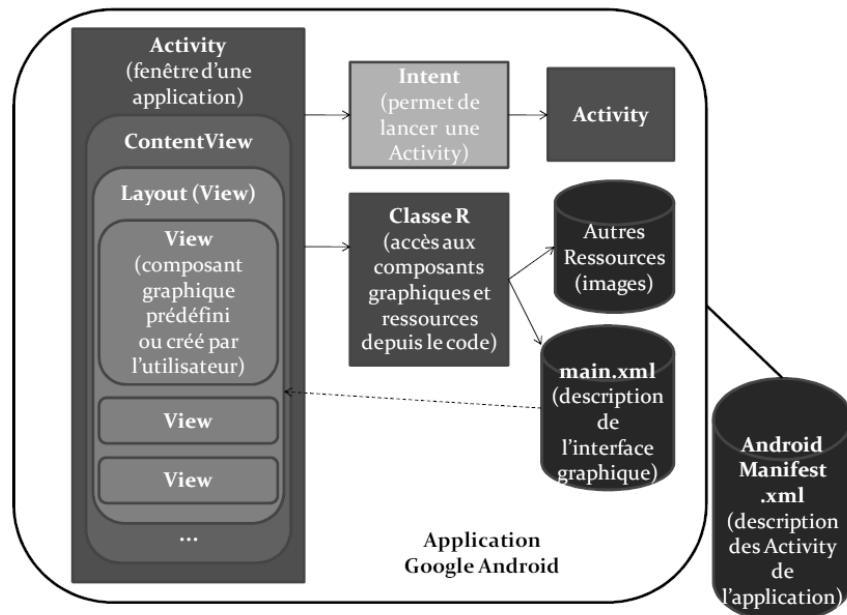
```

On aura aussi remarqué l'utilisation de la classe **Void**, qui permet de manipuler une référence quelconque ainsi que celle d'une liste de paramètres indéterminés spécifiés par l'ellipse "..." (ce qui a le désavantage de ne pas permettre le contrôle des types de ces paramètres).

21. Les intents

21.1 Application, activités et intents

En fait, une application Android apparaît classiquement comme structurée selon ce schéma :



Autrement dit, une application Android est loin d'être monolithique : elle est formée d'activités (et/ou de services) qui dialoguent entre eux au moyen de messages appelés "intents". Mais comment lance-t-on un intent ?

21.2 Un message asynchrone

La politique de sécurité d'Android est modélisée selon le principe de la **sandbox** bien connue en Java : les applications sont presque totalement séparées les unes des autres, garantissant ainsi qu'une application ne peut mettre en péril l'ensemble du système. "Presque totalement", parce qu'il faut tout de même bien de temps en temps qu'un composant applicatif communique avec un autre : c'est là qu'interviennent les **intents**. Pour rappel, un **intent** (les Français disent "**intention**" – bof) est un message asynchrone spécifiant une action et contenant l'URI du composant à manipuler ou de l'action à initier.

On peut donc encore voir un intent comme **un ensemble de données qui peut être passé à un autre composant applicatif** (de la même application ou non) **que l'on fera démarrer**.

Un tel "message" peut être envoyé

- ◆ **explicitement** à un objet bien précis;
- ◆ dans le but de réaliser une action (lire de la musique, scanner un code barre, ...) auquel cas l'envoi est **implicite** (c'est l'objet associé de par sa nature à l'action qui recevra le message).

Comme on l'a déjà deviné lors de l'examen de l'AndroidManifest.xml, un système de filtres permet à chaque application de filtrer et de gérer uniquement les **Intents** qui présentent un intérêt pour celle-ci. Une application peut donc être dans un état d'inactivité tout en restant à l'écoute des intents circulant dans le système.

21.3 Démarrage explicite

Du point de vue programmation, un tel intent est une Instance de la classe **Intent** (package android.content). Parmi les divers constructeurs possibles, celui qui permet d'envoyer un intent à un composant précis est :

```
public Intent (Context packageContext, Class<?> cls)
```

où Context est une classe abstraite dont l'implémentation est fournie par Android (typiquement désigné par le champ this de la classe appelante) et Class est évidemment **l'activité que l'on veut faire démarrer au sein de la même application**.

Donc, si une application est composée de plusieurs écrans qui s'enchaînent les uns à la suite des autres en fonction des actions de l'utilisateur, chaque écran sera représenté par une activité définissant son interface utilisateur et sa logique et donc chaque activité de l'application nécessitera l'emploi d'un Intent pour être démarrée.

Pour ajouter des informations à un Intent, il suffit de faire appel à l'un des ses innombrables méthodes **putExtra()** comme par exemple la méthode :

```
public Intent putExtra (String name, int value)
public Intent putExtra (String name, String value)
...

```

le premier paramètre étant une **clé** et le deuxième la **valeur** associée à la clé selon le principe d'une hashtable. Encore mieux, on peut ajouter toute une hashtable à un Intent avec :

```
public Intent putExtras (Bundle extras)
```

où un objet **Bundle** (package android.os) implémente précisément un mécanisme clé-valeur de type HashTable, avec des méthodes :

```
public void putInt (String key, int value)  
public void putFloat (String key, float value)  
public void putString (String key, String value) ...
```

Les méthodes **getExtras()**, **getIntExtra(String key)**, **getStringExtra(String key)**, ... permettront évidemment de récupérer ces données.

L'objet Intent étant ainsi créé et configuré, une activité cible sera démarrée par l'appel de la méthode :

```
public void startActivity (Intent intent)
```

Donc, dans la version la plus simple :

```
Intent msgDemarrage = new Intent(ActiviteAppellante.this, ActiviteAppelee.class);  
startActivity(msgDemarrage);
```

Un exemple classique d'utilisation de ce procédé est celui d'une application composée de plusieurs écrans qui s'enchaînent les uns à la suite des autres en fonction de l'action de l'utilisateur : chaque écran est associé à une activité avec son interface utilisateur et sa logique et le passage de l'un à l'autre se fera par Intent.

On se souviendra qu'un système de filtres permet à chaque application de filtrer et de gérer uniquement les Intents qui l'intéressent. Ces filtres sont spécifiés dans le fichier AndroidManifest.xml (comme déjà évoqué vaguement au paragraphe 9.3) au moyen d'une balise **<intent-filter>** au sein de la balise **<activity>** - par exemple :

```
<activity android:name="..."  
        android:label="@string/app_name">  
    <intent-filter>  
        <action android:name="android.intent.action.DEFAULT" />  
    </intent-filter>  
</activity>
```

Il est clairement fait allusion à une constante :

```
public static final String ACTION_DEFAULT
```

qui est en fait un synonyme de

```
public static final String ACTION_VIEW = "android.intent.action.VIEW"
```

dont la signification est de provoquer l'affichage de l'interface de l'activité (donc de visualiser les données à l'utilisateur de l'application). On pourra comparer avec notre premier exemple (application "bonjour") :

```
<activity android:name=".HelloBonjour"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

où l'on utilise les constantes :

- ◆ public static final String **ACTION_MAIN** = "android.intent.action.MAIN"
qui spécifie quelle activité est le point d'entrée de l'application (donc, pas d'action véritable);
- ◆ public static final String **CATEGORY_LAUNCHER** =
"android.intent.category.LAUNCHER"
qui spécifie que la vue de l'application doit être affichée au-dessus de toutes les autres vues.

21.4 Communication entre deux activités : mode d'emploi

a) A → B : simple envoi d'un intent

i) Dans l'activité A :

- ◆ créer le Bundle
- ◆ créer l'Intent
- ◆ appeler putExtras() du Bundle
- ◆ appeler startActivity()

ii) Dans l'activité B : dans la méthode

```
public void onCreate(Bundle savedInstanceState) :
```

- ◆ récupérer l'Intent qui a démarré l'activité :
- ```
public Intent getIntent ()
```
- ◆ en extraire le Bundle : public Bundle getExtras() puis en extraire les infos contenues.  
ou  
en extraire les informations une par une par getXXXExtra().

Bien sûr, il conviendra de ne pas oublier de déclarer la deuxième activité dans AndroidManifest.xml :

```
<activity android:name=".AAA"
 android:label="@string/app_name">
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER" />
```

```
</intent-filter>
</activity>
<activity android:name=".BBB"></activity>
```

b) A  $\leftarrow \rightarrow$  B : envoi d'un intent et attente de la réponse

i) Dans l'activité A ( $\rightarrow$ ) :

créer le Bundle

- ◆ créer l'Intent
- ◆ appeler putExtras du Bundle
- ◆ MAIS utiliser :

```
public void startActivityForResult (Intent intent, int requestCode)
```

(le code  $\geq 0$  peut servir à la cible à identifier la source)

ii) Dans l'activité B : dans la méthode public void **onCreate**(Bundle savedInstanceState) :

- ◆ récupérer l'Intent : getIntent ()
- ◆ en extraire le Bundle : getExtras ()
- ◆ après analyse, envoyer un code de résultat avec

```
public final void setResult (int resultCode)
```

iii) Dans l'activité A ( $\leftarrow$ ) : ajouter la méthode

```
protected void onActivityResult (int requestCode, int resultCode, Intent data)
```

dans laquelle on récupère le code renvoyé par l'activité B quand elle s'est terminée.

## 21.5 Démarrage implicite

Si il s'agit de faire démarrer une autre application, on utilisera plutôt le constructeur :

```
public Intent (String action, Uri uri)
```

où l'objet **URI** (du package java.net) sera probablement créé au moyen de la méthode statique de cette classe :

```
public static Uri parse (String urlString)
```

On écrira donc quelque chose du genre :

```
String url = "http://www.google.be";
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
startActivity(intent);
```

Bien sûr, on aura remarqué qu'on ne désigne plus ainsi un composant applicatif bien précis : vu la constante ACTION\_VIEW, on demande simplement à Android de visualiser ce qui correspond à l'URI : en clair, Android va tenter de chercher une application s'étant définie

---

comme capable de répondre à l'action ACTION\_VIEW, donc probablement le browser inclus de base dans le téléphone. L'appel est donc bien implicite : *l'application à ouvrir est déterminé selon la fonctionnalité demandée.*

Un autre exemple :

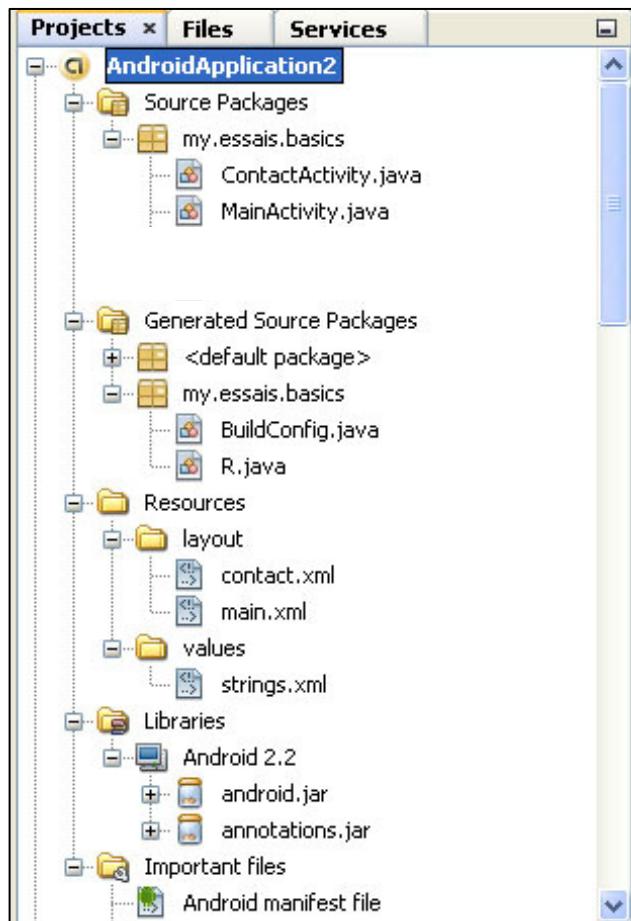
```
String numero_a_appeler = "tel:0485682465";
Intent inTel = new Intent(Intent.ACTION_DIAL, Uri.parse(numero_a_appeler));
startActivity(inTel);
```

## 21.6 Dialogue entre deux activités par intent

Nous allons entamer un projet suivant le scénario suivant. Une première activité **MainActivity** demande l'introduction d'un login. Un appui sur un bouton permet d'envoyer par intent ce login à une deuxième activité **ContactActivity**. Celle-ci s'adressera ensuite à un service **RandomService** qui émet un nombre aléatoire dans un toast - mais nous en reparlerons au paragraphe suivant consacré aux services.



Le projet ressemble à ceci



tandis que le fichier manifeste déclare essentiellement les composants de l'application, pour l'instant les deux activités :

### AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="my.essais.basics"
 android:versionCode="1"
 android:versionName="1.0">
 <application android:label="@string/app_name" >
 <activity android:name="MainActivity">
 android:label="@string/app_name">
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
 </activity>
 <activity android:name="ContactActivity">
 android:label="@string/suite_name">
 <intent-filter>
 <action android:name="android.intent.action.DEFAULT" />
```

```
<category android:name="android.intent.category.DEFAULT" />
</intent-filter>
</activity>
</application>
</manifest>
```

Les fichiers de ressources (main.xml, contact.xml et strings.xml) ne présentent aucun intérêt particulier, si ce n'est que contact.xml déclare la méthode de traitement de l'appui sur le bouton "Nombre aléatoire" :

```
...
<Button
 android:id="@+id/boutonNA"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Nombre aléatoire"
 android:onClick="onButtonClick" />
...
```

Passons donc à la programmation proprement dite. La première activité **MainActivity** récupère le login de son GUI et envoie à la deuxième activité, quand on appuie sur le bouton "Ok", un intent contenant ce login :

### MainActivity.java

```
package my.essais.basics;

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

public class MainActivity extends Activity
{
 private EditText ZENom;
 private Button BLogin;
 /** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState)
 {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);
 this.ZENom = (EditText)this.findViewById(R.id.editNom);
 BLogin = (Button)this.findViewById(R.id.boutonLogin);
 BLogin.setOnClickListener(new View.OnClickListener()
 {
```

```
public void onClick(View v)
{
 BLogin.setText("** C'est parti ! **");
 Intent suite = new Intent(MainActivity.this, ContactActivity.class);
 suite.putExtra("login", ZENom.getText().toString());
 startActivity(suite);
}
);
}
}
```

La deuxième activité **ContactActivity** récupère le login au moyen de la méthode `getExtras()` de l'intent qui l'a mise en service :

### ContactActivity.java

```
package my.essais.basics;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
//import android.content.ServiceConnection;
import android.os.Bundle;
//import android.os.IBinder;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
//import my.essais.basics.RandomService.LocalBinder;

/**
 *
 * @author Vilvens
 */
public class ContactActivity extends Activity
{
 private TextView login;

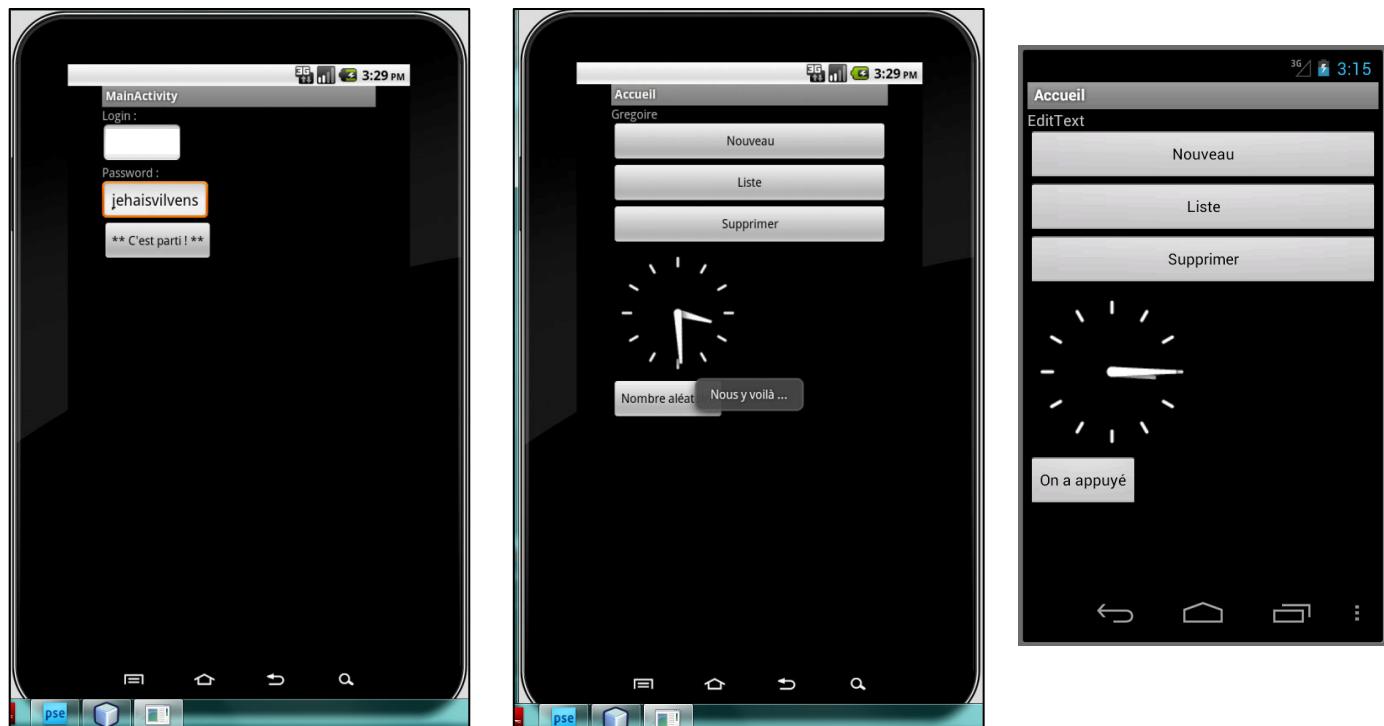
 @Override
 public void onCreate(Bundle icicle)
 {
 super.onCreate(icicle);
 setContentView(R.layout.contact);
 login = (TextView)this.findViewById(R.id.labelLogin);
 login.setText(this.getIntent().getExtras().get("login").toString());
 }
}
```

```
@Override
public void onStart()
{
 super.onStart();
 login = (TextView)this.findViewById(R.id.labelLogin);
 login.setText(this.getIntent().getExtras().get("login").toString());

 Toast.makeText(this, "Nous y voilà ...", Toast.LENGTH_LONG).show();
}

public void onClick(View v)
{
 Button BNOMBRE = (Button)this.findViewById(R.id.boutonNA);
 BNOMBRE.setText("On a appuyé");
}
}
```

On peut constater que les deux activités ont communiqué :



A suivre ... (pour le service)

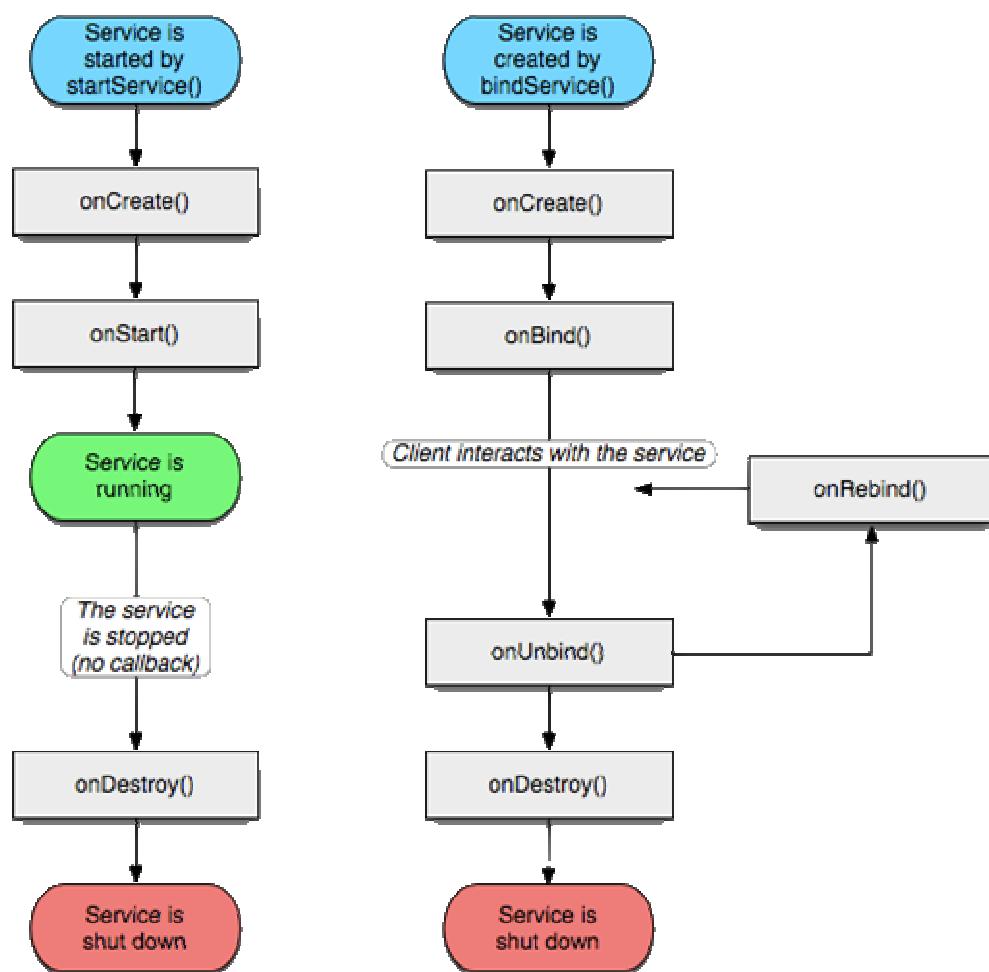
## 22. Les services

### 22.1 Le cycle de vie d'un service Android

Un service diffère d'une activité par le fait qu'il attend que l'on fasse appel à lui :

- ◆ il peut être démarré en cas de besoin au moyen de la méthode `startService()`, puis arrêté (par lui-même ou par un autre composant); on parle encore d'un "**service local**";
- ◆ il peut se mettre en attente d'une connexion établie par un autre composant (au moyen de la méthode `bindService()`), composant qui l'utilisera en utilisant les méthodes d'un interface que le service a défini et rendu public; on parle alors d'un "**service distant**".

Schématiquement, la documentation Android décrit leur cycle de vie de la manière suivante :



### 22.2 Un service local musical

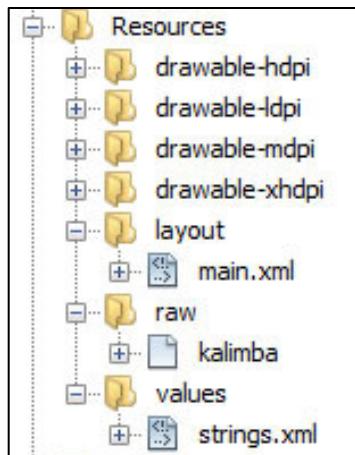
Exemple classique, nous allons implémenter un service local qui lance ou arrête la lecture d'un fichier mp3 (appelons-le finement "ServiceMusic"). Pour cela, nous allons utiliser la classe **MediaPlayer** (du package `android.media`) qui permet une gestion très poussée des séquences musicales et vidéo. Nous allons ici l'utiliser de manière élémentaire dans le service qui sera utilisé par une simple activité disposant de deux boutons "Ecouter" et "Arreter"



(appelons-la "ActivityMusic"). Un objet player est créé au moyen de la factory :

