Développement sous Android

Jean-Francois Lalande - May 2015 - Version 2.2

Le but de ce cours est de découvrir la programmation sous Android, sa plate-forme de développement et les spécificités du développement embarqué sur *smartphone*.

Les grandes notions abordées dans ce cours sont:

- Bâtir l'interface d'une application
- Naviguer et faire communiquer des applications
- Manipuler des données (préférences, fichiers, ...)
- Services, threads et programmation concurrente
- Les capteurs, le réseau

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE

1 Plan du module

Plan du module

1.1 Ressources

Les ressources de ce cours sont disponibles en ligne à l'adresse:

http://www.univ-orleans.fr/lifo/Members/Jean-Francois.Lalande/teaching.html.

On y trouve deux versions du même contenu:

- Slides du cours
- Support de cours

2 Le SDK Android

2.1 Introduction

Il est important de prendre la mesure des choses. A l'heure actuelle (May 2015):

- juillet 2011: **550 000** activations **par jour**
- décembre 2011: **700 000** activations **par jour**
- sept. 2012: **1.3 millions** d'activations **par jour** (Wikipedia)
- avril 2013: **1.5 millions** d'activations **par jour** (Wikipedia)

Il y aurait donc un parc de **400 millions** d'appareils Android.

Vous pouvez visionner de la propagande ici et là.

Historique des versions

Le nombre de *release* est impressionnant [Version]:

Nom	Version	Date
Android	1.0	09/2008
Petit Four	1.1	02/2009
Cupcake	1.5	04/2009
Donut	1.6	09/2009
Gingerbread	2.3	12/2010

Honeycomb	3.0	02/2011
Ice Cream Sandwich	4.0.1	10/2011
Jelly Bean	4.1	07/2012
KitKat	4.4	10/2013
Lollipop	5.0	10/2014

2.2 Android

L'ecosystème d'Android s'appuie sur deux piliers:

- le langage Java
- le SDK qui permet d'avoir un environnement de développement facilitant la tâche du développeur

Le kit de développement donne accès à des exemples, de la documentation mais surtout à l'API de programmation du système et à un émulateur pour tester ses applications.

Stratégiquement, Google utilise la licence Apache pour Android ce qui permet la redistribution du code sous forme libre ou non et d'en faire un usage commercial.

Le plugin *Android Development Tool* permet d'intégrer les fonctionnalités du SDK à Eclipse. Il faut l'installer comme un plugin classique en précisant l'URL du plugin. Ensuite, il faut renseigner l'emplacement du SDK (préalablement téléchargé et décompressé) dans les préférences du plugin ADT.

L'Operating System

Android est en fait un système de la famille des Linux, pour une fois sans les outils GNU. L'OS s'appuie sur:

- un noyau Linux (et ses drivers)
- un couche d'abstraction pour l'accès aux capteurs (HAL)
- une machine virtuelle: *Dalvik Virtual Machine*
- des applications (navigateur, gestion des contacts, application de téléphonie...)
- des bibliothèques (SSL, SQLite, OpenGL ES, etc...)

[Dalvik] est le nom de la machine virtuelle open-source utilisée sur les systèmes Android. Cette machine virtuelle exécute des fichiers .dex, plus ramassés que les .class classiques. Ce format évite par exemple la duplication des **String** constantes. La machine virtuelle utilise elle-même moins d'espace mémoire et l'adressage des constantes se fait par un pointeur de 32 bits.

[Dalvik] n'est pas compatible avec une JVM du type Java SE ou même Java ME. La librairie d'accès est donc redéfinie entièrement par Google.