```
public static MediaPlayer create (Context context, Uri uri)
```

Le deuxième paramètre correspond à la source musicale ou vidéo. Pour nous, ce sera un simple fichier MP3 placé dans le répertoire /res/raw (attention : sans son extension ".mp3") :



On peut ensuite paramétriser ce player. Nous utiliserons simplement :

```
public void setLooping (boolean looping)
```

pour empêcher une lecture en boucle. Démarrage et arrêt se programment simplement au moyen des méthodes :

```
public void start ()
public void stop ()
```

Le service en lui-même hérite de la classe de base **Service**. Dans ce cas, il travaille au sein du thread de l'activité qui l'a appelé (ce qui n'est pas recommandé en fait pour une lecture de MP3 ou une attente de communication réseau - mais passons ici). La seule méthode qui doit impérativement être redéfinie est

```
public abstract IBinder onBind (Intent intent)
```

mais ceci ne présente de réel intérêt que pour un service distant. Par contre, les méthodes suivantes reflètent bien le cycle de vie minimal d'un service lancé localement:

```
public void onCreate ()
public void onDestroy ()
```

(remarquons tout de même que si le système détruit le service à cause d'un manque de mémoire, cette méthode ne sera pas exécutée)

et

```
public void onStart (Intent intent, int startId)
//This method was deprecated in API level 5.
public int onStartCommand (Intent intent, int flags, int startId)
```

---

On peut préciser les deux derniers paramètres de cette dernière méthode :

\* int Flags : le flag donne des informations complémentaires selon sa valeur :

- ◆ 0 : rien
- ◆ START\_FLAG\_REDELIVERY : si l'intent a déjà été délivré mais on le délivre à nouveau car le service a été interrompu.
- ◆ START\_FLAG\_RETRY : on redémarre le service après qu'il ait été interrompu de façon anormale.

\* int startId : permet d'identifier le lancement (1 si le service a été lancé une fois, 2 si on l'a lancé deux fois, etc...).

Pour la valeur de retour, elle doit être une constante qui déterminera le comportement du service après son exécution :

- ◆ START\_NOT\_STICKY : si le système tue le service, il ne sera pas recréé (il faudra de nouveau utiliser la méthode startService(intent i)).
- ◆ START\_STICKY : si le système tue le service, il sera recréé mais l'intent de la méthode vaudra null.
- ◆ START\_REDELIVER\_INTENT : si le système tue le service, il sera recréé et le paramètre intent sera identique à l'intent précédent.

La méthode de réponse au démarrage fait bien sur allusion au lancement du service qui se fera avec l'implémentation par le Context (donc l'activité) de

public abstract ComponentName **startService** (Intent service),

l'arrêt se commandant avec l'implémentation de :

public abstract boolean **stopService** (Intent service)

Le service s'écrira donc simplement:

### ServiceMusic.java

```
package my.mesapplications.services;
```

```
import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;
import android.widget.Toast;
```

```
public class ServiceMusic extends Service
```

```
{
```

```
 private MediaPlayer MP;
 private String message;
```

```
 @Override
```

```
 public IBinder onBind(Intent intent) //pour les services distants
```

```
{
```

```

 return null;
 }

 @Override
 public void onCreate()
 {
 Toast.makeText(this, "Creation", Toast.LENGTH_SHORT).show();
 MP = MediaPlayer.create(this, R.raw.kalimba); //chargement du fichier mp3
 MP.setLooping(false); //ne pas écouter en boucle
 }

 @Override
 public int onStartCommand(Intent intent, int flags, int startId)
 {
 MP.start();
 message = intent.getStringExtra("msg");
 Toast.makeText(this, message + " Démarrage",
 Toast.LENGTH_SHORT).show();
 return START_NOT_STICKY;
 }

 @Override
 public void onDestroy()
 {
 Toast.makeText(this, "Arret", Toast.LENGTH_SHORT).show();
 MP.stop();
 }
}

```

tandis que l'activité sera :

### ActivityMusic.java

```

package my.mesapplications.services;

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

public class ActivityMusic extends Activity
{
 /** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState)
 {
 super.onCreate(savedInstanceState);
 }
}

```

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

Button bjouer = (Button) findViewById(R.id.b_jouer);
bjouer.setOnClickListener(
 new View.OnClickListener()
 {
 @Override
 public void onClick(View v)
 {
 Intent i = new Intent(ActivityMusic.this, ServiceMusic.class);
 String test = "Welcome in World Music";
 i.putExtra("msg", test);
 startService(i);
 }
 }
);

Button barreter = (Button) findViewById(R.id.b_arreter);
barreter.setOnClickListener(
 new View.OnClickListener()
 {
 @Override
 public void onClick(View v)
 {
 Intent i = new Intent(ActivityMusic.this, ServiceMusic.class);
 stopService(i);
 }
 }
);
}
```

L'interface graphique auquel il est fait allusion est fort simple :

### main.xml

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

<LinearLayout
 android:id="@+id/widget32"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:orientation="vertical"
 xmlns:android="http://schemas.android.com/apk/res/android">
<TextView
 android:id="@+id/invite"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Enquête musicale"
 android:layout_gravity="center_horizontal" />
```

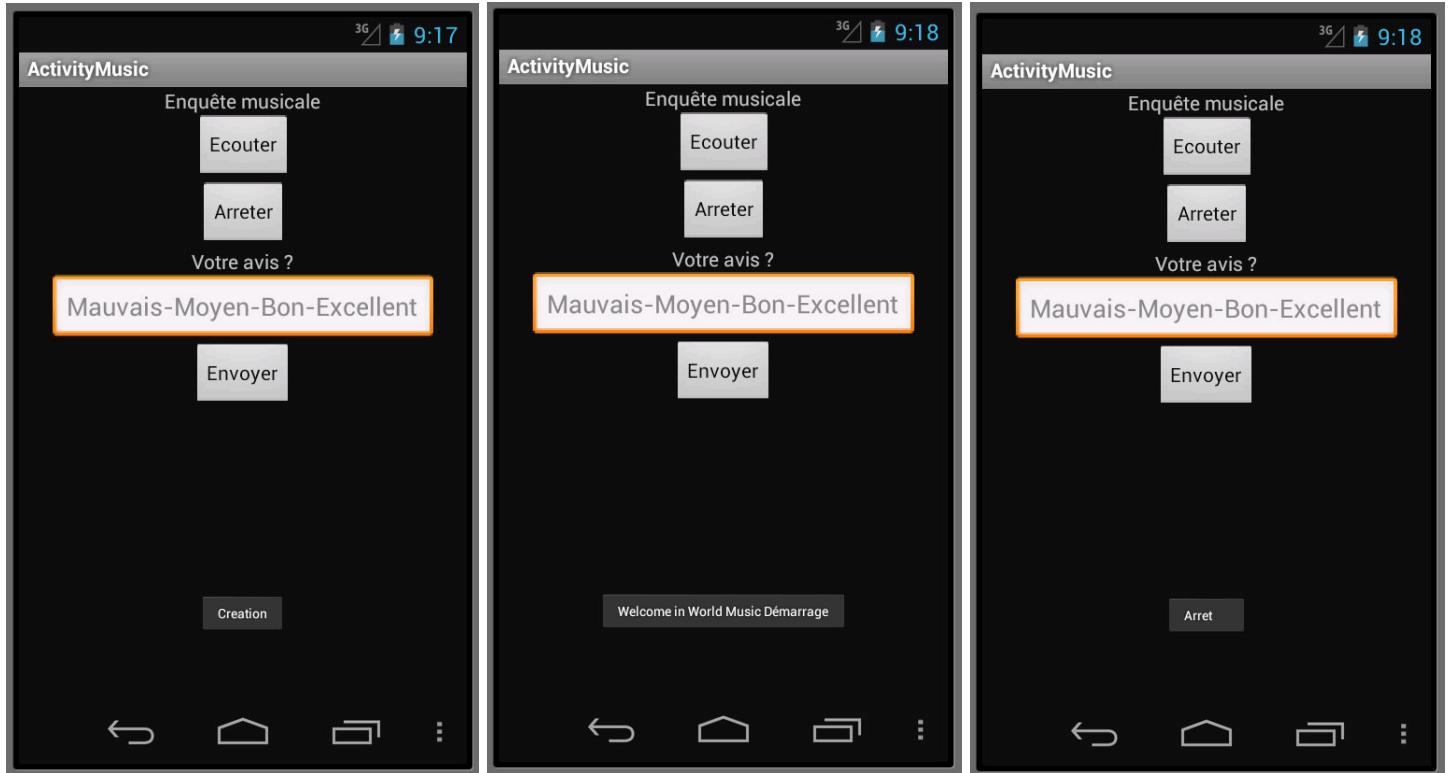
```
<Button
 android:id="@+id/b_jouer"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Ecouter"
 android:layout_gravity="center_horizontal" />
<Button
 android:id="@+id/b_arreter"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Arreter"
 android:layout_gravity="center_horizontal" />
<TextView
 android:id="@+id/question"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Votre avis ?"
 android:layout_gravity="center_horizontal" />
<EditText
 android:id="@+id/zt_avis"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:hint="@string/avis"
 android:textSize="18sp"
 android:layout_gravity="center_horizontal" />
<Button
 android:id="@+id/b_envoyer"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Envoyer"
 android:layout_gravity="center_horizontal" />
</LinearLayout>
```

avec

### strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <string name="app_name">ActivityMusic</string>
 <string name="avis">Mauvais-Moyen-Bon-Excellent</string>
</resources>
```

Résultat (imaginez la musique au 2ème écran ;-) !) :



### 22.3 Le dialogue entre une activité et un service distant

Revenons à présent à nos deux activités `MainActivity` et `ContactActivity`, cette dernière utilisant à présent un service d'émission d'un nombre aléatoire. Comme on sait, un tel service est une classe dérivée de la classe **Service** (package `android.app`). Dans notre cas, il se limite à fournir un nombre aléatoire quand on (une activité ou un service) le lui demande par un intent.

Nous pourrions traiter le service comme un "service local" à l'application, en le faisant démarrer comme une activité mais au moyen de

```
public ComponentName startService (Intent service)
```

Mais nous allons plutôt le traiter comme un "service distant", c'est-à-dire comme un service qui peut très bien avoir été lancé par un autre processus et auquel il faut donc plutôt s'"attacher". Le mécanisme est cette fois plus compliqué que dans le cas de la communication de deux activités ou services locaux : au lieu d'un simple démarrage, nous attaquerons le service au moyen de la méthode

```
public abstract boolean bindService (Intent service, ServiceConnection conn, int flags)
```

qui reçoit notamment comme paramètre un `ServiceConnection`. Qu'est-ce donc ?

En fait, l'utilisation du service est gérée par un objet implémentant l'interface **IBinder** : son rôle est d'exposer des méthodes utilisables par le client et ce sont ces méthodes qui appellent en fait les fonctionnalités du service. Ce Binder est donc en quelque sorte le "front" du service. Son utilisation est de type IPC, réel dans le cas où l'émetteur et le récepteur tournent dans deux processus distincts, seulement simulé dans le cas où l'émetteur et le récepteur tournent dans le même processus. Il sera obtenu lorsqu'on se lie au service, puisque

c'est ce que retourne la méthode  **onBind()** du service. La classe **Binder** propose une implémentation minimale de cet interface et le service définira en classe imbriquée son propre Binder. Notre version d'un Binder (disons un **LocalBinder**) se limitera à en hériter et à fournir carrément la référence du service demandé :

```
public class LocalBinder extends Binder
{
 RandomService getService()
 {
 return RandomService.this;
 }
}
```

afin que les utilisateurs du service puissent en utiliser la méthode publique `getRandomNumber()`. Nous aurions pu plutôt, par exemple, le doter d'une méthode `getRandomRange(int bi, int bs)` qui aurait utiliser la méthode du service en veillant à ce que le nombre fourni se trouve dans l'intervalle `[bi, bd]`.

Pour utiliser le service dans notre activité, il en faut une référence et donc une connexion à ce service. L'interface **ServiceConnection** représente en fait le côté client de la communication avec le service. En effet, au moyen des deux méthodes :

```
abstract void onServiceConnected(ComponentName name, IBinder service)
 appelée quand on est connecté au service
abstract void onServiceDisconnected(ComponentName name)
 appelée quand on est déconnecté au service
```

l'implémentation que nous construirons définira ce qui doit être fait une fois la connexion au service obtenue (et on utilisera pour cela l'instance de notre Binder passé en paramètre) et ce qu'il faut faire quand on se déconnecte de ce service.

L'attachement au service pourra alors se faire, comme déjà dit, au moyen de la méthode **bindService** (`Intent service, ServiceConnection conn, int flags`) qui reçoit notamment comme paramètre le ServiceConnection défini au préalable.

Résumons-nous ;-):

### RandomService.java

```
package my.essais.basics;
```

```
import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import java.util.Random;
```

```
/***
 *
 * @author Vilvens
 */
```

```
public class RandomService extends Service
{
 private final IBinder mBinder = new LocalBinder();
 private final Random mGenerator = new Random();

 public class LocalBinder extends Binder
 {
 RandomService getService()
 {
 return RandomService.this;
 }
 }

 @Override
 public IBinder onBind(Intent intent)
 {
 return mBinder;
 }

 public int getRandomNumber()
 {
 return mGenerator.nextInt(100);
 }
}
```

L'activité a été complétée pour faire un appel au service :

### ContactActivity.java

```
package my.essais.basics;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import my.essais.basics.RandomService.LocalBinder;

public class ContactActivity extends Activity
{
 private RandomService rs;
 private boolean connecteAuService;
 private TextView login;
```

```
@Override
public void onCreate(Bundle icicle)
{
 super.onCreate(icicle);
 setContentView(R.layout.contact);
 login = (TextView)this.findViewById(R.id.widget47);
 login.setText(this.getIntent().getExtras().get("login").toString());
 connecteAuService = false;
 rs=null;
}

@Override
public void onStart()
{
 super.onStart();
 login = (TextView)this.findViewById(R.id.widget47);
 login.setText(this.getIntent().getExtras().get("login").toString());

 Toast.makeText(this, "Nous y voilà ...", Toast.LENGTH_LONG).show();

 Intent intent = new Intent(this, RandomService.class);
 bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
}

@Override
protected void onStop()
{
 super.onStop();
 if (connecteAuService) {
 unbindService(mConnection);
 connecteAuService = false;
 }
}

public void onButtonClick(View v)
{
 Button BNOMBRE = (Button)this.findViewById(R.id.widget50);
 BNOMBRE.setText("On a appuyé");
 if (connecteAuService)
 {
 Toast.makeText(this, "Connecté au service ...", Toast.LENGTH_LONG).show();
 int num = rs.getRandomNumber();
 Toast.makeText(this, "number: " + num, Toast.LENGTH_LONG).show();
 }
 else
 Toast.makeText(this, "PAS connecté au service ...", Toast.LENGTH_LONG).show();
}
```

```
private ServiceConnection mConnection = new ServiceConnection()
{
 @Override
 public void onServiceConnected(ComponentName className, IBinder fService)
 {
 LocalBinder binder = (LocalBinder) fService;
 rs = binder.getService();
 connecteAuService = true;
 }
 @Override
 public void onServiceDisconnected(ComponentName arg0)
 {
 connecteAuService = false;
 }
};
```

Bien sûr, le service doit être déclaré dans le manifeste :

### AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="my.essais.basics"
 android:versionCode="1"
 android:versionName="1.0">
 <application android:label="@string/app_name" >
 <activity android:name="MainActivity" android:label="@string/app_name">
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
 </activity>
 <activity android:name="ContactActivity" android:label="@string/suite_name">
 <intent-filter>
 <action android:name="android.intent.action.DEFAULT" />
 <category android:name="android.intent.category.DEFAULT" />
 </intent-filter>
 </activity>
 <service android:name=".RandomService">
 <intent-filter>
 <action android:name="android.intent.action.DEFAULT" />
 <category android:name="android.intent.category.DEFAULT" />
 </intent-filter>
 </service>
 </application>
</manifest>
```

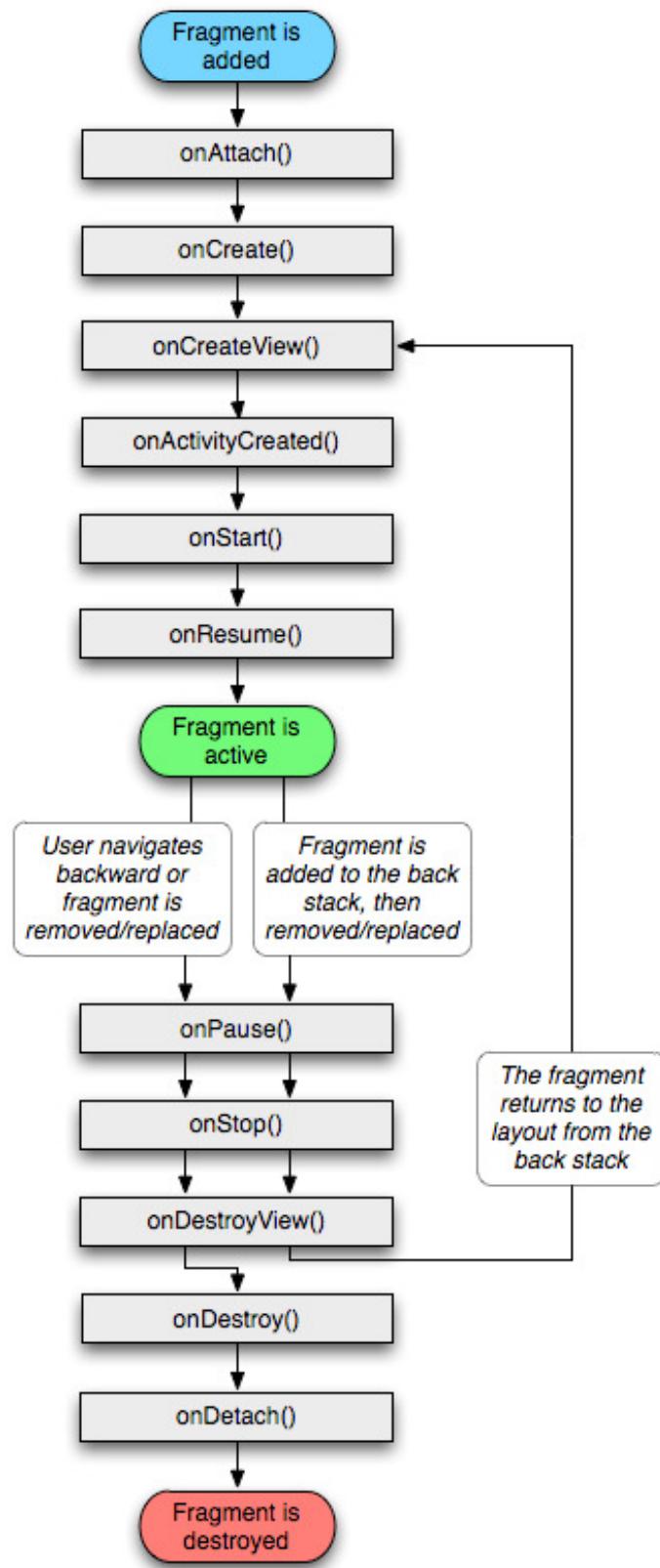
et l'application se déroule alors comme souhaité avec émission du nombre aléatoire final.

## 23. Les fragments

### 23.1 Le principe des fragments

Comme son nom l'indique, un fragment représente un morceau de GUI, utilisable dans une activité seule ou même par plusieurs. En fait, un fragment tient à la fois d'une View, puisqu'il participe à l'affichage de la vue complète de l'application, et de l'Activity, puisqu'il est capable de gérer les événements qui le concernent.

Un fragment possède donc son propre cycle de vie, celui-ci étant étroitement lié à celui de l'activité dans laquelle il est utilisé (si l'activité contenante est mise en pause, arrêtée ou détruite, tous les fragments la composant passeront dans le même état). La documentation Andoid fournit un schéma de transitions analogue à celui des activités et des services :

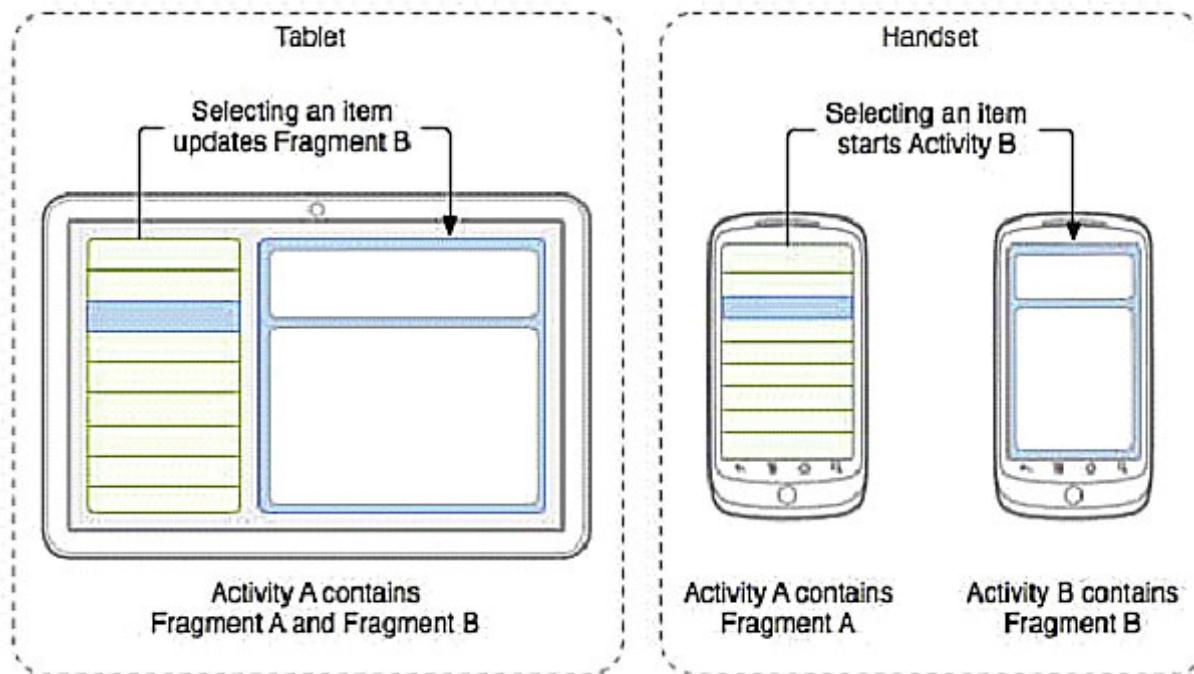


On peut remarquer que :

- ◆ une partie des méthodes sont semblables à celles d'une activité : onCreate(), onStart(), onResume(), onPause(), onStop() et onDestroy();
- ◆ d'autres méthodes ont été ajoutées au cycle de vie du fragment pour faire le lien avec celui de l'activité qui l'utilise (comme onActivityCreated()); ainsi, à chaque action effectuée lors de l'initialisation de l'activité, le fragment s'en retrouve averti et continuera sa propre initialisation (ceci évite qu'un fragment veuille s'afficher alors que l'activité contenante n'est pas encore prête);
- ◆ chaque fragment ajouté à une activité est aussi empilé sur une pile nommée "backstack" : celle-ci sert à conserver les instances des fragments selon leur ordre d'apparition, de manière à permettre ainsi à l'utilisateur de revenir en arrière au moyen de la touche "retour-annuler" du smartphone.

La question évidente : quand est-il utile de se servir de fragments ?

- ◆ bien sûr, pour gérer des parties de GUI indépendamment les unes des autres;
- ◆ mais aussi pour remplacer une architecture faisant intervenir une suite d'activités communiquant par intents par une autre architecture composée de fragments qui se succèdent au sein d'une même activité grâce à un "swipe" (mouvement du doigt sur l'écran, de gauche à droite ou de droite à gauche) ou un choix dans une ActionBar, permettant ainsi l'affichage d'un autre fragment sans pour autant changer d'activité: la navigation en est simplifiée;
- ◆ également, pour élaborer un affichage "responsive", c'est-à-dire qu'une même application adapte son affichage selon le support sur lequel elle s'exécute, par exemple un seul fragment sur un smartphone, renvoyant à un autre lors d'un clic (au sein de la même activité ou d'une autre), mais deux fragments côté à côté dans le cas d'une tablette (la place disponible sur l'écran des tablettes le permettant).



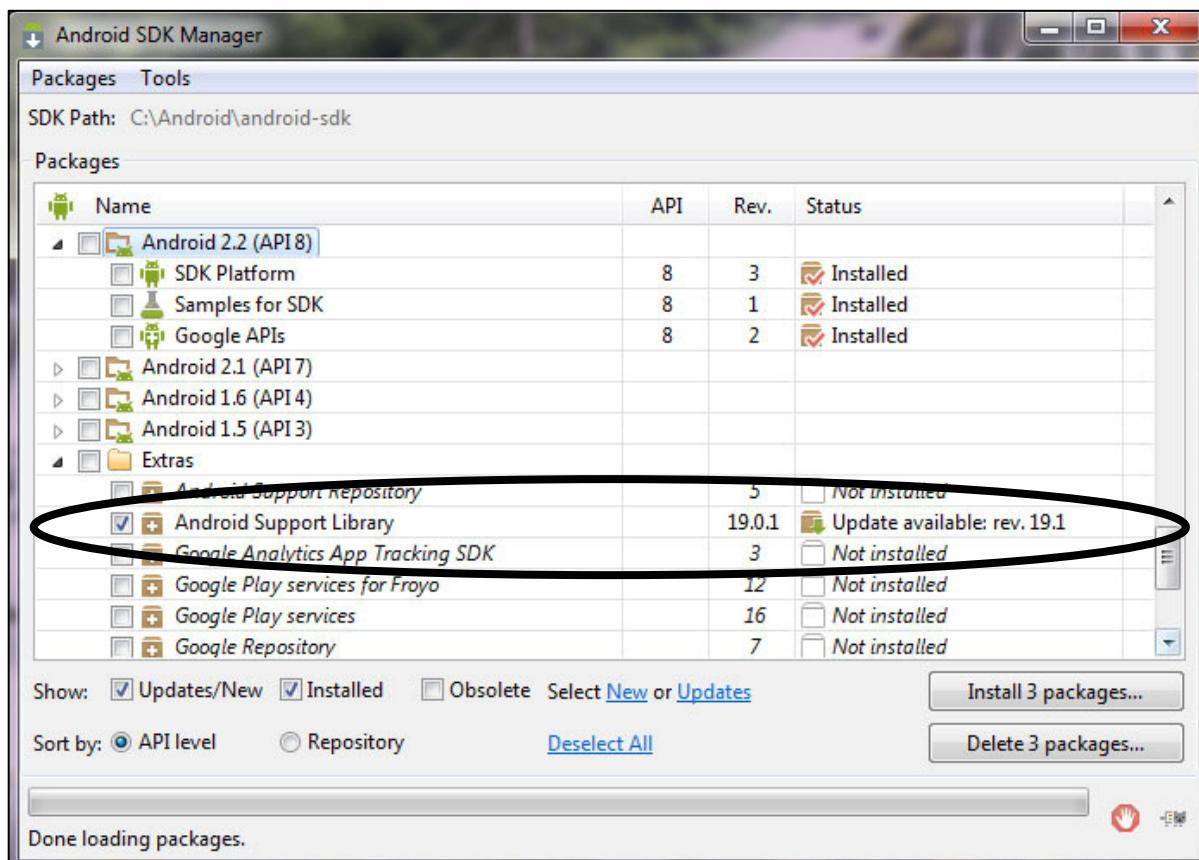
Une application comme GMail permet bien de se représenter le fonctionnement des fragments :

- ◆ un premier fragment montre une liste de conversations;
- ◆ la sélection de l'une d'entre elles affiche son contenu dans un deuxième fragment;
- ◆ une nouvelle conversation est initiée au moyen d'un troisième fragment.

Dans le cas de notre application pour médecins, on imagine facilement qu'un fragment serait chargé de l'encodage d'une consultation tandis que l'autre contiendrait la liste de ces consultations (ou des patients correspondant).

### **23.2 Une librairie de support des fragments**

Les fragments ne sont apparus qu'à partir de la version 3.0 d'Android, faisant que les smartphones tournant sur une version antérieure de l'OS ne peuvent les utiliser de manière native. Mais il existe une **Android Compatibility Library** (ACL), ou encore "Support Library" nommée **v4** (le jar android-support-v4.jar dans le répertoire \extras\android\support\v4 du répertoire du SDK) qu'il suffit d'intégrer au projet pour ajouter aux potentialités de l'application la gestion correcte de ces fragments (*il faut copier le fichier jar manuellement dans le répertoire libs, car NetBeans a verrouillé l'onglet Libraries pour ne travailler qu'avec un seul package Android auto-consistant*). Mais au préalable, cette librairie doit être installée, ce que l'on peut vérifier avec le SDK Manager :

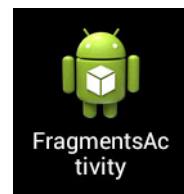


Afin d'assurer la compatibilité la plus large, nous utiliserons cette librairie au lieu de la librairie native (ce qui se remarquera par des imports du type "import android.support.v4.app.Fragment") - il n'existe que quelques différences minimes entre ces deux librairies.

### 23.3 Le développement d'une application à fragments statiques

L'exemple classique est celui d'une activité à deux fragments, une action sur le premier faisant passer sur le deuxième (l'équivalent sans fragments serait une première activité permettant de passer à une deuxième activité au moyen d'un intent).

Pour fixer les idées, et à la suite des multiples tutoriaux que l'on trouve sur le sujet, disons que le deuxième fragment affiche un texte tandis que le premier permet d'encoder avec en bonus une barre de défilement pour situer l'importance de ce texte.



Du point de vue programmation, l'activité contenant des fragments doit hériter de la classe **FragmentActivity** (package android.support.v4.app) afin de disposer des fonctionnalités de gestion de ces fragments. Une classe fragment doit hériter quant à elle d'une classe **Fragment** (même package).

Si donc notre seule activité se résume à ceci :

#### **FragmentsActivity.java** (version 1)

```
package my.mesapplications.newmedecin;

import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

public class FragmentsActivity extends FragmentActivity
{
 /** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState)
 {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.vue_fragments);
 }
}
```

avec comme fichier définissant son GUI :

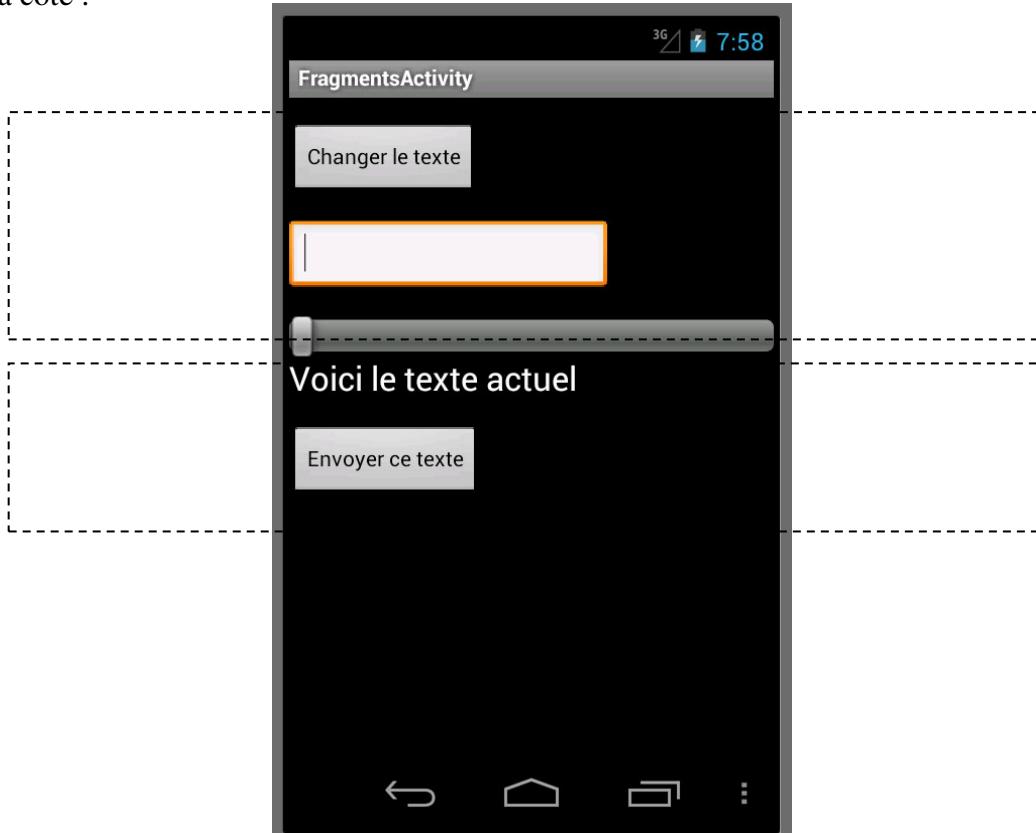
#### **vue\_fragments.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 tools:context=".FragmentsActivity" >

 <fragment
 android:id="@+id/premier_fragment"
 android:name="my.mesapplications.newmedecin.PremierFragment"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_alignParentLeft="true"
 android:layout_alignParentTop="true"
 tools:layout="@layout/premier_fragment" />
```

```
<fragment
 android:id="@+id/deuxième_fragment"
 android:name="my.mesapplications.newmedecin.DeuxiemeFrgment"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_alignParentLeft="true"
 android:layout_centerVertical="true"
 tools:layout="@layout/deuxième_fragment" />
</RelativeLayout>
```

on obtiendra l'affichage suivant, qui bien sûr affiche les deux fragments (définis ci-dessous) côté à côté :



On remarquera, outre les tags prévisibles <fragment ...> l'attribut de tag " tools:context " qui permet de préciser à quelle activité la vue est associée (un intérêt est que l'on peut ainsi récupérer un thème, celui-ci étant lié à une activité dans le fichier manifeste, et pas à une vue).

Les deux fragments en question sont :

1) fragment d'encodage :

### **premier\_fragment.xml**

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

```
tools:context=".FragmentsActivity" >
<Button
 android:id="@+id/b_change_texte"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_centerHorizontal="true"
 android:layout_marginTop="17dp"
 android:text="Changer le texte" />
<EditText
 android:id="@+id/tf_nouveau_texte"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_centerHorizontal="true"
 android:layout_marginTop="16dp"
 android:ems="10"
 android:inputType="text" >
 <requestFocus />
</EditText>
<SeekBar
 android:id="@+id/bar_avancement"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_alignParentLeft="true"
 android:layout_below="@+id/editText1"
 android:layout_marginTop="14dp" />
</LinearLayout>
```

avec

### PremierFragment.java

```
package my.mesapplications.newmedecin;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class PremierFragment extends Fragment
{
 @Override
 public View onCreateView(LayoutInflater inflater, ViewGroup container,
 Bundle savedInstanceState)
 {
 // Inflate the layout for this fragment
 View view = inflater.inflate(R.layout.premier_fragment, container, false);
 return view;
 }
}
```

On remarquera le paramètre **LayoutInflater** (package android.view) : analogue au MenuInflater déjà rencontrée, il remplit une View des éléments graphiques lus dans un fichier XML au moyen de sa méthode

```
public View inflate (int resource, ViewGroup root, boolean attachToRoot)
```

le dernier paramètre à false exprimant que les paramètres de layout sont ceux de la vue mère, mais que la ressource n'est pas intégrée à celle-ci.

Un LayoutInflater n'est jamais instancié directement mais plutôt obtenu au moyen de la méthode:

```
public LayoutInflater getLayoutInflater ()
```

qui fournit le LayoutInflater qui a déjà été associé au contexte et qui a été configuré pour le mobile hôte.

2) fragment d'affichage :

### deuxieme\_fragment.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:orientation="vertical"
 tools:context=".FragmentsActivity" >
 <TextView
 android:id="@+id/tf_nouveau_texte"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_centerHorizontal="true"
 android:layout_centerVertical="true"
 android:text="Voici le texte actuel"
 android:textAppearance="?android:attr/textAppearanceLarge" />
 <Button
 android:id="@+id/b_envoyer"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_below="@+id/seekBar1"
 android:layout_centerHorizontal="true"
 android:layout_marginTop="17dp"
 android:text="Envoyer ce texte" />
</LinearLayout>
```

avec

### DeuxiemeFragment.java

```
package my.mesapplications.newmedecin;

import android.os.Bundle;
import android.support.v4.app.Fragment;
```

```

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class DeuxiemeFragment extends Fragment
{
 @Override
 public View onCreateView(LayoutInflater inflater, ViewGroup container,
 Bundle savedInstanceState)
 {
 View view = inflater.inflate(R.layout.deuxieme_fragment,
 container, false);

 return view;
 }
}

```

### 23.4 Le principe des fragments dynamiques

Lorsqu'il s'agit de gérer les fragments de manière dynamique, c'est-à-dire de passer d'un fragment à un autre, on utilise un contrôleur de fragments, implémentation de la classe abstraite **FragmentManager** que l'on obtient au moyen de la méthode `getSupportFragmentManager()`. Quand un tel objet veut modifier les fragments affichés, il crée une transaction au moyen de la méthode



`public abstract FragmentTransaction beginTransaction ()`

L'objet implementant la classe abstraite **FragmentTransaction** possède une série de méthodes permettant diverses opérations. Tout d'abord :

`public abstract FragmentTransaction add (int containerViewId, Fragment fragment)`  
`public abstract FragmentTransaction show (Fragment fragment)`  
`public abstract FragmentTransaction hide (Fragment fragment)`

ce qui donnera dans la méthode `onCreate()` de notre activité :

```

public void onCreate(Bundle savedInstanceState)
{
 super.onCreate(savedInstanceState);
 setContentView(R.layout.vue_fragments);

 fragment1 = new PremierFragment();
 fragment2 = new DeuxiemeFragment();
 FragmentManager fm = getSupportFragmentManager();
 FragmentTransaction ft = fm.beginTransaction();
 ft.add(R.id.container_fragments, this.fragment1);
 ft.add(R.id.container_fragments, this.fragment2);
 ft.show(this.fragment1);
 ft.hide(fragment2);
 ft.commit();
}

```

Bien sûr, l'appel de la méthode

```
public abstract int commit()
```

rend les changements effectifs. On peut bien sûr à tout moment créer une nouvelle transaction pour modifier le "paysage".

Pour que l'appui sur le bouton provoque le passage au deuxième fragment, on utilise d'autres méthodes de l'objet FragmentTransaction :

```
public abstract FragmentTransaction addToBackStack (String name)
```

pour qu'un retour en arrière restaure l'ancienne situation - en quelque sorte, la transaction est mémorisée

```
public abstract FragmentTransaction replace (int containerViewId, Fragment fragment)
```

pour remplacer le fragment actif par le fragment spécifié

```
public abstract FragmentTransaction setCustomAnimations (int enter, int exit)
```

pour spécifier le type de transition - les entiers désignent des ressources sous forme de fichier XML comme

### slide\_in\_left.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android" >
 <translate
 android:duration="700"
 android:fromXDelta="-100%"
 android:toXDelta="0%" >
 </translate>
</set>
```

On peut donc imaginer que l'appui sur le bouton faisant passer d'un fragment à un autre soit :

```
public void onClick(View v)
{
 FragmentManager fm = getSupportFragmentManager();
 FragmentTransaction ft = fm.beginTransaction();

 ft.setCustomAnimations(android.R.anim.slide_in_left, android.R.anim.slide_out_right);
 ft.replace(R.id.container_fragments, fragment2);
 ft.addToBackStack(null);

 ft.commit();
}
```

A remarquer qu'il convient de définir les animations avant l'empilage - sinon, elles ne sont pas mémorisées !