Le plugin de développement d'Eclipse: ADT

Un projet basé sur le plugin ADT est décomposé de la manière suivante:

- **src/**: les sources Java du projet
- **libs/**: bibliothèques tierces
- **res/**:
  - **res/drawable**: ressources images

    - **res/layout**: description des IHMs en XML
    - **res/values**: chaines de caractères et dimensions
- **gen/**: les ressources auto générées par ADT
- **assets/**: ressources brutes (*raw bytes*)
- **bin/**:
    - **bin/classes**: les classes compilées en *.class*
    - **bin/classes.dex**: exécutable pour la JVM Dalvik
    - **bin/myapp.zip**: les ressources de l'application
    - **bin/myapp.apk**: application empaquetée avec ses ressource et prête pour le déploiement

Les éléments d'une application

Une application Android peut être composée des éléments suivants:

- des activités (**android.app.Activity**): il s'agit d'une partie de l'application présentant une vue à l'utilisateur
- des services (**android.app.Service**): il s'agit d'une activité tâche de fond sans vue associée
- des fournisseurs de contenus (**android.content.ContentProvider**): permet le partage d'informations au sein ou entre applications
- des widgets (**android.appwidget.***): une vue accrochée au Bureau d'Android
- des *Intents* (**android.content.Intent**): permet d'envoyer un message pour un composant externe sans le nommer explicitement
- des récepteurs d'*Intents* (**android.content.BroadcastReceiver**): permet de déclarer être capable de répondre à des *Intents*
- des notifications (**android.app.Notifications**): permet de notifier l'utilisateur de la survenue d'événements

Le Manifest de l'application

Le fichier **AndroidManifest.xml** déclare l'ensemble des éléments de l'application.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="andro.jf"
      android:versionCode="1"
      android:versionName="1.0">
  <application android:icon="@drawable/icon"
               android:label="@string/app_name">

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

```
    <service>...</service>
    <receiver>...</receiver>
    <provider>...</provider>

    </application>
</manifest>
```

2.3 Les ressources

Les ressources de l'applications sont utilisées dans le code au travers de la classe statique **R**. ADT re-génère automatiquement la classe statique **R** à chaque changement dans le projet. Toutes les ressources sont accessibles au travers de **R**, dés qu'elles sont déclarées dans le fichier XML ou que le fichier associé est déposé dans le répertoire adéquat. Les ressources sont utilisées de la manière suivante:

```
android.R.type_ressource.nom_ressource
```

qui est de type int. Il s'agit en fait de l'identifiant de la ressource. On peut alors utiliser cet identifiant ou récupérer l'instance de la ressource en utilisant la classe **Resources**:

```
Resources res = getResources();
String hw = res.getString(R.string.hello);
XXX o = res.getXXX(id);
```

Une méthode spécifique pour les objets graphiques 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:

```
TextView texte = (TextView)findViewById(R.id.le_texte);
texte.setText("Here we go !");
```

Les chaines

Les chaines constantes de l'application sont situées dans **res/values/strings.xml**. L'externalisation des chaines permettra de réaliser l'internationalisation de l'application. Voici un exemple:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello Hello JFL !</string>
    <string name="app_name">AndroJF</string>
</resources>
```

La récupération de la chaine se fait via le code:

```
Resources res = getResources();
String hw = res.getString(R.string.hello);
```

Internationalisation

Le système de ressources permet de gérer très facilement l'internationalisation d'une application. Il suffit de créer des répertoires **values-XX** où **XX** est le code de la langue que l'on souhaite implanter. On place alors dans ce sous répertoire le fichier xml **strings.xml** contenant les chaines traduites associées aux même clefs que dans **values/strings.xml**. On obtient par exemple pour les langues *es* et *fr* l'arborescence:

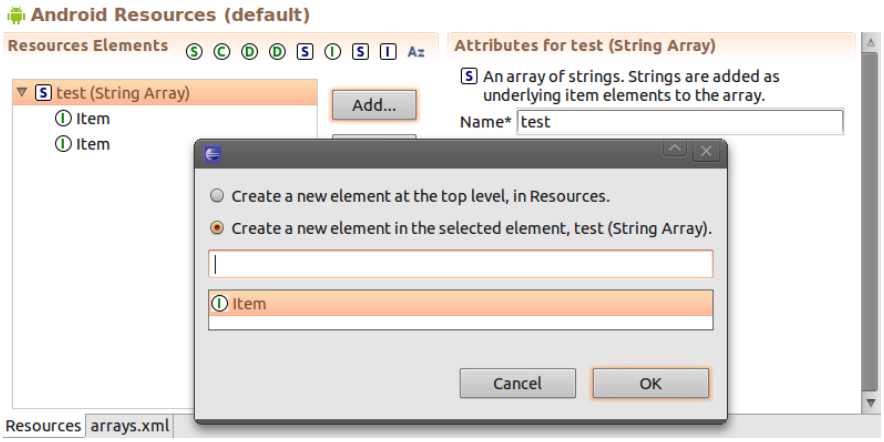
```
MyProject/
  res/
     values/
         strings.xml
     values-es/
         strings.xml
     values-fr/
         strings.xml
```

Android chargera le fichier de resources approprié en fonction de la langue du système.

Autres valeurs simples

Plusieurs fichies xml peuvent être placés dans **res/values**. Cela permet de définit des chaines, des couleurs, des tableaux. L'assistant de création permet de créer de nouveaux fichiers de ressources contenant des valeurs simples, comme par exemple un tableau de chaines:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string-array name="test">
  <item>it1</item>
  <item>it2</item>
</string-array>
</resources>
```