---

### 23.5 Fragments, ActionBar et communication entre fragments

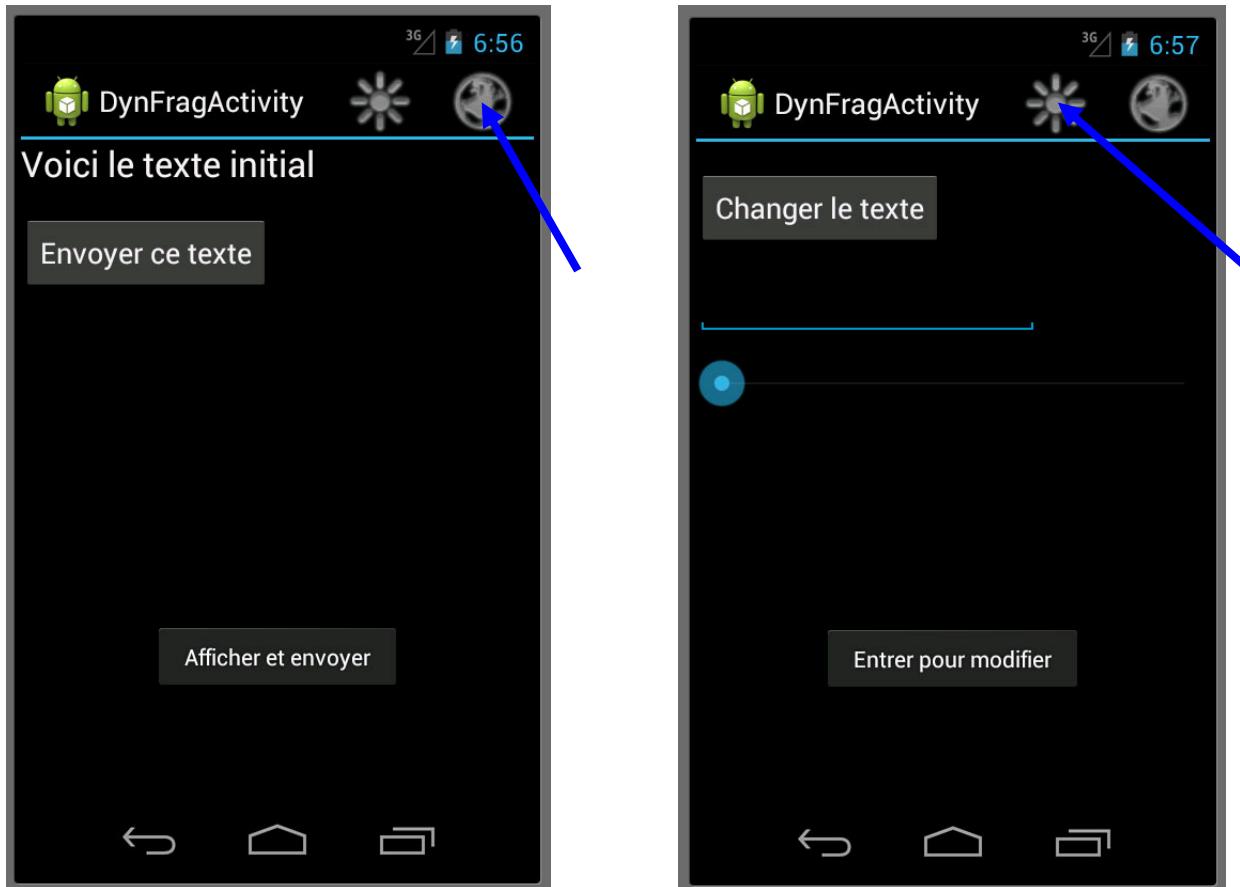
Mais l'utilisation la plus classique des fragments est celle qui utilise une ActionBar permettant de choisir le fragment actif :

```
menu/menu_fragments.xml
<menu xmlns:android="http://schemas.android.com/apk/res/android">

<item android:id="@+id/entrer"
 android:title="Entrer"
 android:showAsAction="ifRoom"
 android:icon="@drawable/sun" >
</item>

<item android:id="@+id/afficher"
 android:title="Afficher"
 android:showAsAction="ifRoom"
 android:icon="@drawable/globe" />
</menu>
```

et donc visuellement :



La gestion des fragments est alors dévolue entièrement à l'activité durant l'ensemble de l'exécution de l'application : il n'y a donc pas de description statique XML de l'interface de l'activité :

### layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
 xmlns:android="http://schemas.android.com/apk/res/android"
 android:id="@+id/frame_contener"
 android:layout_width="match_parent"
 android:layout_height="match_parent">
</FrameLayout>
```

L'activité en elle-même met donc en place le fragment choisi par l'intermédiaire de l'ActionBar :

### DynFragActivity.java

```
package my.mesapplications.fragments;

import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.support.v4.app.FragmentManager;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.Toast;

public class DynFragActivity extends FragmentActivity
{
 private FragmentEntrer form;
 private FragmentAfficher view;

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

 this.form = new FragmentEntrer(); this.view = new FragmentAfficher();

 FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
 ft.add(R.id.frame_contener, this.form, "FormFrag"); // noms internes (tags)
 ft.add(R.id.frame_contener, this.view, "ShowFrag");
 ft.show(this.form); ft.hide(this.view);
 ft.commit();
 }

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

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
 FragmentTransaction ft = this.getSupportFragmentManager().
 beginTransaction();

 switch (item.getItemId())
 {
 case R.id.entrer:
 ft.hide(this.view);
 ft.show(this.form);
 ft.commit();
 Toast.makeText(DynFragActivity.this, "Entrer pour
 modifier",Toast.LENGTH_SHORT).show();
 return true;

 case R.id.afficher:
 ft.hide(this.form);
 ft.show(this.view);
 ft.commit();
 Toast.makeText(DynFragActivity.this, "Afficher et
 envoyer",Toast.LENGTH_SHORT).show();
 return true;
 }

 ft.commit();
 return super.onOptionsItemSelected(item);
}
}
```

Les deux fragments sont semblables aux précédents du point de vue XML :

```
layout/fragment_entrer.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
...
<Button
 android:id="@+id/b_change_texte"
...
 android:text="Changer le texte" />
<EditText
 android:id="@+id/tf_nouveau_texte"
...
</EditText>

<SeekBar
 android:id="@+id/bar_avancement"
...
</LinearLayout>
```

### layout/fragment\_afficher.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
...
<TextView
 android:id="@+id/tf_texte"
...
 android:text="@string/texteinitial"
 android:textAppearance="?android:attr/textAppearanceLarge" />
<Button
 android:id="@+id/b_envoyer"
...
 android:text="Envoyer ce texte" />
</LinearLayout>
```

avec simplement :

### values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <string name="app_name">DynFragActivity</string>
 <string name="texteinitial">Voici le texte initial</string>
</resources>
```

Du point de vue du code Java, le premier fragment va bien sûr traiter l'appui sur le bouton de changement de texte. Pour cela, il lui faut

- ◆ soit la référence du deuxième fragment (où l'on changera la chaîne affichée),
- ◆ soit la référence de l'activité container qui se chargera de modifier ensuite l'autre fragment.

Nous allons adopter cette deuxième solution en utilisant la méthode

```
public void onAttach (Activity activity)
```

appelée la première fois qu'un fragment est attaché à une activité par le FragmentManager. Nous pourrons ainsi initialiser une variable membre du fragment (appelons-là "*cible*") qui désignera cette activité. Comme, dans une autre utilisation, il pourrait s'agir d'autre chose que d'une activité, nous allons plutôt déclarer que cette variable membre implémente un interface SetterString qui déclare une méthode setValue() :

### SetterString.java

```
package my.mesapplications.fragments;

public interface SetterString
{
 public void setValue(String value);
}
```

Ainsi, notre fragment d'entrée s'écrit :

### FragmentEntrer.java

```
package my.mesapplications.fragments;

import android.app.Activity;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class FragmentEntrer extends Fragment implements OnClickListener
{
 private SetterString cible;
 private EditText nouveauTexte;

 @Override
 public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
 savedInstanceState)
 {
 View view = inflater.inflate(R.layout.fragment_entrer, container, false);

 this.nouveauTexte = (EditText) view.findViewById(R.id.tf_nouveau_texte);
 ((Button) view.findViewById(R.id.b_change_texte)).setOnClickListener(this);

 return view;
 }

 @Override
 public void onAttach(Activity activity)
 {
 super.onAttach(activity);
 try
 {
 cible = (SetterString) activity;
 }
 catch (ClassCastException e)
 {
 throw new ClassCastException(activity.toString() + " il faut implémenter
 SetterString");
 }
 }
}
```

```
public void onClick(View v)
{
 Toast.makeText((Activity)cible, "On a appuyé : " + this.nouveauTexte.getText(),Toast.LENGTH_SHORT).show();
 cible.setValue(this.nouveauTexte.getText().toString());
}
```

Le deuxième fragment est en fait alors très simple, puisqu'il lui suffit d'exposer une méthode publique permettant de modifier la chaîne affichée (appelons cette méthode updateField() ) :

### FragmentAfficher.java

```
package my.mesapplications.fragments;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class FragmentAfficher extends Fragment
{
 @Override
 public View onCreateView(LayoutInflater inflater, ViewGroup container,
 Bundle savedInstanceState)
 {
 View view = inflater.inflate(R.layout.fragment_afficher, container, false);
 return view;
 }

 public void updateField(String value)
 {
 ((TextView) getActivity().findViewById(R.id.tf_texte)).setText(value);
 }
}
```

Il reste donc à notre activité à implémenter l'interface :

### DynFragActivity.java

```
package my.mesapplications.fragments;

import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.support.v4.app.FragmentManager;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
```

```
import android.widget.Toast;

public class DynFragActivity extends FragmentActivity implements SetterString
{
 private FragmentEntrer form;
 private FragmentAfficher view;

 public void onCreate(Bundle savedInstanceState)
 {...}

 @Override
 public boolean onCreateOptionsMenu(Menu menu)
 {...}

 @Override
 public boolean onOptionsItemSelected(MenuItem item)
 {...}

 public void setValue(String value)
 {
 FragmentAfficher frag = (FragmentAfficher) this.getSupportFragmentManager().
 findFragmentByTag("ShowFrag"); // utilisation du nom interne

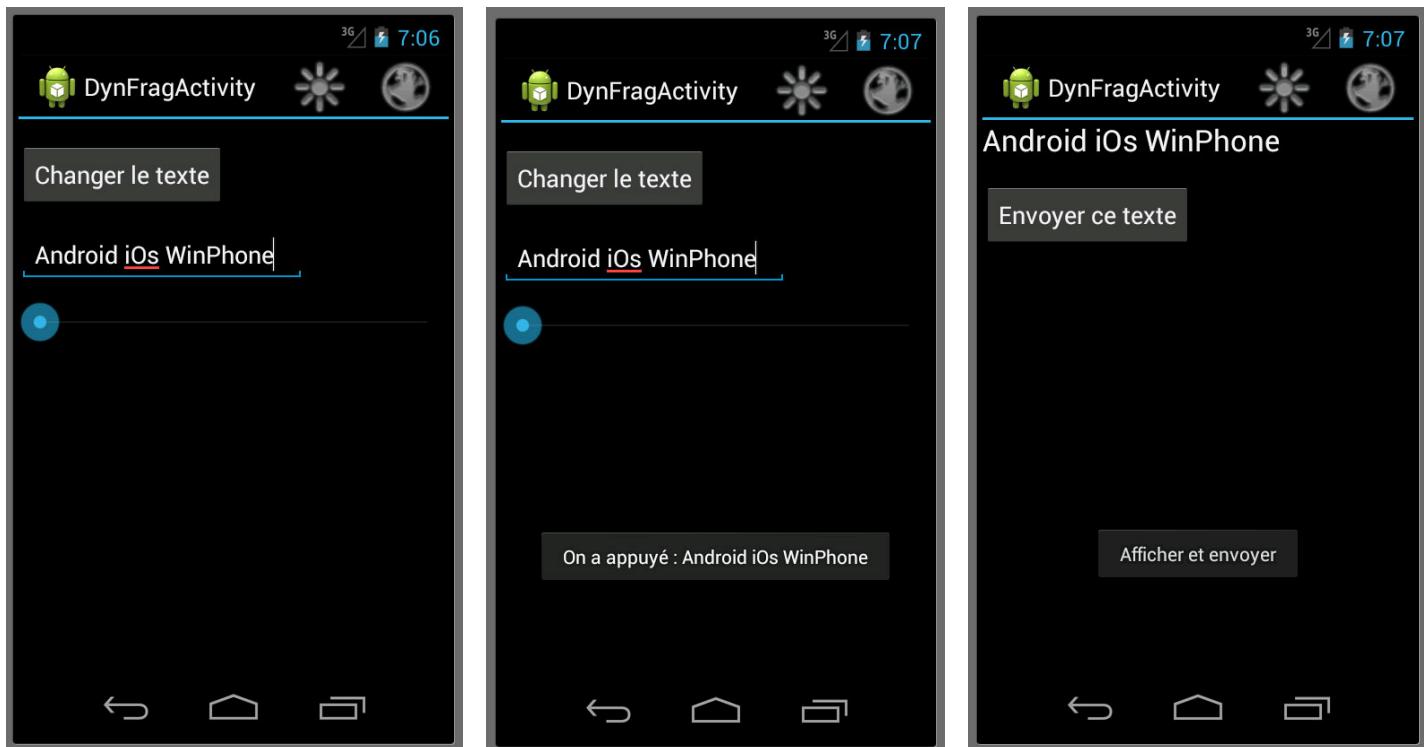
 if (frag != null) frag.updateField(value);
 else Toast.makeText(getApplicationContext(),"Pas de fragment reconnu...",Toast.LENGTH_LONG).show();
 }
}
```

Avec un AndroidManifest.xml tout à fait lambda :

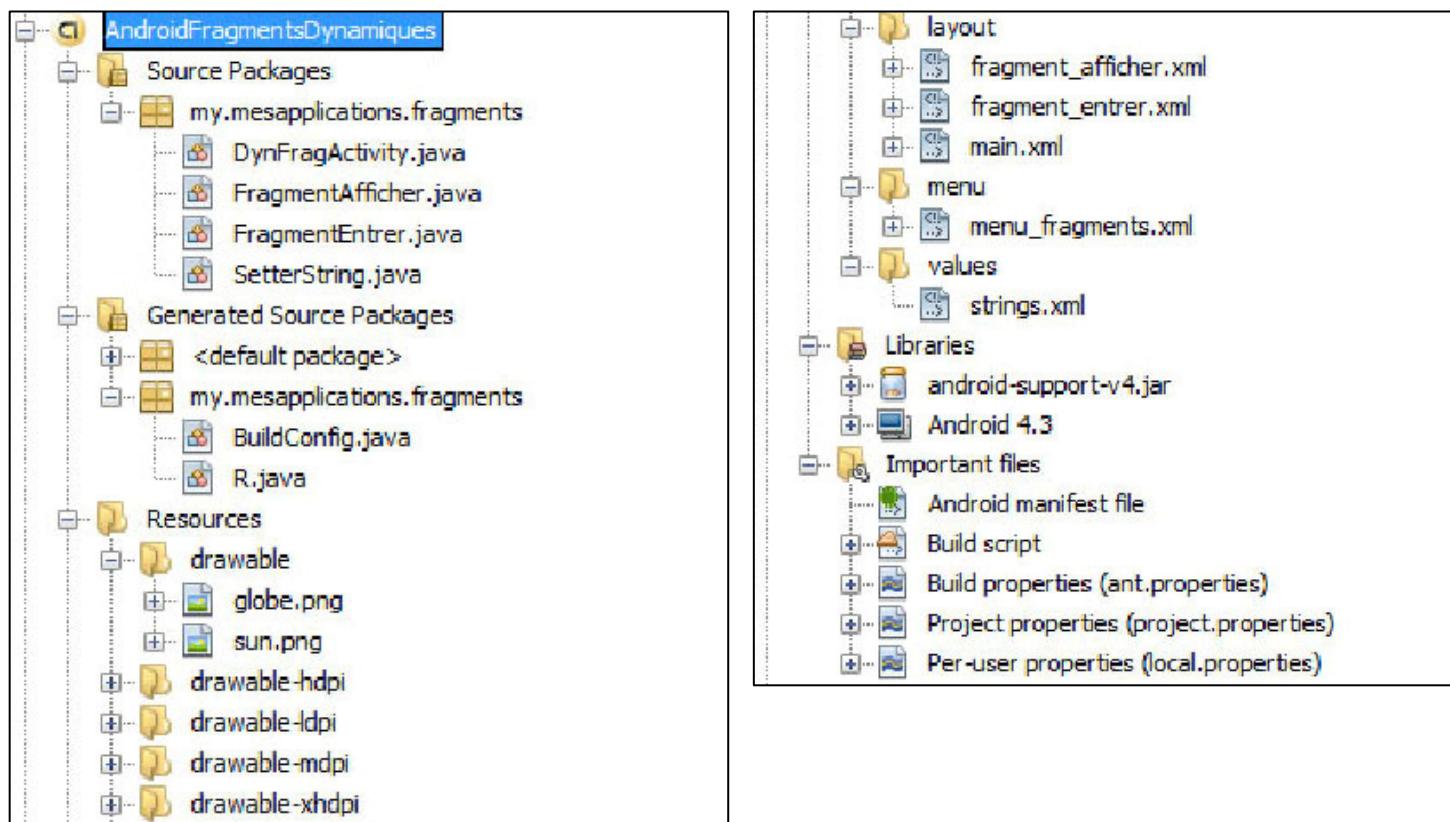
### AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="my.mesapplications.fragments"
 android:versionCode="1"
 android:versionName="1.0">
 <application android:label="@string/app_name" android:icon="@drawable/ic_launcher">
 <activity android:name="DynFragActivity"
 android:label="@string/app_name">
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
 </activity>
 </application>
 <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="11" />
</manifest>
```

on obtient :



Le projet complet se présente finalement comme suit :



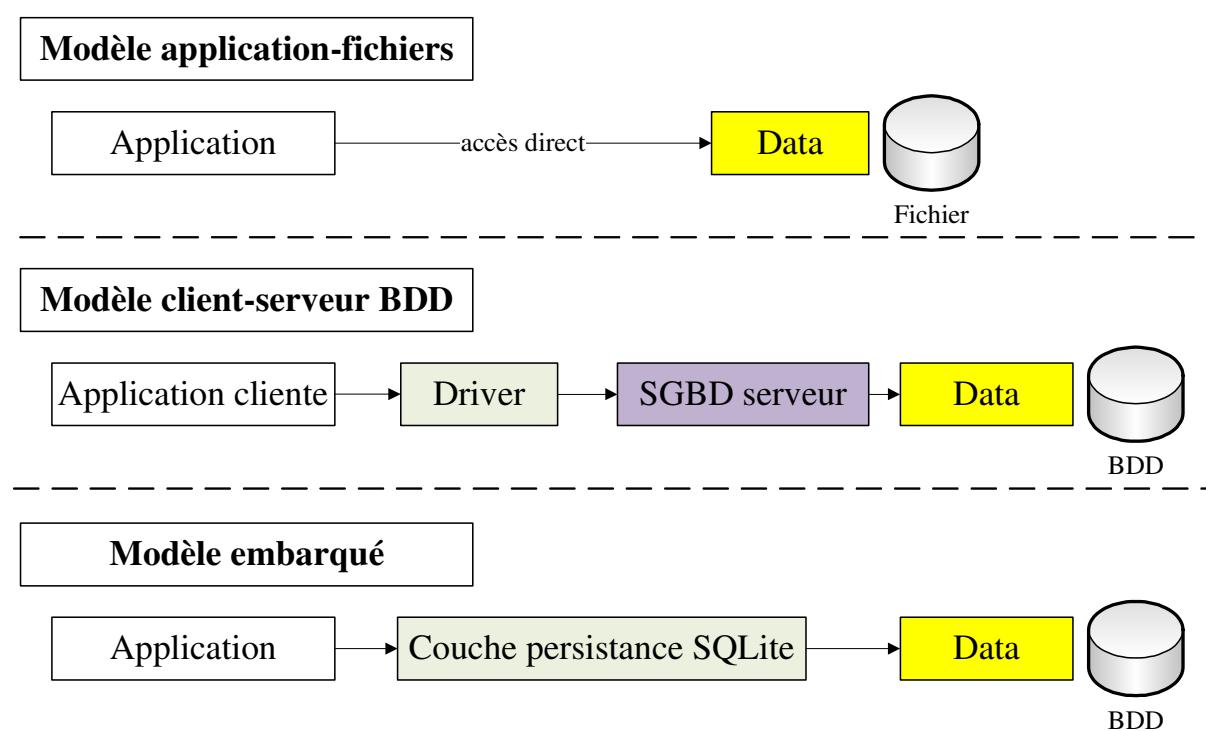
## 24. SQLite

### 24.1 Un moteur interopérable

SQLite est un moteur Open Source de bases de données relationnelles, accessible par un langage de type SQL, qui s'intègre dans une application à priori quelconque (mais pour nous, il s'agira d'une application Android) et lui permet de sauvegarder des données en local, avec une faible empreinte mémoire et des temps de réponses raisonnables. Il est développé par un SQLite Consortium qui comporte notamment Oracle, Adobe et Mozilla.



On pourrait éventuellement penser à un JDBC dédié du Java standard, mais le concept est ici différent : il n'y a pas ici de client et de serveur, mais seulement une couche de persistance évoluée appartenant au runtime et disponible pour toute application (package android.database). Schématiquement



En fait, SQLite est indépendant de la plate-forme utilisée : quand une application crée une base de données, SQLite sauve les data et meta-data dans un fichier de manière indépendante de cette plate-forme hôte. Ce fichier est sauvé dans une répertoire

<rédertoire\_données>/data/<nom\_application>/databases/<nom\_base>

le répertoire des données étant celui fourni par

`Environment.getDataDirectory()`

Dans le cas d'Android, chaque version correspond à une version bien spécifique de SQLite :

	SQLite 3 .7.11	SQLite 3.7.4	SQLite 3.6.22	SQLite3.5.9
SDK 19 – 4.4	✓	✓	✓	✓
SDK 18 – 4.3	✓	✓	✓	✓
SDK 17 – 4.2	✓	✓	✓	✓
SDK 16 – 4.1	✓	✓	✓	✓
SDK 15 – 4.0.3	✗	✓	✓	✓
SDK 14 – 4.0	✗	✓	✓	✓
SDK 13 – 3.2	✗	✓	✓	✓
SDK 12 – 3.1	✗	✓	✓	✓
SDK 11 – 3.0	✗	✓	✓	✓
SDK 10 – 2.3.3	✗	✗	✓	✓
SDK 9 – 2.3.1	✗	✗	✓	✓
SDK 8 – 2.2	✗	✗	✓	✓
SDK 7 – 2.1	✗	✗	✗	✓
SDK 4 – 1.6	✗	✗	✗	✓
SDK 3 – 1.5	✗	✗	✗	✓

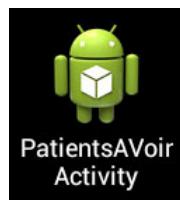
La version de SQLite qui sera utilisée dans une application est liée au SDK minimal sur lequel elle peut fonctionner : le choix est donc réalisé automatiquement au moment de la compilation.

SQLite reste un moteur "léger" et présente de ce fait quelques restrictions par rapport à un SGBD relationnel complet :

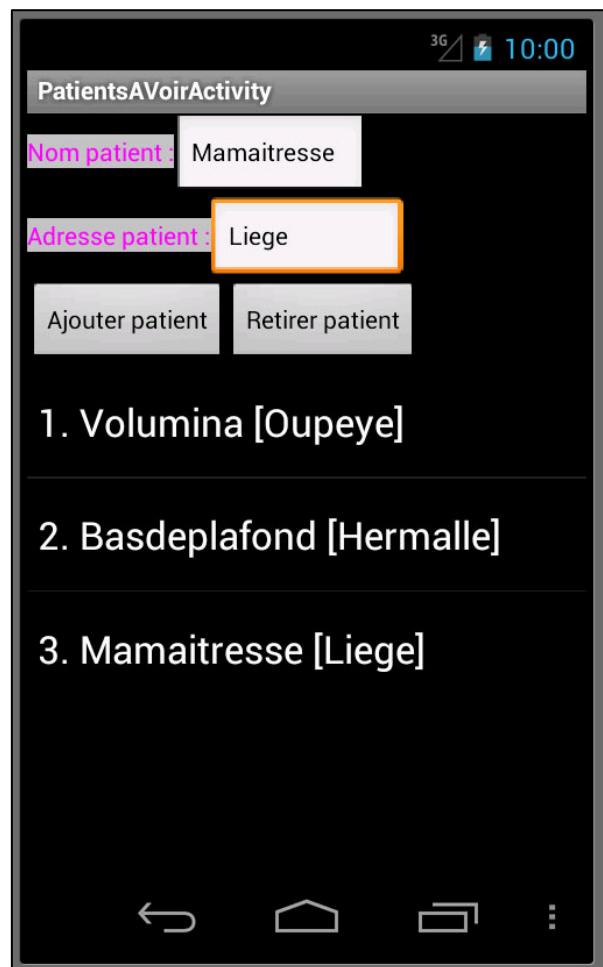
- ◆ pas de vérification de la concordance entre type de champ et type de la donnée affectée;
- ◆ seule la jointure LEFT JOIN est implémentée;
- ◆ seuls les TRIGGER de type FOR EACH ROW sont implémentés;
- ◆ les vues sont simplement read-only (pas de mise à jour à travers les vues);
- ◆ les seuls ALTER TABLE possibles sont RENAME TABLE et ADD COLUMN; le point regrettable est donc l'absence de support pour ADD CONSTRAINT;
- ◆ assez logiquement vu le contexte, GRANT et REVOKE ne sont pas supportés.

Fort heureusement, les clés étrangères sont supportées dans toutes les versions actuelles d'Android (à partir de la version 3..6.19 de SQLite).

## 24.2 Une application contexte avec ListActivity



Pour illustrer l'utilisation de SQLite, nous allons nous placer dans le contexte d'une application qui permet à un médecin d'encoder le nom des patients qu'il doit voir au fur et à mesure que ceux-ci le contacte. Très simplement, elle ressemblera à ceci :



Bien sur, le bouton "Ajouter patient" permet d'ajouter un patient dans la file tandis que "Retirer patient" permet de retirer le premier patient de la liste.

Rien d'extraordinaire à ce stade. On utilisera bien sûr une classe lambda Patient :

### **Patient.java**

```
package my.mesapplications.sqlite;

public class Patient
{
 private static int compteur=0;

 private long id;
 private String nom;
 private String adresse;

 public Patient()
 {
 nom=null; adresse=null;
 id=++compteur;
 }

 public long getId() { return id; }
```

```
public void setId(long id) { this.id = id; }
public String getNom() { return nom; }
public void setNom(String n) { this.nom = n; }
public String getAdresse() { return adresse; }
public void setAdresse(String a) { this.adresse = a; }

@Override
public String toString() { return Long.toString(id) + ". " + nom + "[" + adresse + "]"; }

public static int getCompteur() { return compteur; }
public static void setCompteur(int c) { compteur = c; }
}
```

Mais en ce qui concerne l'activité, nous allons nous servir d'une **ListActivity** (package android.app) qui présente la spécificité de posséder d'origine une ListView que l'on peut lier avec une source de données. On fait référence à cette ListView au moyen de la méthode :

```
public ListView getListView ()
```

On peut associer à cette ListView "intégrée" un adapter au moyen des méthodes

```
public ListAdapter getListAdapter ()
public void setListAdapter (ListAdapter adapter)
```

Pour que cette ListView apparaisse dans l'interface graphique (on pourrait ne pas le vouloir dès le démarrage), il suffit de la déclarer dans le fichier main.xml au moyen de

```
<ListView
 android:id="@+id/list"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:text="@string/hello" />
```

l'identifiant faisant référence à l'id "list" du package android (on pourrait y faire référence avec "android.R.id.list"). L'ensemble de notre fichier main.xml aura donc l'aspect suivant :

### main.xml

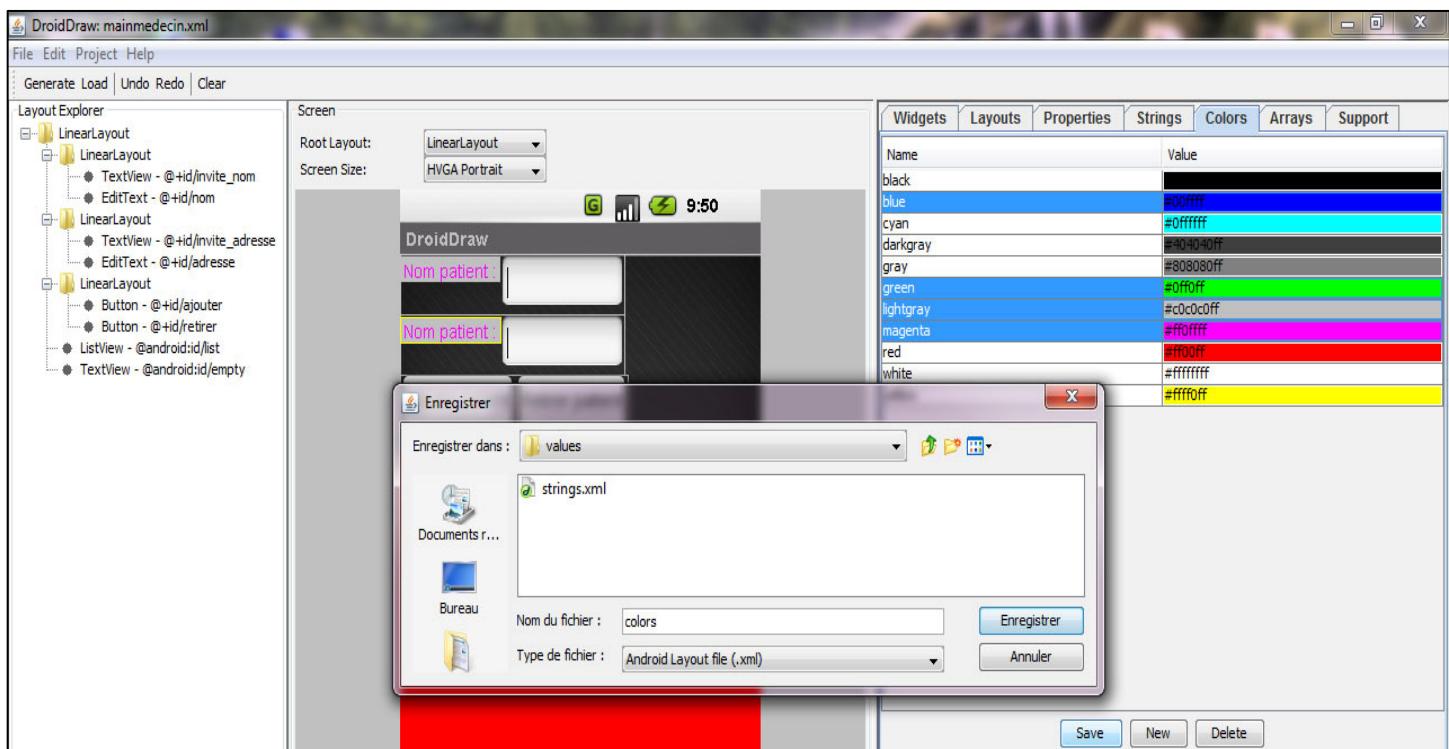
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:orientation="vertical" >

 <LinearLayout
 android:id="@+id/groupe_nom"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:orientation="horizontal">
```

```
<TextView
 android:id="@+id/invite_nom"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:background="@color/lightgray"
 android:text="Nom patient :"
 android:textColor="@color/magenta" />
<EditText
 android:id="@+id/nom"
 android:layout_width="110dp"
 android:layout_height="wrap_content"
 android:textSize="14sp" />
</LinearLayout>
<LinearLayout
 android:id="@+id/groupe_adresse"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:orientation="horizontal">
 <TextView
 android:id="@+id/invite_adresse"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:background="@color/lightgray"
 android:text="Adresse patient :"
 android:hint="où ?"
 android:textColor="@color/magenta" />
 <EditText
 android:id="@+id/adresse"
 android:layout_width="110dp"
 android:layout_height="wrap_content"
 android:textSize="14sp" />
</LinearLayout>
<LinearLayout
 android:id="@+id/groupe"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content" >
 <Button
 android:id="@+id/ajouter"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Ajouter patient"
 android:onClick="onClick"/>
 <Button
 android:id="@+id/retirer"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Retirer patient"
 android:onClick="onClick"/>
```

```
</LinearLayout>
<ListView
 android:id="@+android:id/list"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:text="@string/hello" />
<TextView android:id="@+android:id/empty"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:background="#FF0000"
 android:text="-- RIEN --"/>
</LinearLayout>
```

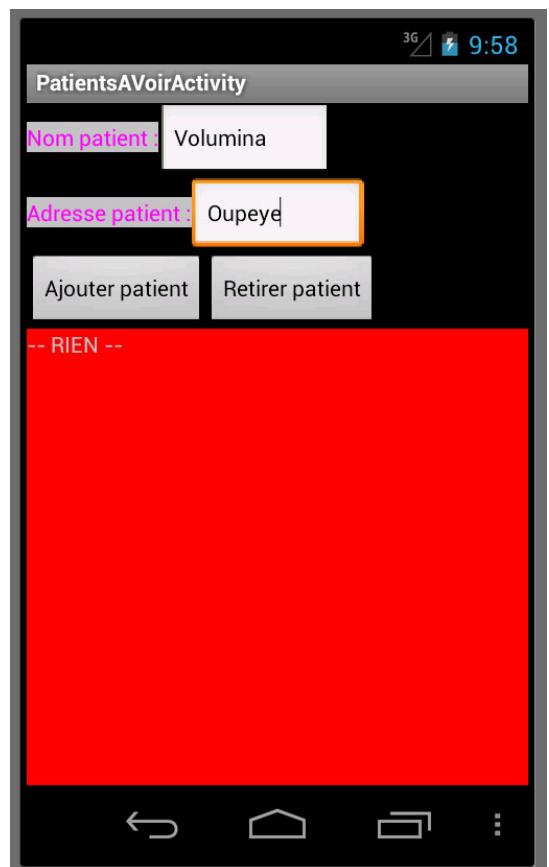
On remarquera en passant l'utilisation de couleurs pour les TextView. Celles-ci sont placées dans un fichiers colors.xml créé depuis DroiDraw :



### colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<color name="red">#ffff0000</color>
<color name="lightgray">#ffc0c0c0</color>
<color name="white">#ffffffff</color>
<color name="yellow">#ffffff00</color>
<color name="blue">#ff0000ff</color>
<color name="magenta">#ffff00ff</color>
...
</resources>
```

On remarquera encore que la ListView du package Android a été écrite pour faire usage d'un TextView dont l'id est "@**android:id/empty**" qui indiquera si la liste est vide. Résultat :



A remarquer aussi que la ListView intégrée est scollable (dès que le nombre d'items dépasse la taille d'affichage) !



Quant à notre activité, elle aura donc pour code :

### **PatientsAVoirActivity.java**

```
package my.mesapplications.sqlite;

import android.app.ListActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

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

 List<Patient> modele = new ArrayList<Patient>();
 ArrayAdapter<Patient> adapter =
 new ArrayAdapter<Patient>(this, android.R.layout.simple_list_item_1, modele);
 setListAdapter(adapter);
 }

 public void onClick(View view)
 {
 ArrayAdapter<Patient> adapter = (ArrayAdapter<Patient>) getListAdapter();
 Patient patient = new Patient();
 EditText nom = (EditText) findViewById(R.id.nom);
 patient.setNom(nom.getText().toString());
 EditText adresse = (EditText) findViewById(R.id.adresse);
 patient.setAdresse(adresse.getText().toString());

 switch (view.getId())
 {
 case R.id.ajouter: adapter.add(patient); break;
 case R.id.retirer: if (getListAdapter().getCount() > 0)
 {
 patient = (Patient) ListAdapter().getItem(0);
 adapter.remove(patient);
 }
 break;
 }
 adapter.notifyDataSetChanged();
 }
}
```

```
@Override
protected void onResume()
{
 //Patient.setCompteur(0); Souhaitable ?
 super.onResume();
}
}
```

Venons-en à présent à la persistance proprement dite ...

### 24.3 Les bases de SQLite sous Android

Supposons donc vouloir créer en local (c'est-à-dire sur le mobile) une base de données avec des données concernant les patients et voyons quel arsenal minimal nous est nécessaire.

Il faut savoir qu'il n'existe pas de base de données initiale : ce sera à l'application de créer sa base. Pour cette opération, et aussi pour ouvrir une base existante, on aura recours à une classe dérivée **MedecinSQLiteHelper** de la classe abstraite **SQLiteOpenHelper** (package android.database.sqlite) : ce sera cette classe qui non seulement créera la base si nécessaire, mais l'ouvrira à la demande et gérera les éventuelles mise à jour de la base en termes de la définition de ses tables et de ses contraintes. L'implémentation de cette classe abstraite implique la redéfinition de deux méthodes :

- ◆ public abstract void **onCreate** (SQLiteDatabase db)  
appelée lors de la création de la base (le réceptacle typique de la création des tables);
- ◆ public abstract void **onUpgrade** (SQLiteDatabase db, int oldVersion, int newVersion)  
appelée lors des modifications de la base (donc des tables, des contraintes, etc) - on remarquera le numéro des versions passé en paramètres.

Le constructeur de notre classe appellera classiquement celui de la classe mère, qui est :

```
public SQLiteOpenHelper (Context context, String name, SQLiteDatabase.CursorFactory
factory, int version)
```

le deuxième paramètre étant le nom de la base, le troisième paramètre pouvant être mis à null pour utiliser un curseur par défaut (voir ci-dessous).

Notre classe sera finalement :

#### **MedecinSQLiteHelper.java**

```
package my.mesapplications.sqlite;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class MedecinSQLiteHelper extends SQLiteOpenHelper
{
 public static final String TABLE_PATIENTS = "patientsavoir";
```

```
// si on veut faire dans le générique
public static final String COLONNE_ID = "_id"; // toujours "_id" par convention !
public static final String COLONNE_NOM = "nom";
public static final String COLONNE_ADRESSE = "adresse";

private static final String NOM_BDD = "patients.db";
private static final int VERSION_BDD = 1;

public MedecinSQLiteHelper(Context context, String name,
 SQLiteDatabase.CursorFactory factory, int version)
{
 super(context, name, factory, version);
}

public MedecinSQLiteHelper(Context context)
{
 super(context, NOM_BDD, null, VERSION_BDD);
}

@Override
public void onCreate(SQLiteDatabase db)
{
 db.execSQL("create table " + TABLE_PATIENTS +
 "(_id integer primary key autoincrement, "
 + "nom text not null, "
 + "adresse text);");
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
{
 Log.w(MedecinSQLiteHelper.class.getName(),
 "Mise à jour de la version " + oldVersion + " à "
 + newVersion + " - les anciennes données seront perdues");
 db.execSQL("drop table if exists " + TABLE_PATIENTS);
 onCreate(db);
}
```

On constate que les commandes SQL proprement dites sont prises en charge par un objet **SQLiteDatabase** qu'il nous faut donc apprendre à connaître ...

#### 24.4 Une couche DAO : SQLiteDatabase

La classe **SQLiteDatabase** (package android.database.sqlite) que 'on voit apparaître systématiquement comme paramètre dans les méthodes `onCreate()` et `onUpdate()` du helper est en fait un **DAO** (Data Access Object) typique : c'est la classe qui manipule les commandes SQL. On y trouve par exemple les méthodes

- ◆ public void **beginTransaction** ()  
public void **beginTransactionNonExclusive** ()

et

public void **endTransaction** ()

- ◆ public void **execSQL** (String sql)  
pour les commandes qui ne renvoient pas de data
- ◆ public Cursor **query** (String table, String[] columns, String selection,  
String[] selectionArgs, String groupBy, String having, String orderBy, String limit)

et

public Cursor **rawQuery** (String sql, String[] selectionArgs)

pour les commandes qui renvoient des données : celles-ci sont accessibles au moyen de l'objet implémentant l'interface **Cursor** (package android.database), lequel comporte

- \* des méthodes de déplacement comme  
public abstract boolean **move** (int offset)  
public abstract boolean **moveToFirst** ()  
public abstract boolean **moveToLast** ()  
public abstract boolean **moveToNext** ()  
...

- \* des méthodes d'accès aux champs comme  
public abstract String **getString** (int columnIndex)  
...

Bien sûr, un **SQLiteDatabase** possède aussi les méthodes

public long **insert** (String table, String nullColumnHack, ContentValues values)  
public int **update** (String table, ContentValues values, String whereClause, String[] whereArgs)  
public int **delete** (String table, String whereClause, String[] whereArgs)

mais les deux premières méthodes utilisent des **ContentValues** (pacakge android.content). En fait, un tel objet agit comme une hashtable, se composant de clés et de valeurs associées. Ici, les clés sont les noms des colonnes et les valeurs sont la valeur à placer dans cette colonne.  
Bien sûr, on retrouve les méthodes classiques des containers associtifs :

---

```
public void put (String key, Integer value)
public Object get (String key)
public String getAsString (String key)
...
```

En pratique, on n'instancie pas un tel objet SQLiteDatabase : **on l'obtient de l'objet qui gère la base**, donc de l'implémentation de **SQLiteOpenHelper** au moyen des méthodes :

```
public SQLiteDatabase getReadableDatabase ()
public SQLiteDatabase getWritableDatabase ()
```

- les connaisseurs auront reconnu les curseurs read-only et read-wite.

A remarquer que dans ce dernier cas de read-write, la méthode onCreate() est appelée, ainsi que la méthode

```
public void onOpen (SQLiteDatabase db)
```

De plus, les tables sont mises en cache, si bien que l'utilisation de la méthode endTransaction() et même

```
public synchronized void close()
```

ne doit pas être oubliée.

### **Remarque**

La différence entre les deux méthodes rawQuery() et query() est que rawQuery() accepte directement une requête écrite en SQL tandis que la méthode query() requiert une série de paramètres constitutifs de cette requête. A comparer :

```
Cursor cursor = getReadableDatabase().rawQuery("select * from todo where _id = ?",
 new String[] { id })
);

Cursor cursor = getReadableDatabase().query(DATABASE_TABLE,
 new String[] { KEY_ROWID, KEY_CATEGORY, KEY_SUMMARY,
 KEY_DESCRIPTION }, null, null, null, null, null);
```

### **24.5 Une classe DAO**

Plutôt que d'encombrer notre activité avec l'utilisation explicite d'un objet SQLiteDatabase fourni par un objet SQLiteOpenHelper, nous allons charger une classe de gérer l'accès aux données - ainsi qu'on l'enseigne dans toutes les bonnes écoles ;-). La classe PatientsDAO va donc utiliser ces deux classes pour fournir à l'activité des méthodes comme :

- ◆ public Patient **creerPatient**(String n, String a) : elle ajoute un tuple patient à la table patientsavoir et retourne l'objet Patient correspondant (pour quoi faire ? pour l'adapter qui contrôle la vue !)