Autres ressources

D'autres ressources sont spécifiables dans **res**:

- les menus
- les images (**R.drawable**)
- des dimensions (**R.dimen**)
- des couleurs (**R.color**)

2.4 Les activités

Une application Android étant hebergée sur un système embarqué, le cycle de vie d'une application ressemble à celle d'une application Java ME. L'activité peut passer des états:

- démarrage -> actif: détient le focus et est démarré
- actif -> suspendue: ne détient plus le focus
- suspendue -> actif:
- suspendue -> détruit:

Le nombre de méthodes à surcharger et même plus important que ces états:

```
public class Main extends Activity {
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.acceuil); }
  protected void onDestroy() {
    super.onDestroy(); }
  protected void onPause() {
    super.onPause(); }
  protected void onResume() {
    super.onResume(); }
  protected void onStart() {
    super.onStart(); }
  protected void onStop() {
    super.onStop(); } }
```

Cycle de vie d'une activité

onCreate() / onDestroy(): permet de gérer les opérations à faire avant l'affichage de l'activité, et lorsqu'on détruit complètement l'activité de la mémoire. On met en général peu de code dans **onCreate()** afin d'afficher l'activité le plus rapidement possible.

onStart() / onStop(): ces méthodes sont appelées quand l'activité devient visible/invisible pour l'utilisateur.

onPause() / onResume(): une activité peut rester visible mais être mise en pause par le fait qu'une autre activité est en train de démarrer, par exemple B. **onPause()** ne doit pas être trop long, car B ne sera pas créé tant que **onPause()** n'a pas fini son exécution.

onRestart(): cette méthode supplémentaire est appelée quand on relance une activité qui est passée par **onStrop()**. Puis **onStart()** est aussi appelée. Cela permet de différencier le premier lancement d'un relancement.

Le cycle de vie des applications est très bien décrit sur la page qui concerne les Activity.

Sauvegarde des interfaces d'activité

L'objet **Bundle** passé en paramètre de la méthode **onCreate** permet de restaurer les valeurs des interfaces d'une activité qui a été déchargée de la mémoire. En effet, lorsque l'on appuie par exemple sur la touche *Home*, en revenant sur le bureau, Android peut-être amené à déchargé les éléments graphiques de la mémoire pour gagner des ressources. Si l'on rebascule sur l'application (appui long sur *Home*), l'application peut avoir perdu les valeurs saisies dans les zones de texte.

Pour forcer Android à décharger les valeurs, il est possible d'aller dans "Development tools > Development Settings" et de cocher "Immediately destroy activities".

Si une zone de texte n'a pas d'identifiant, Android ne pourra pas la sauver et elle ne pourra pas être restaurée à partir de l'objet **Bundle**.

Si l'application est complètement détruite (tuée), rien n'est restauré.