- ◆ public void **retirerPatient**(Patient patient) : elle supprime de la table le patient passé en paramètre
- ◆ public List<Patient> **getListePatients()** : elle fournit une liste d'objet Patient; elle parcourt bien sûr un curseur et utilise pour transformer le tuple lu en un objet Patient la méthode (private) :
- ◆ private Patient **curseurToPatient**(Cursor c).

Notre classe DAO peut à présent s'écrire comme suit :

### **PatientsDAO.java**

```
package my.mesapplications.sqlite;

import java.util.ArrayList;
import java.util.List;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;

public class PatientsDAO
{
 private SQLiteDatabase database;
 private MedecinSQLiteHelper dbHelper;
 private String[] listeColonnes = { MedecinSQLiteHelper.COLONNE_ID,
 MedecinSQLiteHelper.COLONNE_NOM,
 MedecinSQLiteHelper.COLONNE_ADRESSE };

 public PatientsDAO(Context context)
 {
 dbHelper = new MedecinSQLiteHelper(context);
 }

 public void open() throws SQLException
 {
 database = dbHelper.getWritableDatabase();
 }

 public void close() { dbHelper.close(); }

 public Patient creerPatient(String n, String a)
 {
 ContentValues values = new ContentValues();
 values.put(MedecinSQLiteHelper.COLONNE_NOM, n);
 values.put(MedecinSQLiteHelper.COLONNE_ADRESSE, a);
 }
}
```

```
long insertId = database.insert(MedecinSQLiteHelper.TABLE_PATIENTS, null,
 values);
Cursor c = database.rawQuery("select * from " +
 MedecinSQLiteHelper.TABLE_PATIENTS +
 " where " + MedecinSQLiteHelper.COLONNE_ID + " = " +
 Long.toString(insertId), null);
c.moveToFirst();
Patient nouveauPatient = curseurToPatient(c);
c.close();
return nouveauPatient;
}

public void retirerPatient(Patient patient)
{
 long id = patient.getId();
 database.delete(MedecinSQLiteHelper.TABLE_PATIENTS,
 MedecinSQLiteHelper.COLONNE_ID + " = " + id, null);
}

public List<Patient> getListePatients()
{
 List<Patient> listePatients = new ArrayList<Patient>();

 Cursor c = database.query(MedecinSQLiteHelper.TABLE_PATIENTS,
 listeColonnes, null, null, null, null, null);

 c.moveToFirst();
 while (!c.isAfterLast())
 {
 Patient patient = curseurToPatient(c);
 listePatients.add(patient);
 c.moveToNext();
 }
 // make sure to close the curseur
 c.close();
 return listePatients;
}

private Patient curseurToPatient(Cursor c)
{
 Patient patient = new Patient();
 patient.setId(c.getLong(0));
 patient.setNom(c.getString(1));
 patient.setAdresse(c.getString(2));
 return patient;
}
```

## 24.6 Retour à ListActivity

Notre activité va donc à présent s'adapter au fait que la liste de patients qu'elle doit gérer lui sera fourni par une source de données matérialisée par un objet PatientsDAO. Elle passera par celui-ci pour prévenir l'adapter de l'apparition ou de la disparition d'un patient dans la liste :

### PatientsAVoirActivity.java (2)

```
package my.mesapplications.sqlite;

//import android.app.Activity;
//import android.app.ListActivity;
//import android.os.Bundle;
//import android.view.View;
//import android.widget.ArrayAdapter;
//import android.widget.EditText;
//import java.util.ArrayList;
//import java.util.List;
//import java.util.Random;

public class PatientsAVoirActivity extends ListActivity
{
 private PatientsDAO datasource;

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

 //1. Version de base sans BDD
 /*List<Patient> modele = new ArrayList<Patient>();
 ArrayAdapter<Patient> adapter =
 new ArrayAdapter<Patient>(this,android.R.layout.simple_list_item_1, modele);
 setListAdapter(adapter);
 */

 //2. Version avec BDD
 datasource = new PatientsDAO(this);
 datasource.open();

 List<Patient> values = datasource.getlistePatients();
 ArrayAdapter<Patient> adapter =
 new ArrayAdapter<Patient>(this,android.R.layout.simple_list_item_1, values);
 setListAdapter(adapter);

 }
}
```

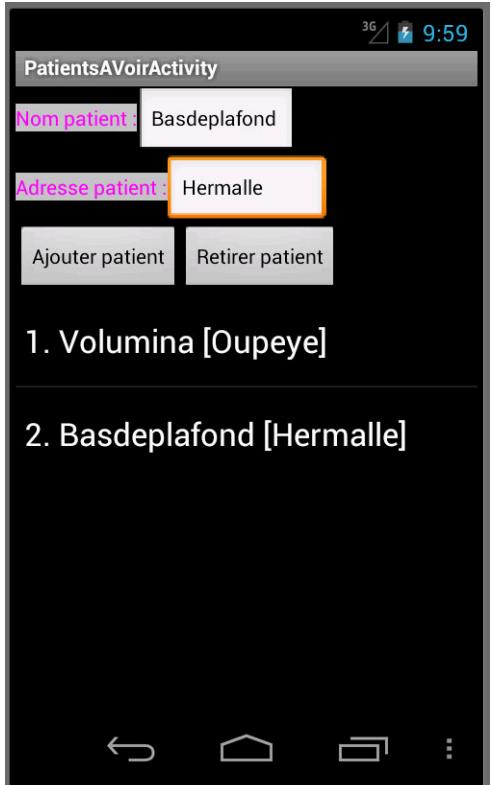
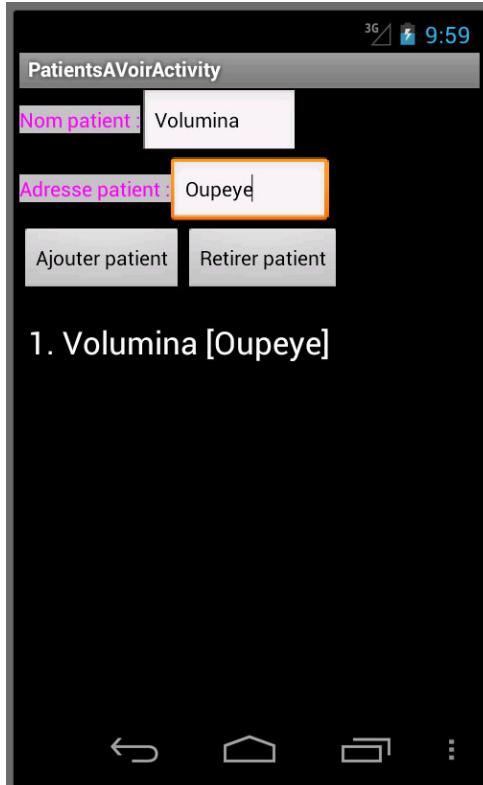
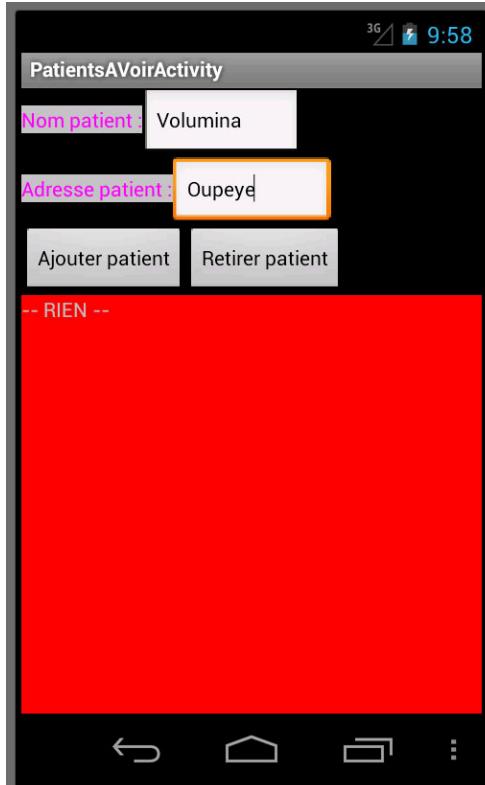
```
public void onClick(View view)
{
 ArrayAdapter<Patient> adapter = (ArrayAdapter<Patient>) getListAdapter();
 EditText nom = (EditText) findViewById(R.id.nom);
 EditText adresse = (EditText) findViewById(R.id.adresse);

 switch (view.getId())
 {
 case R.id.ajouter:
 //1. Sans BDD
 /*Patient patient = new Patient();
 patient.setNom(nom.getText().toString(),adresse.getText().toString());
 */
 //2. Avec BDD
 Patient patient =
 datasource.creerPatient(nom.getText().toString(),adresse.getText().toString());
 adapter.add(patient);
 break;
 case R.id.retirer:
 if (getListAdapter().getCount() > 0)
 {
 patient = (Patient) getListAdapter().getItem(0);
 datasource.retirerPatient(patient);
 adapter.remove(patient);
 }
 break;
 }
 adapter.notifyDataSetChanged();
}

@Override
protected void onResume()
{
 //Patient.setCompteur(0);
 datasource.open();
 super.onResume();
}

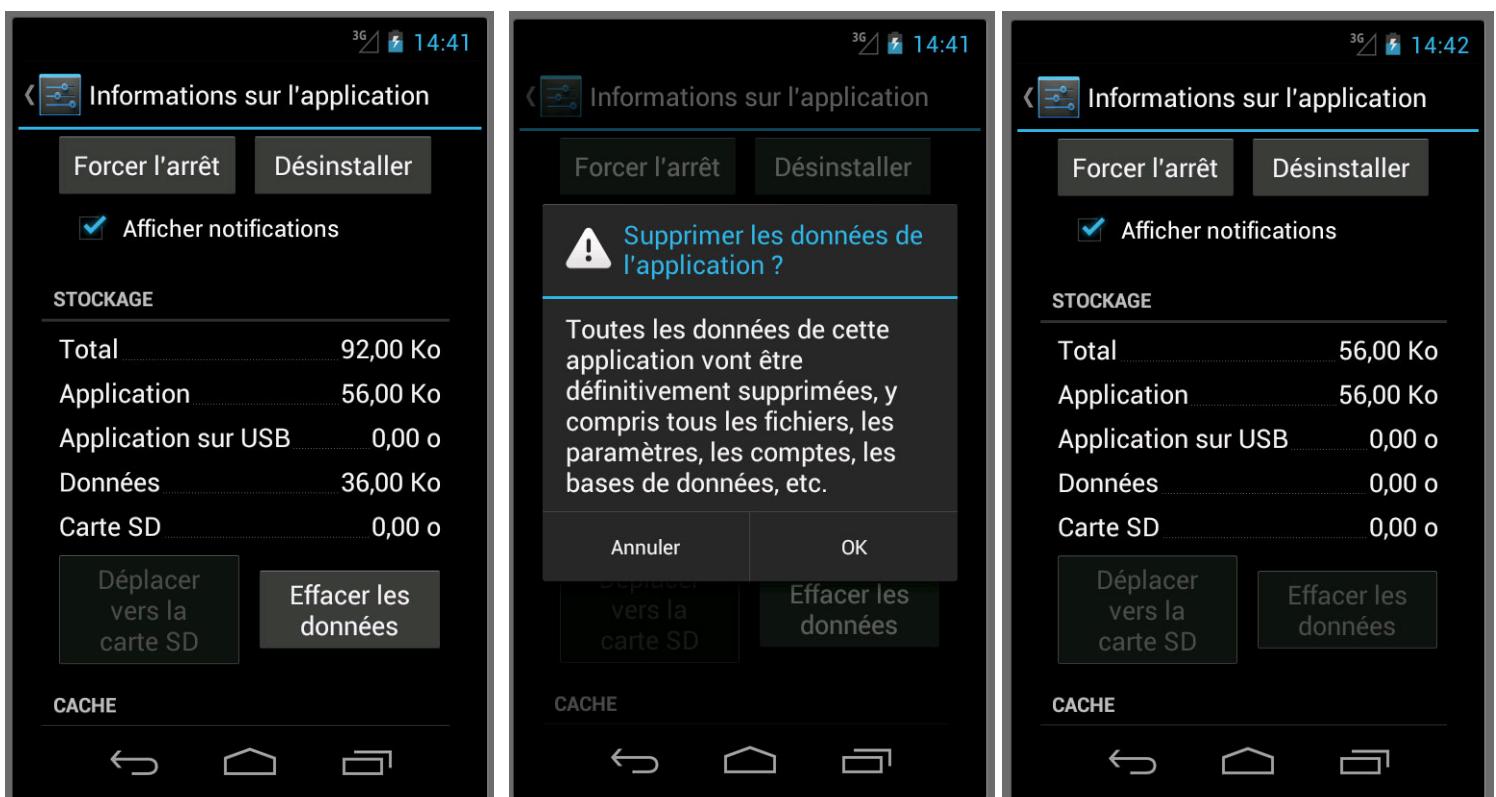
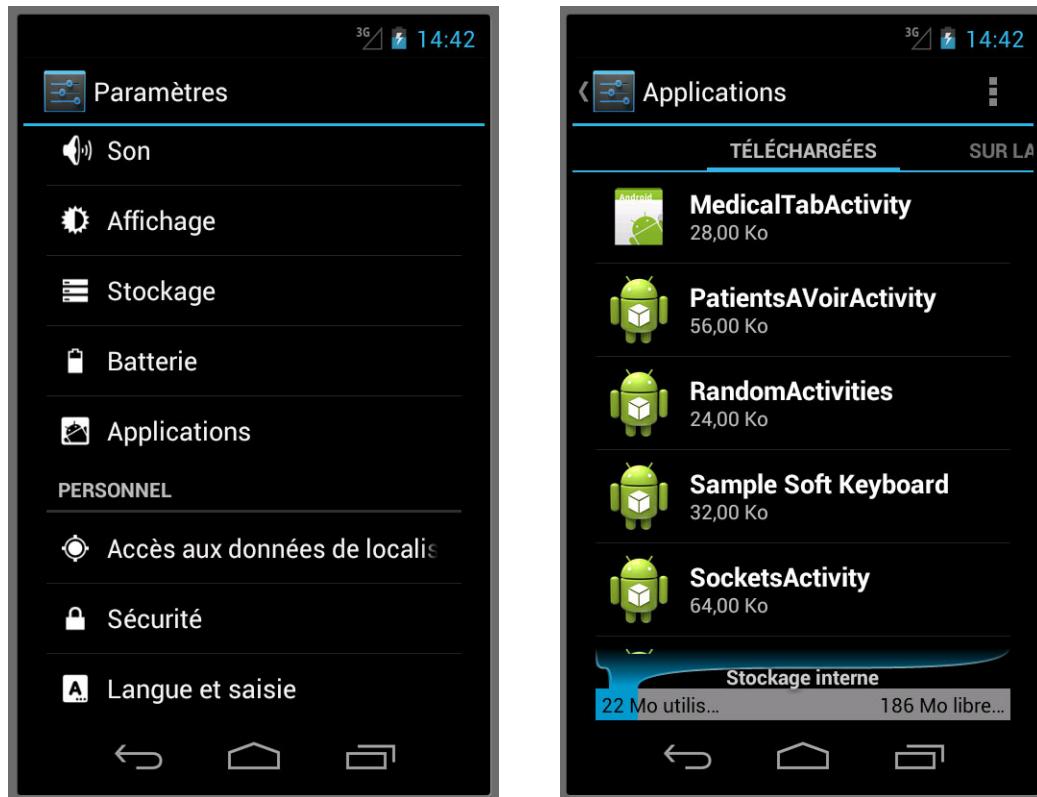
@Override
protected void onPause()
{
 datasource.close();
 super.onPause();
}
```

Un exemple d'exécution serait :



## 24.7 Effacer les données

Lors de l'évolution de la base et de la mise au point de l'application, il peut être utile d'effacer carrément la base existante de façon manuelle. Il suffit pour cela d'aller effacer les données de l'application comme indiqué ci-dessous :



## 25. Les graphiques statistiques

### 25.1 Un élément indispensable du data mining

Suite logique du traitement des données au moyen des SGBDs locaux ou serveurs, le traitement statistiques des données, plus particulièrement le data mining, est vite ressenti comme une nécessité. Un aspect très courant est celui de la visualisation des données au moyen de graphiques statistiques.

Plusieurs libraires sont proposées pour réaliser de tels graphiques. Citons simplement :

- ◆ AFreeChart : on s'en doute, il s'agit de l'adaptation du célèbre JFreeChart pour J2SE; cette librairie utilise cependant **une View particulière**, la DemoView;
- ◆ AChartEngine : librairie Open Source créée par Dan Dromereschi, qui **s'intègre très naturellement dans les Views courantes d'Android**.

Cette librairie **AChartEngine** (<http://www.achartengine.org/> - les javadocs y sont disponibles) existe à l'heure actuelle en sa version 1.1. Open Source, elle s'utilise sous licence Apache 2.0. Elle permet de réaliser tous les graphiques statistiques courants : histogrammes, graphiques linéaires, nuages de points, courbes d'interpolations, diagramme sectoriel, combinaisons de plusieurs types de graphes, etc (voir <https://code.google.com/p/achartengine/> pour quelques vues suggestives). Elle est distribuée en un seul jar **achartengine-1.1.0.jar** qu'il suffit d'intégrer au projet.

### 25.2 Fonctionnement de base de la librairie AChartEngine

L'utilisation est assez similaire à celle d'autres librairies de graphiques statistiques :

a) un *dataset*, instance de la classe **XYMultipleSeriesDataset** du package org.achartengine.model représente les données : il contient en fait une ou plusieurs *séries de données*, structurées selon le type de graphique envisagé (package org.achartengine.model):

- ◆ série pour nuage des points ou graphiques linéaires (classe **XYSeries**);
- ◆ série pour diagramme sectoriel (classe **CategorySeries**);
- ◆ série pour les digrammes en beignets" ou "doughnut" (classe **MultipleCategorySeries**);
- ◆ série chronologiques (classe **TimeSeries**, classe dérivée de XYSeries)

b) un *renderer* est responsable de tout l'aspect visuel : couleurs, polices, légendes, etc; on dispose bien sûr de renderers (org.achartengine.renderer) dédiés aux différents types de graphiques.

b1) Plus précisément **DefaultRenderer** est la classe représentant le renderer "global" du graphique comportant de multiples séries; il utilisera un renderer spécifique par série, instance de la classe **SimpleSeriesRenderer** dans le cas des graphiques linéaires et histogrammes, qu'il mémorisera par :

```
public void addSeriesRenderer(SimpleSeriesRenderer renderer)
...
```

DefaultRenderer possède toute une série de méthodes de configuration du graphique, avec même :

public void **setZoomButtonsVisible**(boolean visible)

qui permet de faire apparaître des boutons de zoom parfaitement fonctionnels ☺ !

La classe **SimpleSeriesRenderer** possède une classe dérivée **XYMultipleSeriesRenderer** qui est la classe dédiée aux séries de type "nuage de points". Pour ces deux classes, notons quelques méthodes :

◆ **SimpleSeriesRenderer** :

```
public void setColor(int color)
public void setShowLegendItem(boolean showLegend)
public void setDisplayChartValues(boolean display)
public void setStroke(BasicStroke stroke)
...
```

avec la classe **Color** contenant les constantes habituelles et avec la classe **BasicStroke** contenant les constantes indiquant le style de ligne.

◆ **XYSeriesRenderer** :

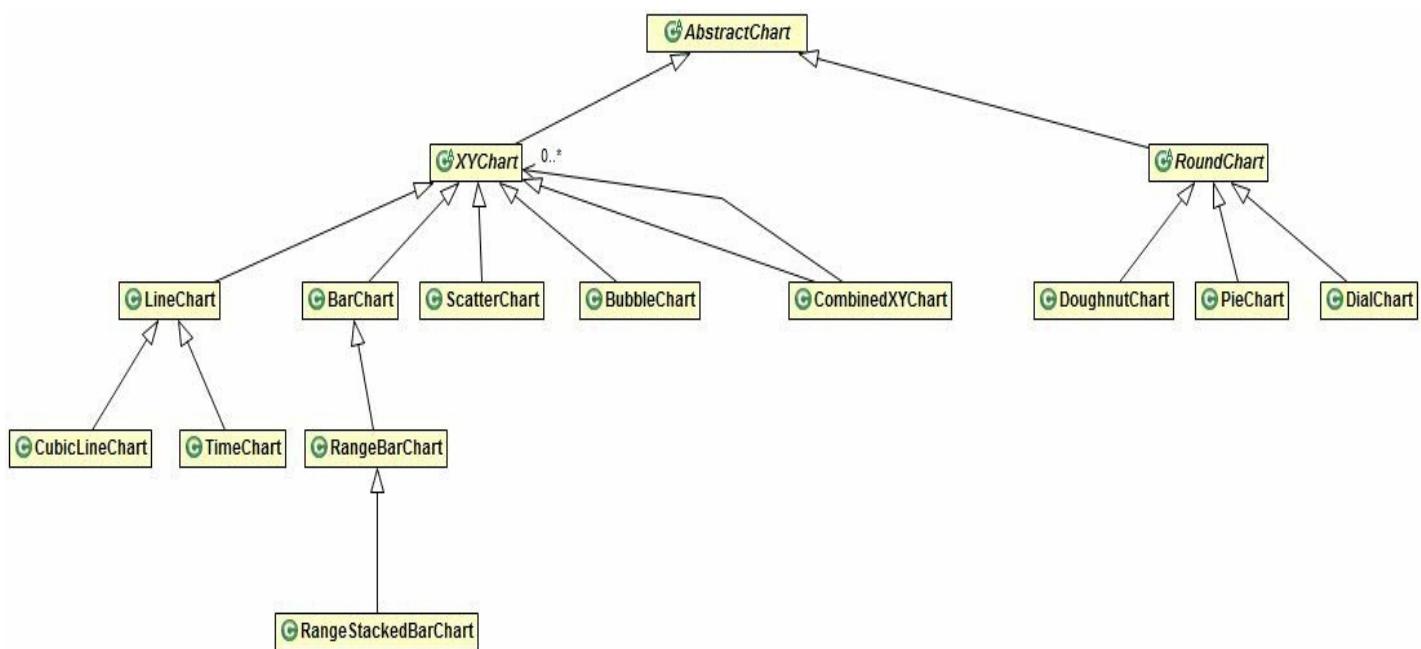
```
public void setFillPoints(boolean fill)
public void setLineWidth(float lineWidth)
public void setPointStyle(PointStyle style)
```

avec la classe **PointStyle** contenant les constantes indiquant les types de points possibles ( cercle, losange, etc)..

b2) La classe **DialRenderer**, héritée de **DefaultRenderer**, est dédiée aux graphiques de forme circulaires avec des méthodes spécifiques :

```
public void setAngleMax(double max)
public void setAngleMin(double max)
public void setMaxValue(double min)
public void setMinValue(double min)
...
```

c) un chart est chargé de la logique d'association entre les données et leur rendu. Il existe toute une hiérarchie de classes chart dédiées à chaque type de graphique. La classe abstraite de base **AbstractChart** possède deux classe filles, **RoundChart** et **XYChart**, bien sûr dédiées aux graphiques circulaires et aux graphiques linéaires. Elles mêmes se spécialisent en classe dédiées (toutes ces classes se trouvent dans le package org.achartengine.chart) :



[© <http://jaxenter.com>]

d) un objet **GraphicalView** encapsule l'objet chart approprié au type de graphique souhaité : il s'agit donc d'une View d'Android dédiée à gérer les graphiques de la librairie. Elle possède une méthode

`public void repaint()`

pour être réaffichée correctement.

C'est le rôle de la classe factory **ChartFactory** (package org.achartengine) de générer l'objet View (et donc le chart qu'il contient) au moyen de la méthode de classe appropriée au graphique souhaité. Citons :

- ◆ `public static final GraphicalView getLineChartView(android.content.Context context, XYMultipleSeriesDataset dataset, XYMultipleSeriesRenderer renderer)`
- ◆ `public static final GraphicalView getBarChartView(android.content.Context context, XYMultipleSeriesDataset dataset, XYMultipleSeriesRenderer renderer, BarChart.Type type)`
- ◆ `public static final GraphicalView getDoughnutChartView(android.content.Context context, MultipleCategorySeries dataset, DefaultRenderer renderer)`
- ◆ `public static final GraphicalView getPieChartView(android.content.Context context, CategorySeries dataset, DefaultRenderer renderer)`
- ◆ `public static final GraphicalView getScatterChartView(android.content.Context context, XYMultipleSeriesDataset dataset, XYMultipleSeriesRenderer renderer)`
- ◆ ...

En réalité, toutes ces méthodes possèdent une "sœur" qui retourne l'objet chart encapsulé dans un Intent (qui sera alors passé comme paramètre au démarrage de l'activité qui en fait usage) au moyen de méthodes comme (par exemple) :

`public static final android.content.Intent getLineChartIntent(android.content.Context context, XYMultipleSeriesDataset dataset, XYMultipleSeriesRenderer renderer).`

### 25.3 Un exemple de graphique circulaire

Supposons vouloir faire apparaître un diagramme circulaire (diagramme "en tarte" ou "en camembert" représentant les sièges obtenus par les partis au conseil communal d'une petite localité de Boursoulavie. L'activité s'écrire en reprenant les points cités plus haut :



#### **StatActivity.java**

```
package my.mesapplications.stat;

import org.achartengine.ChartFactory;
import org.achartengine.GraphicalView;
import org.achartengine.model.CategorySeries;
import org.achartengine.model.SeriesSelection;
import org.achartengine.renderer.DefaultRenderer;
import org.achartengine.renderer.SimpleSeriesRenderer;

import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup.LayoutParams;
import android.widget.LinearLayout;
import android.widget.Toast;
//import org.achartengine.chart.BarChart;

public class StatActivity extends Activity
{
 private static final int[] couleurs = new int[] { Color.GREEN,
 Color.BLUE, Color.RED, Color.rgb(125, 125, 0) };
 private static double[] resultats = new double[] { 8, 23, 31, 17 };
 private static final String[] nomPartis = new String[] { "Ecolos naturistes", "Grand
 ami de la NV-A", "Gauche pour la démocratie contrôlée", "Parti Papiste et
 Oecuménique" };

 private CategorySeries serieStat = new CategorySeries("Elections à
 DataMiningGulch");
 private DefaultRenderer rendererGlobal = new DefaultRenderer();
 private GraphicalView vue;

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

 rendererGlobal.setApplyBackgroundColor(true);
 rendererGlobal.setBackgroundColor(Color.argb(100, 50, 50, 50));
 rendererGlobal.setChartTitleTextSize(20);
 rendererGlobal.setLabelsTextSize(15);
```

```

rendererGlobal.setLegendTextSize(15);
rendererGlobal.setMargins(new int[] { 20, 30, 15, 0 });
rendererGlobal.setZoomButtonsVisible(true);
rendererGlobal.setStartAngle(90);

for (int i = 0; i < resultats.length; i++)
{
 serieStat.add(nomPartis[i] + " " + resultats[i], resultats[i]);
 SimpleSeriesRenderer renderer = new SimpleSeriesRenderer();
 renderer.setColor(couleurs[(serieStat.getItemCount() - 1) %
 couleurs.length]);
 rendererGlobal.addSeriesRenderer(renderer);
}

if (vue != null)
{
 vue.repaint();
}
}

@Override
protected void onResume()
{
 super.onResume();
 if (vue == null)
 {
 LinearLayout layout = (LinearLayout) findViewById(R.id.chart);
 vue = ChartFactory.getPieChartView(this, serieStat,
 rendererGlobal);
 //vue = ChartFactory.getBarChartView(this, serieStat, rendererGlobal,
 // BarChart.Type.DEFAULT);
 rendererGlobal.setOnClickListener(true);
 rendererGlobal.setSelectableBuffer(10);
 // rayon de la zone clickable

 vue.setOnClickListener(new View.OnClickListener()
 {
 @Override
 public void onClick(View v)
 {
 SeriesSelection seriesSelection =
 vue.getCurrentSeriesAndPoint();
 if (seriesSelection == null)
 {
 Toast.makeText(StatActivity.this,"No ...",
 Toast.LENGTH_SHORT).show();
 }
 else
 {
 Toast.makeText(StatActivity.this,"Chart ..."+
```

```
 (seriesSelection.getPointIndex() + 1) + " was clicked" +
 " point value=" + seriesSelection.getValue(),
 Toast.LENGTH_SHORT).show();
 }
}
});

layout.addView(vue, new LayoutParams(LayoutParams.FILL_PARENT,
 LayoutParams.FILL_PARENT));
}
else
{
 vue.repaint();
}
}

}
```

On remarquera l'ajout de la vue au layout de l'activité : par objet et pas par l'habituel `findViewById()`. Le layout est donc au départ vide :

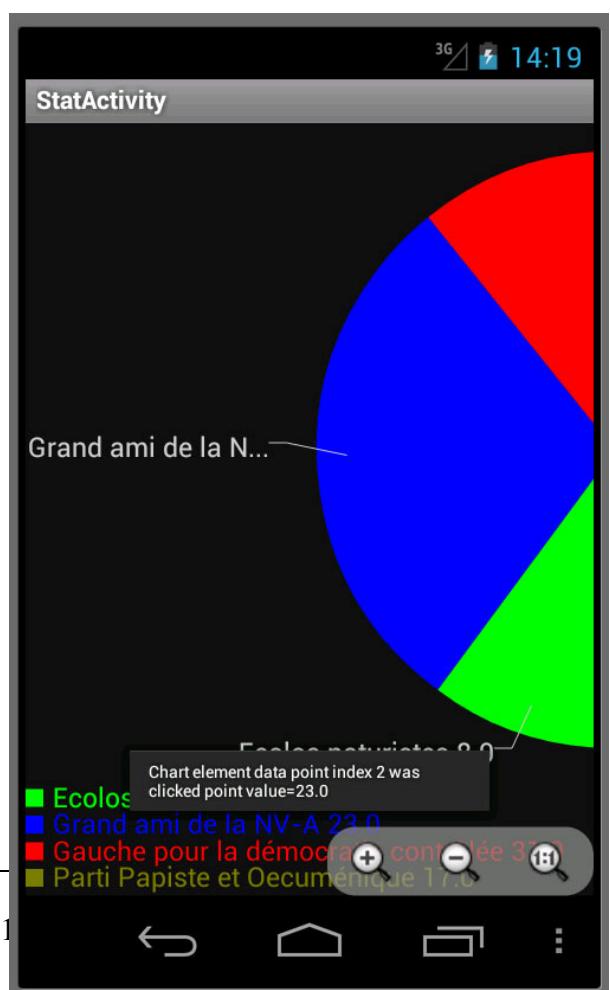
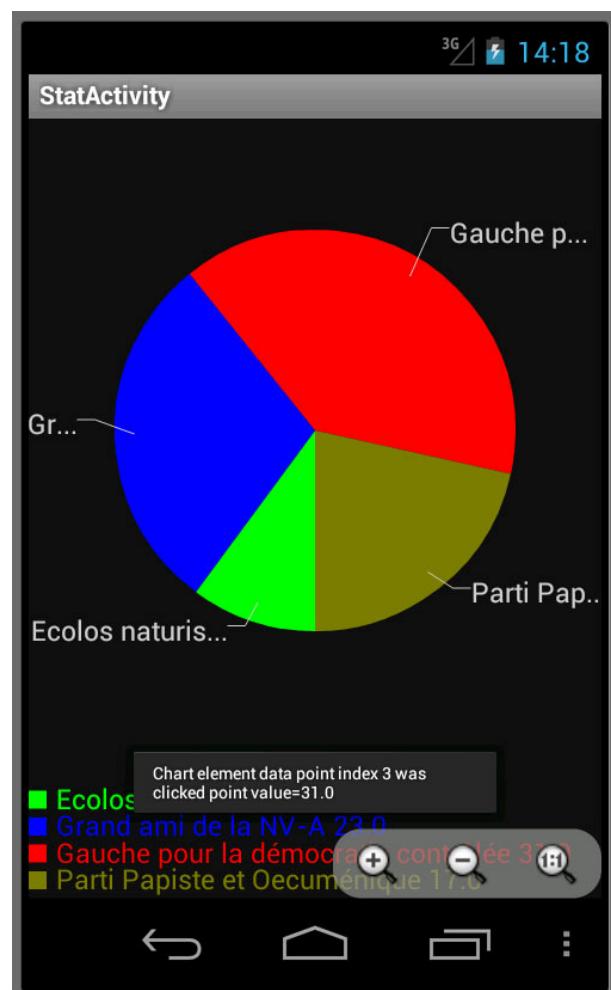
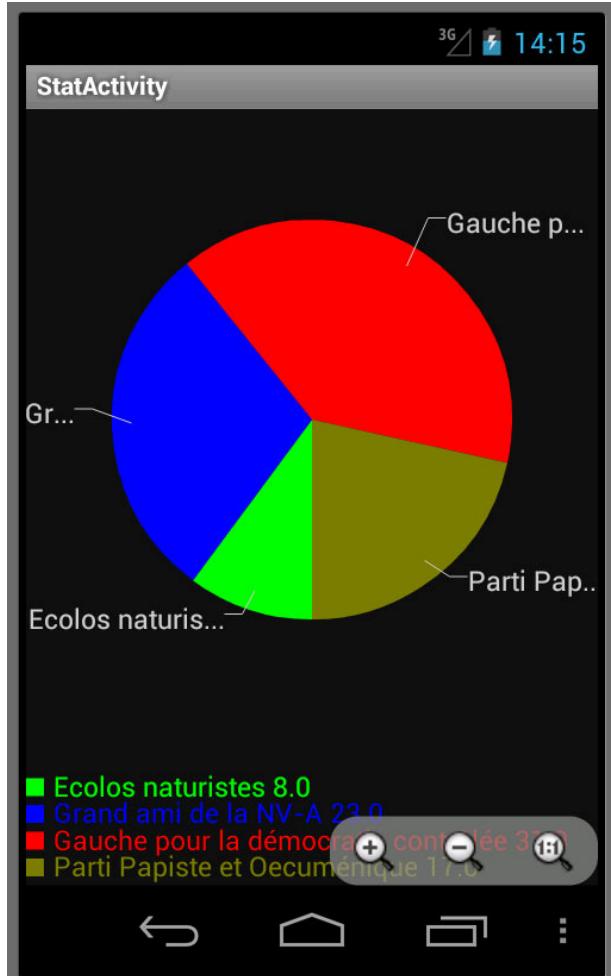
### main.xml

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

 <LinearLayout
 android:id="@+id/chart"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:layout_weight="1"
 android:orientation="horizontal" >
 </LinearLayout>

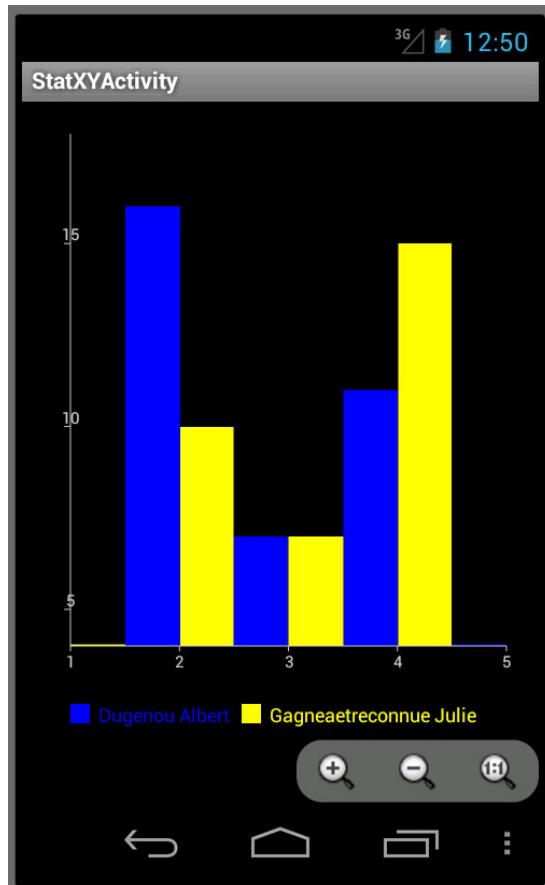
</LinearLayout>
```

Le résultat ressemble à ceci :



#### 25.4 Un exemple de graphique linéaire

Considérons ici l'affichage juxtaposé de deux séries de nombres (disons que ce sont des cotes de deux étudiants) selon le principe de l'histogramme comparé. L'objectif est d'obtenir ceci :



L'application s'écrira simplement (de manière très basique) :

#### StatXYActivity.java

```
package my.mesapplications.stat.basics;

import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.widget.LinearLayout;
import org.achartengine.ChartFactory;
import org.achartengine.GraphicalView;
import org.achartengine.chart.BarChart;
import org.achartengine.chart.PointStyle;
import org.achartengine.model.XYMultipleSeriesDataset;
import org.achartengine.model.XYSeries;
import org.achartengine.renderer.BasicStroke;
import org.achartengine.renderer.XYMultipleSeriesRenderer;
import org.achartengine.renderer.XYSeriesRenderer;
```

```
public class StatXYActivity extends Activity
{
 private GraphicalView vue;
 private XYMultipleSeriesDataset ds = new XYMultipleSeriesDataset();
 private XYMultipleSeriesRenderer rendererMain = new XYMultipleSeriesRenderer();
 private XYSeries serieEtudiant1;
 private XYSeries serieEtudiant2;
 private XYSeriesRenderer renderer1;
 private XYSeriesRenderer renderer2;

 private void initChart()
 {
 serieEtudiant1 = new XYSeries("Dugenou Albert");
 ds.addSeries(serieEtudiant1);
 serieEtudiant2 = new XYSeries("Gagneaetreconnue Julie");
 ds.addSeries(serieEtudiant2);

 renderer1 = new XYSeriesRenderer();
 renderer1.setColor(Color.BLUE);
 renderer1.setLineWidth(3);
 renderer1.setPointStyle(PointStyle.DIAMOND);
 renderer1.setStroke(BasicStroke.SOLID);

 renderer2 = new XYSeriesRenderer();
 renderer2.setColor(Color.YELLOW);
 renderer2.setLineWidth(2);
 renderer2.setPointStyle(PointStyle.CIRCLE);
 renderer2.setStroke(BasicStroke.DASHED);

 rendererMain.addSeriesRenderer(renderer1);
 rendererMain.setZoomButtonsVisible(true);
 rendererMain.addSeriesRenderer(renderer2);
 }

 private void addData()
 {
 serieEtudiant1.add(1, 12); serieEtudiant1.add(2, 16); serieEtudiant1.add(3, 7);
 serieEtudiant1.add(4, 11); serieEtudiant1.add(5, 4);

 serieEtudiant2.add(1, 4); serieEtudiant2.add(2, 10); serieEtudiant2.add(3, 7);
 serieEtudiant2.add(4, 15); serieEtudiant2.add(5, 18);
 }

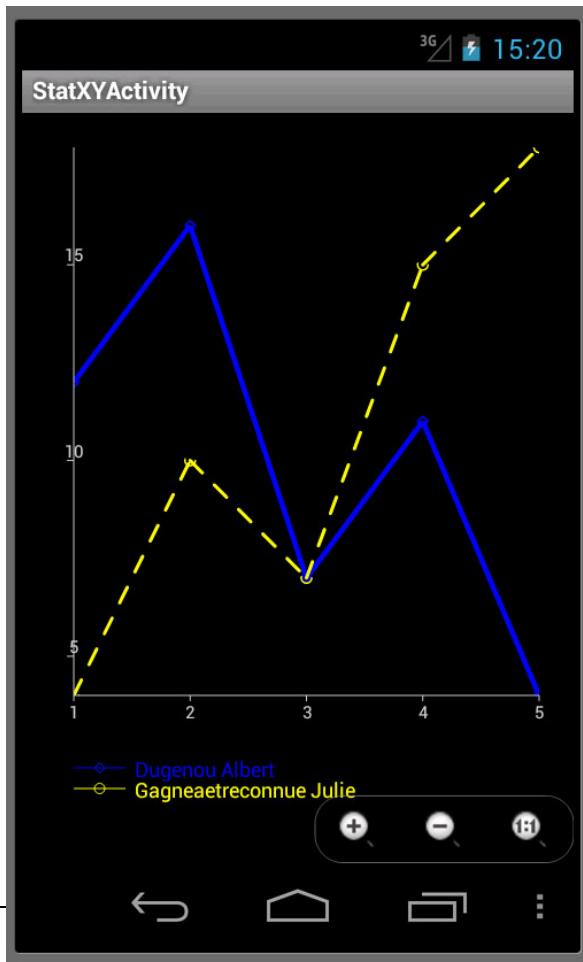
 @Override
 protected void onCreate(Bundle savedInstanceState)
 {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);
 }
}
```

```
@Override
protected void onResume()
{
 super.onResume();
 LinearLayout layout = (LinearLayout) findViewById(R.id.chart);
 if (vue == null)
 {
 initChart();
 addData();
 vue = ChartFactory.getLineChartView(this, ds, rendererMain);
 layout.addView(vue);
 }
 else
 {
 vue.repaint();
 }
}
```

Bien sûr, un histogramme comparé est ici parfaitement adapté. Mais si l'on voulait un graphique en "ligne brisée" (par exemple parce que les différents cours sont les niveaux successifs d'une même matière), il suffirait de remplacer la ligne de construction de la vue par :

```
vue = ChartFactory.getLineChartView(this, ds, rendererMain);
```

ce qui donnerait :



## 26. L'internationalisation

Le Java standard internationalise ses applications avec le mécanisme des bundles : il s'agit de fichiers properties contenant tous les même entrées avec une traduction en la langue qui dicte le suffixe du nom du fichier en question. Android pratique de manière similaire, mais bien sûr avec des fichiers xml. En fait, reprenant notre applications "élections" du point précédent, il suffit de créer dans le sous-répertoire **res** un sous-répertoire **values-fr** (par exemple) dans lequel on crée un fichier strings.xml comme celui-ci :

### res/values-fr/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <string name="app_name">StatXYActivity</string>
 <string name="partiE">Ecolo</string>
 <string name="partiL">MR</string>
 <string name="partiS">PS</string>
 <string name="partiC">CDH</string>
</resources>
```

Dans notre application, il suffira de faire référence à ces entrées au moyen de la méthode que possède toute Activity :

```
public Resources getResources()
```

qui donne bien sûr accès aux ressources de l'apk. Ici, il nous faudra utiliser la méthode de Resources :

```
public Configuration getConfiguration()
```

qui permet d'accéder à la préférence de langue du smartphone hôte au moyen de la variable membre publique :

```
public Locale locale
```

cette dernière possédant des méthodes comme :

```
String getCountry()
static Locale getDefault()
final String getDisplayCountry()
final String getDisplayLanguage()
final String getDisplayName()
...
```

Mais surtout, Resources possède la méthode :

```
public String getString (int id)
```

qui va bien sûr nous permettre de récupérer la valeur de nos entrées. Au final, notre application adaptée aura comme nouveautés :

### StatActivity.java (international)

```
package my.mesapplications.stat;

...
public class StatActivity extends Activity
{
 private static final int[] couleurs = new int[] { Color.GREEN,
 Color.BLUE,Color.RED, Color.rgb(125, 125, 0) };
 private static double[] resultats = new double[] { 8, 23, 31, 17 };
 private static final String[] nomPartis = new String[4];

 private CategorySeries serieStat = new CategorySeries("Elections à
 DataMiningGulch");
 private DefaultRenderer rendererGlobal = new DefaultRenderer();
 private GraphicalView vue;

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

 rendererGlobal.setApplyBackgroundColor(true);
 ...

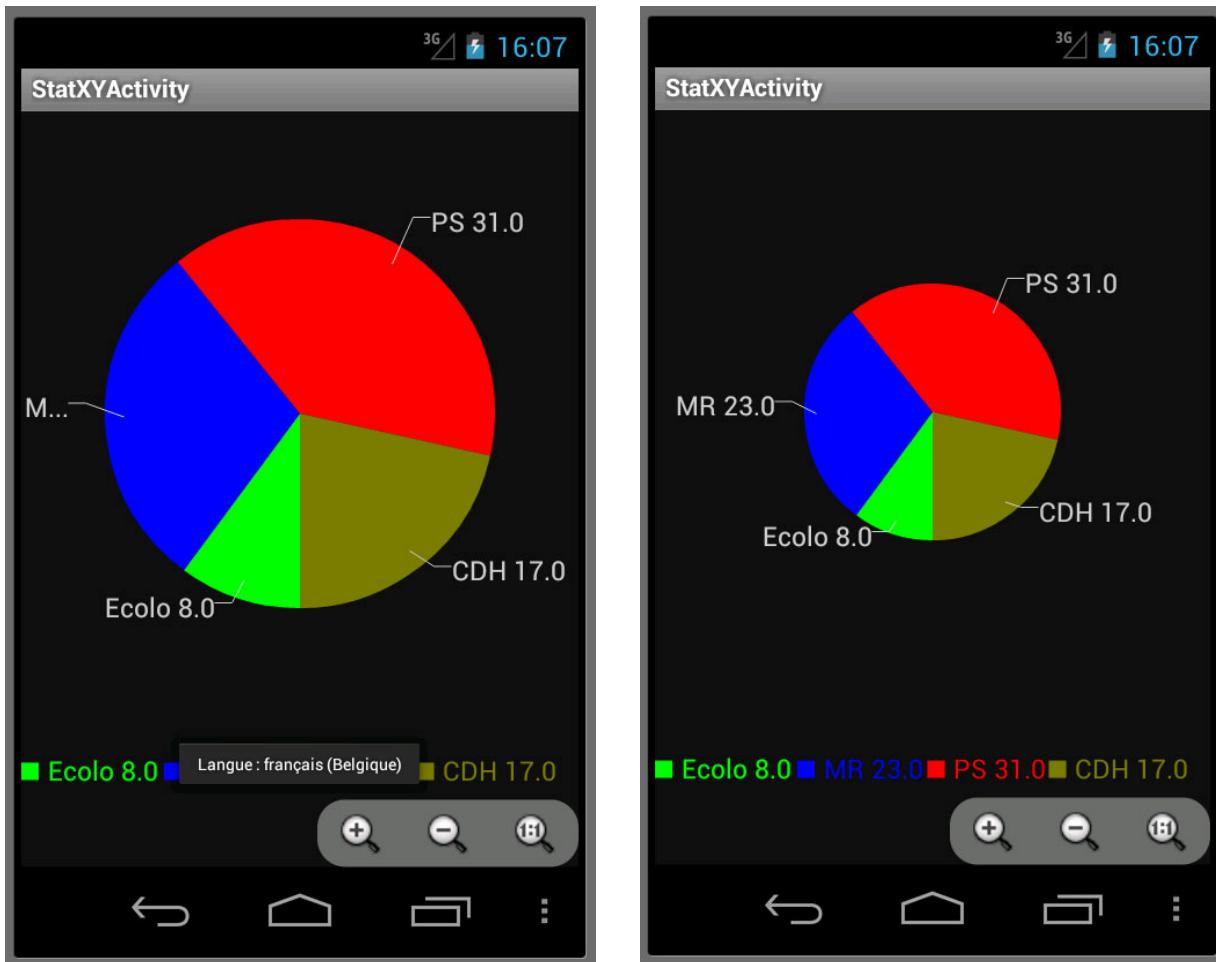
 nomPartis[0]= getResources().getString(R.string.partiE);
 nomPartis[1]= getResources().getString(R.string.partiL);
 nomPartis[2]= getResources().getString(R.string.partiS);
 nomPartis[3]= getResources().getString(R.string.partiC);

 String locale = "Langue : "+this.getResources().getConfiguration().
 locale.getDisplayName();
 Toast.makeText(this,locale,Toast.LENGTH_LONG).show();

 for (int i = 0; i < resultats.length; i++) { ... }

 ...
 }
 ...
}
```

Résultat pour notre AVD configuré en français :



Un développeur attentif aura cependant remarqué dans la console de construction de l'apk ceci :

```
aapt: warning: string 'partiC' has no default translation in C:\java-netbeans-
application\AndroidAFreeChart\res; found: fr
aapt: warning: string 'partiE' has no default translation in C:\java-netbeans-
application\AndroidAFreeChart\res; found: fr
aapt: warning: string 'partiL' has no default translation in C:\java-netbeans-
application\AndroidAFreeChart\res; found: fr
aapt: warning: string 'partiS' has no default translation in C:\java-netbeans-
application\AndroidAFreeChart\res; found: fr
```

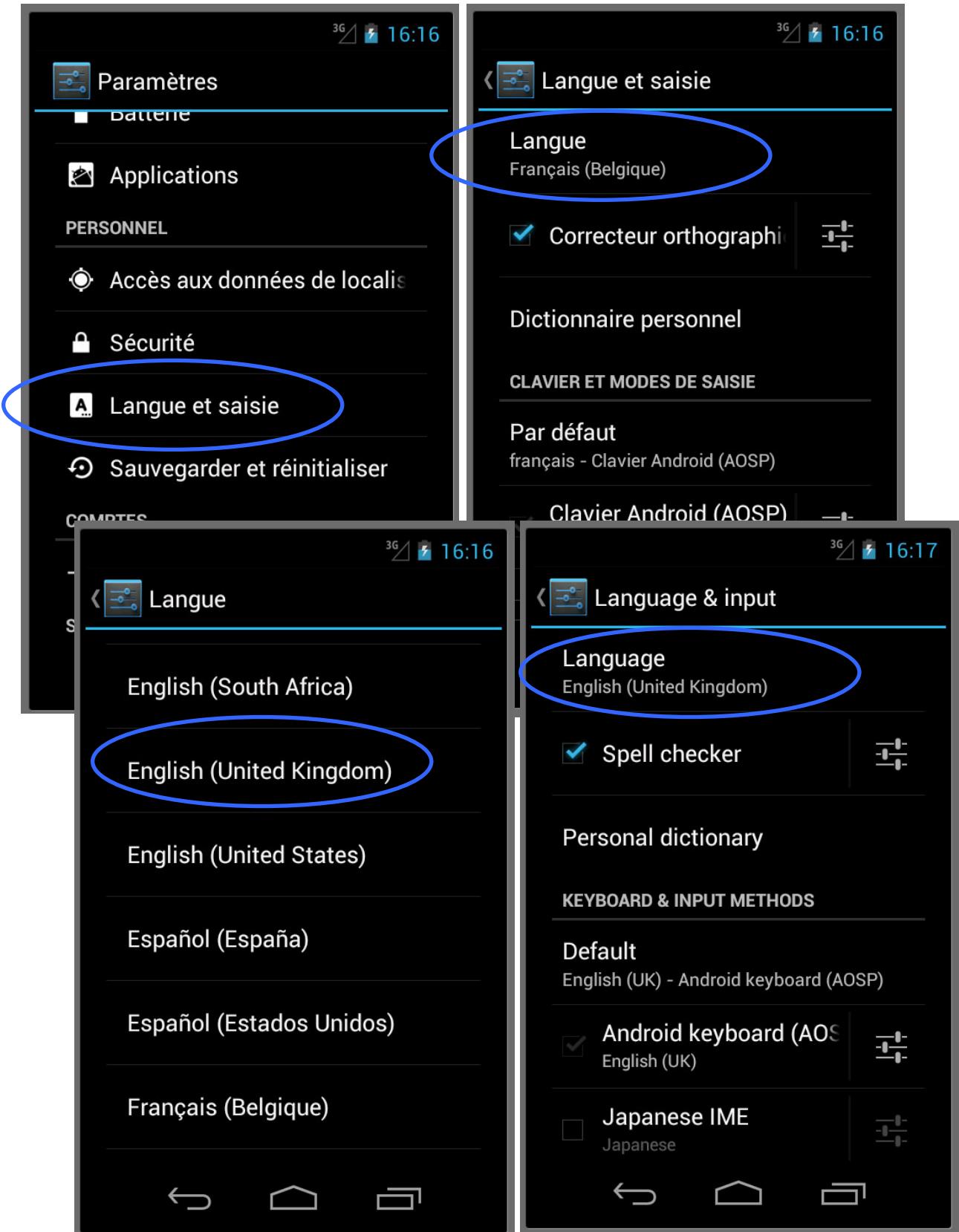
En fait, le fichier strings.xml du sous-répertoire values ne contient pas les entrées "partieE", etc. Ajoutons -les donc :

#### **res/values/strings.xml**

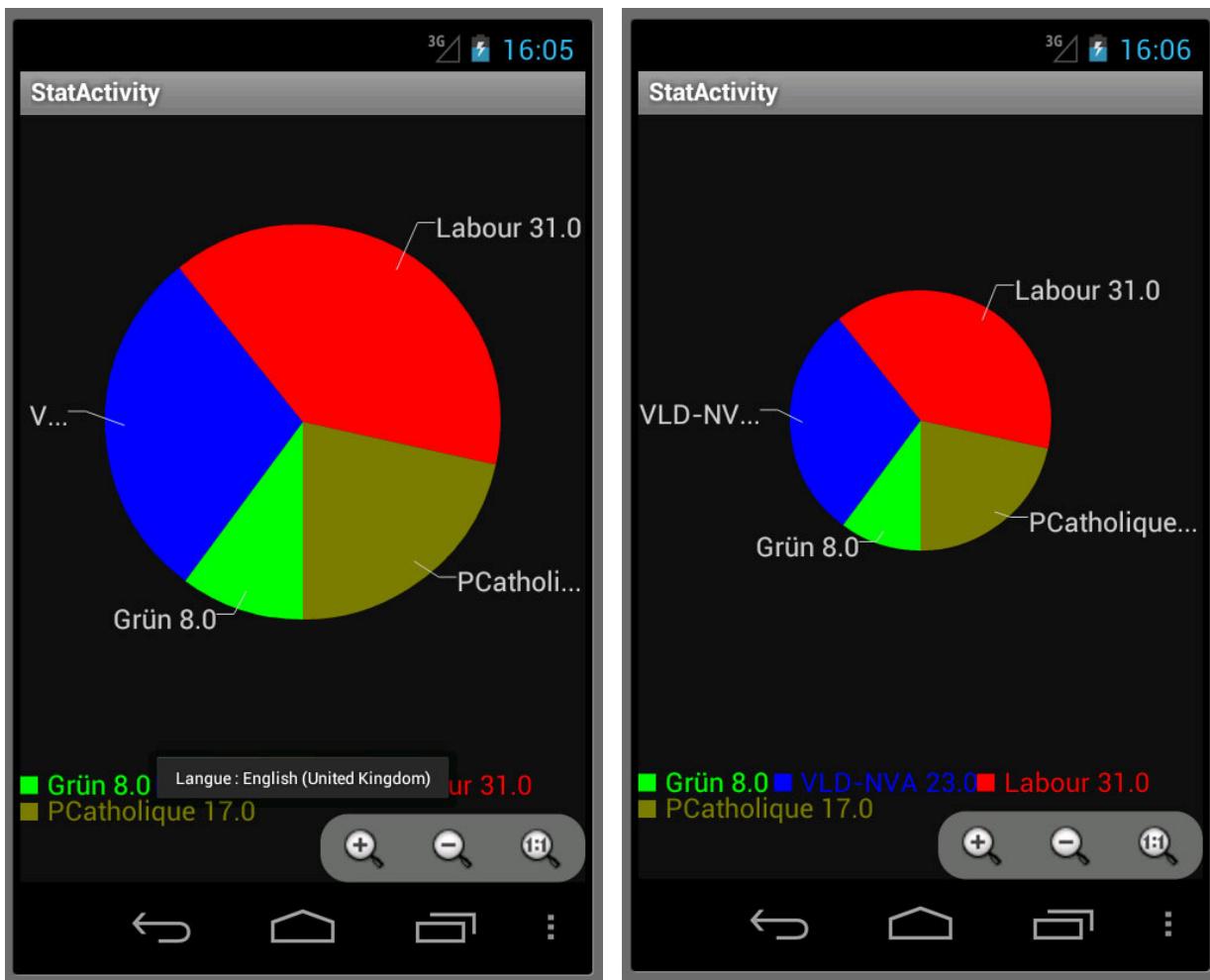
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <string name="app_name">StatActivity</string>
 <string name="partiE">Grün</string>
 <string name="partiL">VLD-NVA</string>
 <string name="partiS">Labour</string>
```

```
<string name="partiC">PCatholique</string>
</resources>
```

Si notre téléphone est configuré en anglais :



on aura alors :

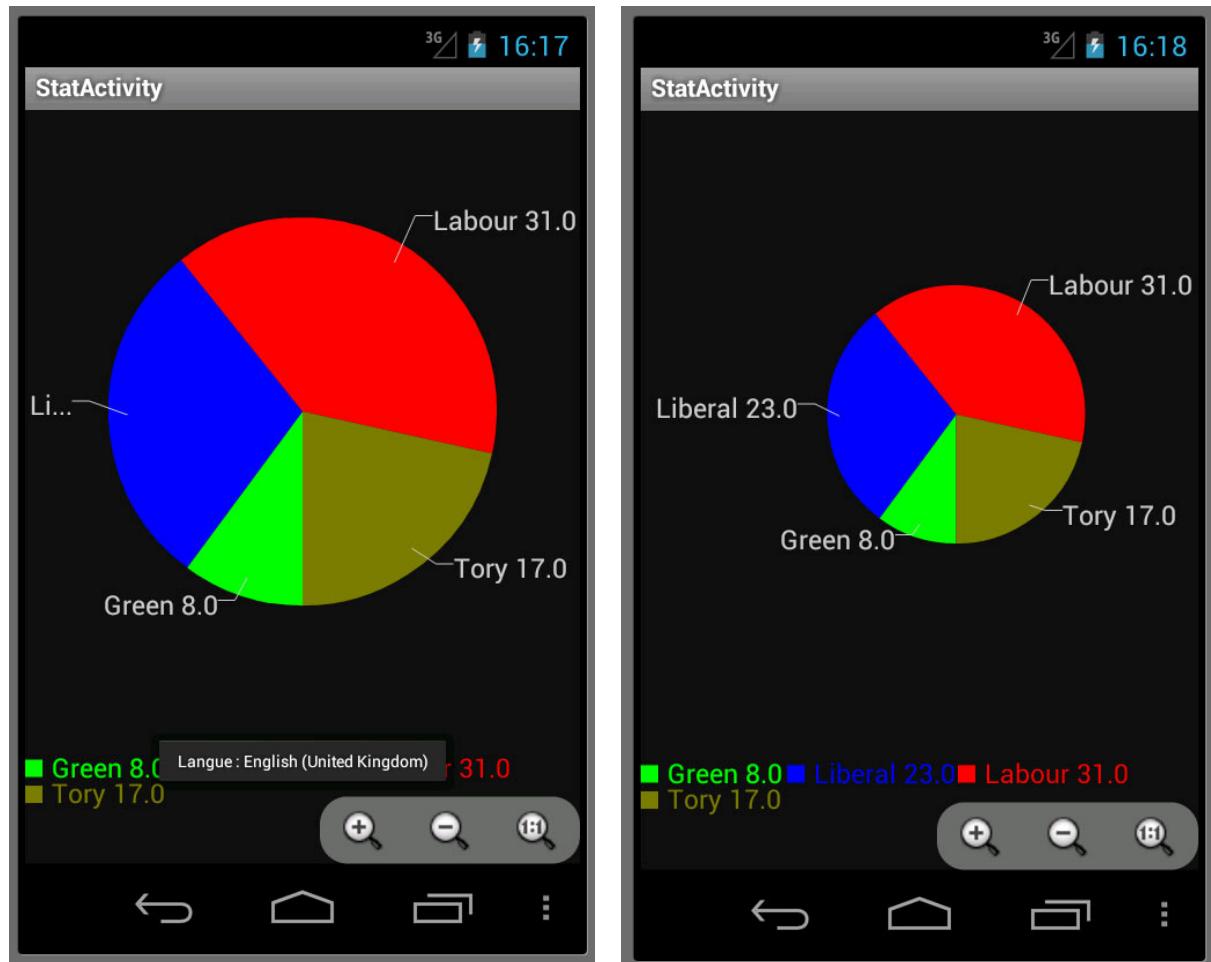


Mais sans doute est-il plus utile d'ajouter la version anglaise :

### values-en/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <string name="app_name">StatActivity</string>
 <string name="partiE">Green</string>
 <string name="partiL">Liberal</string>
 <string name="partiS">Labour</string>
 <string name="partiC">Tory</string>
</resources>
```

si bien que :



En fait, il est possible de désigner encore plus finement la langue utilisée en utilisant une indication supplémentaire sur le pays. Ainsi, par exemple, on peut avoir pour le français les sous-répertoires **values-fr\_rBE**, **values-fr\_rCA**, **values-fr\_rFR** et **values-fr\_rCH**.



## Ouvrages consultés

### Ouvrages imprimés et électroniques

**Benbourahla, N.** Android 5 - Les fondamentaux du développement d'applications Java. Editions ENI. Saint-Herblain, France. 2015.

**Boulanger, M.** Etude de la plate-forme mobile Android et développement d'un prototype d'application cartographique dédiée aux activités en extérieur. TFE Haute Ecole de la Province de Liège (In.Pr.E.S.). 2010.

**Cosson, A.** Kotlin. Les fondamentaux du développement d'applications Android. Paris, France, Ed. Eyrolles, coll. Epsilon. 2018.

**Freres, J.** Etude des technologies Android et utilisation dans le développement de diverses applications mobiles. Seraing, Belgique. 2014.

**Hébutterne, S.** Développez une application Android - Programmation en Java sous Android Studio. Editions ENI. Saint-Herblain, France. 2017.

**Hornick, C., Lambrette, F. & Lejaer, G.** Etude comparative du développement sur les mobiles Android, Windows et Apple dans la perspective de l'application prototype d'un projet d'aide par les TIC aux personnes à autonomie réduite. Seraing, Belgique. 2014.

**Janssen, X.** Développement d'une application Android de suivi des dépenses énergétiques des bâtiments communaux avec intégration des technologies QR Codes et Google Maps. Seraing, Belgique. 2014.

**Mamay, P. & Germain, T.** Gestion informatisée de la traçabilité des denrées alimentaires utilisées par un restaurant de collectivité. TFE Haute Ecole de la Province de Liège (In.Pr.E.S.). 2011.

**Murphy, M.L.** Android Programming Tutorials. CommonsWare LLC, U.S.A. 2011.

**Murphy, M.L.** The busy coder's guide to Android. CommonsWare LLC, U.S.A. 2011.

**Murphy, M.L.** The busy coder's guide to advanced Android development. CommonsWare LLC, U.S.A. 2011.

**Ponthieu, S. & Campello Fuentes, M.** Conception et développement d'un site Web à vocation culturelle avec étude et intégration des technologies mobiles et des cartes à puces. TFE Haute Ecole Rennequin Sualem (In.Pr.E.S.). 2006.

**Verkenne, M.** Etude de la technologie Google Android et utilisation dans une application de gestion de notes de cours. TFE Haute Ecole de la Province de Liège (In.Pr.E.S.). 2010.

**Vilvens, C.** Langage Java (I) : Programmation de base. Seraing, Belgique. 2021.

**Vilvens, C.** Langage Java (II) : Programmation avancée des applications classiques. Seraing, Belgique. 2021.

**Vilvens, C.** Langage Java (IV) : Programmation de protocoles applicatifs et de techniques de sécurité. Seraing, Belgique. 2021.

\*\*\*\*\*

### Sites Internet

<http://java.sun.com/>

➔ <http://www.oracle.com/technetwork/java/index.html>

avec en particulier :

[https://community.oracle.com/community/java/java\\_desktop](https://community.oracle.com/community/java/java_desktop)

<https://jcp.org/en/home/index>



<https://www.meetup.com/fr-FR/Belgian-Java-User-Group/>

Site du Belgian Java User Group



<http://developer.android.com/reference/packages.html>

Références des classes Android

<http://www.openhandsetalliance.com>

Site de l'Open Handset Alliance

<http://developer.android.com/index.html>

Site des développeurs Android

<http://www.ibm.com/developerworksopensource/library/os-android-devel>

Site IBM pour Android