Le code suivant permet de visualiser le déclenchement des sauvegardes:

```
protected void onSaveInstanceState(Bundle outState) {
      super.onSaveInstanceState(outState);
      Toast.makeText(this, "Sauvegarde !", Toast.LENGTH_LONG).show();
}
```

Démonstration

3 Interfaces graphiques

3.1 Vues et gabarits

Les éléments graphiques héritent de la classe **View**. On peut regrouper des éléments graphiques dans une **ViewGroup**. Des **ViewGroup** particuliers sont prédéfinis: ce sont des gabarits (*layout*) qui proposent une prédispositions des objets graphiques:

- **LinearLayout**: dispose les éléments de gauche à droite ou du haut vers le bas
- **RelativeLayout**: les éléments enfants sont placés les uns par rapport aux autres
- **TableLayout**: disposition matricielle
- **FrameLayout**: disposition en haut à gauche en empilant les éléments
- **GridLayout**: disposition matricielle avec N colonnes et un nombre infini de lignes

Les déclarations se font principalement en XML, ce qui évite de passer par les instanciations Java.

Attributs des gabarits

Les attributs des gabarits permettent de spécifier des attributs supplémentaires. Les plus importants sont:

- **android:layout_width** et **android:layout_height**:
  - **="match_parent"**: l'élément remplit tout l'élément parent
  - **="wrap_content"**: prend la place minimum nécessaire à l'affichage
  - **="fill_parent"**: comme **match_parent** (deprecated, API<8)
- **android:orientation**: définit l'orientation d'empilement
- **android:gravity**: définit l'alignement des éléments

Voici un exemple de **LinearLayout**:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:gravity="center"
  android:id="@+id/accueilid"
  >
</LinearLayout>
```

Exemples de ViewGroup

Un **ViewGroup** contient des éléments graphiques. Pour construire un gabarit linéaire on fera:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:gravity="center"
  android:id="@+id/accueilid"
  >
    <TextView android:id="@+id/le_texte" android:text="@string/hello" />
    <TextView android:id="@+id/le_texte2" android:text="@string/hello2" />
</LinearLayout>
```

Alors que pour empiler une image et un texte:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:id="@+id/accueilid"
  >
    <ImageView
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:src="@drawable/mon_image" />
```

```
    <TextView android:id="@+id/le_texte" android:text="@string/hello" />
</FrameLayout
```

L'interface comme ressource

Une interface graphique définie en XML sera aussi générée comme une ressource dans la classe statique **R**. Le nom du fichier xml, par exemple *accueil.xml* permet de retrouver le layout dans le code java au travers de **R.layout.accueil**.

Ainsi, pour associer la première vue graphique à l'activité principale de l'application, il faut faire:

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

Le layout reste modifiable au travers du code, comme tous les autres objets graphiques. Pour cela, il est important de spécifier un id dans la définition XML du gabarit (**android:id="@+id/accueilid"**). Le "+" signifie que cet id est nouveau et doit être généré dans la classe **R**. Un id sans "+" signifie que l'on fait référence à un objet déjà existant.

En ayant généré un *id*, on peut accéder à cet élément et agir dessus au travers du code Java:

```
LinearLayout l = (LinearLayout)findViewById(R.id.accueilid);
l.setBackgroundColor(Color.BLACK);
```

Les labels de texte

En XML:

```
<TextView
  android:id="@+id/le_texte"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/hello"
  android:layout_gravity="center"
  />
```

Par la programmation:

```
public class Activity2 extends Activity {
    public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    LinearLayout gabarit = new LinearLayout(this);
    gabarit.setGravity(Gravity.CENTER); // centrer les éléments graphiques
    gabarit.setOrientation(LinearLayout.VERTICAL); // empiler vers le bas !

    TextView texte = new TextView(this);
    texte.setText("Programming creation of interface !");
    gabarit.addView(texte);
```

```
    setContentView(gabarit);
} }
```

Les zones de texte

En XML:

```
<EditText android:text=""
          android:id="@+id/EditText01"
          android:layout_width="match_parent"
          android:layout_height="wrap_content">
</EditText>
```

Par la programmation:

```
EditText edit = new EditText(this);
edit.setText("Edit me");
gabarit.addView(edit);
```

Interception d'événements:

```
edit.addTextChangedListener(new TextWatcher() {
  @Override
  public void onTextChanged(CharSequence s, int start,
                            int before, int count) {
    // do something here
  }
);
```

Les images

En XML:

```
<ImageView
  android:id="@+id/logoEnsi"
  android:src="@drawable/ensi"
  android:layout_width="100px"
  android:layout_height="wrap_content"
  android:layout_gravity="center_horizontal"
  ></ImageView>
```

Par la programmation:

```
ImageView image = new ImageView(this);
image.setImageResource(R.drawable.ensi);
gabarit.addView(image);
```

Les boutons

En XML:

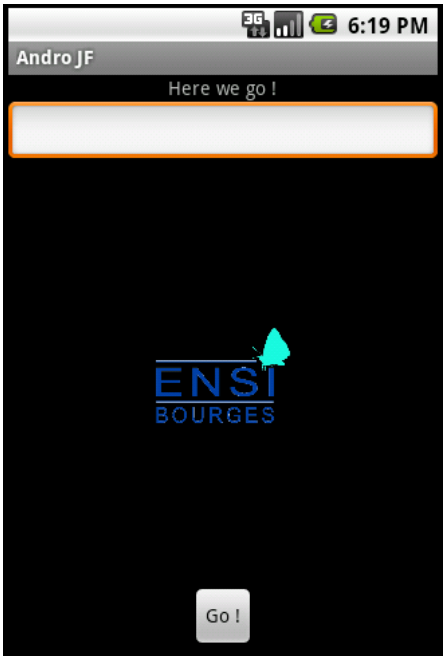
```
<Button android:text="Go !"
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
</Button>
```

La gestion des événements de *click* se font par l'intermédiaire d'un listener:

```
Button b = (Button)findViewById(R.id.Button01);
b.setOnClickListener(new OnClickListener() {

  @Override
  public void onClick(View v) {
    Toast.makeText(v.getContext(), "Stop !", Toast.LENGTH_LONG).show();
  }
  });
}
```

Interface résultat

Ce *screenshot* montre une interface contenant des **TextView**, **EditText**, **ImageView**, et un bouton (cf. ANX_Interfaces-graphiques).



Démonstration

Video

3.2 Inclusions de gabarits

Les interfaces peuvent aussi inclure d'autres interfaces, permettant de factoriser des morceaux d'interface. On utilise dans ce cas le mot clef **include**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >
<include android:id="@+id/include01"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         layout="@layout/acceuil"
         ></include>
</LinearLayout>
```

Si le gabarit correspondant à accueil.xml contient lui aussi un **LinearLayout**, on risque d'avoir deux imbrications de **LinearLayout** inutiles car redondant:

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

Merge de gabarit

Le problème précédent est dû au fait qu'un layout doit contenir un unique *root element*, du type **View** ou **ViewGroup**. On peut pas mettre une série de **TextView** sans *root element*. Pour résoudre ce problème, on peut utiliser le tag **merge**. Si l'on réécrit le layout accueil.xml comme cela:

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

L'inclusion de celui-ci dans un layout linéaire transformera:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android">
  <include android:id="@+id/include01" layout="@layout/acceuil"></include>
</LinearLayout>
```

en:

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

3.3 Positionnement avancé

Pour obtenir une interface agréable, il est souvent nécessaire de réaliser correctement le positionnement des éléments graphiques. La difficulté est d'arriver à programmer un placement qui n'est pas dépendant de l'orientation ou de la taille de l'écran.

Dans [VL], on trouve une explication pour réaliser un placement simple: un texte à gauche et une image à droite de l'écran, alignée avec le texte. Cela peut être particulièrement utile si par exemple on réalise une liste d'item qui contient à chaque fois un texte et une icône.

Le principe réside dans l'imbrication de **LinearLayout**:

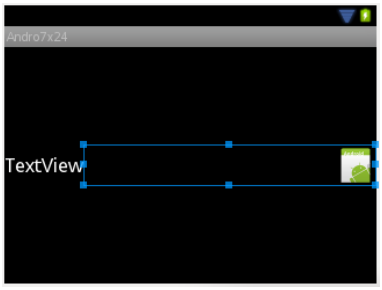
- Un premier *layout* contiendra l'ensemble des éléments. Son orientation doit être *horizontal* et sa gravité (*gravity*) *center*. On inclut ensuite le texte.
- Puis, l'image étant censée être à droite, il faut créer un **LinearLayout** consécutif au texte et préciser que la gravité (*gravity*) est *right*. Pour aligner les éléments, il faut préciser que la gravité du layout (*layout_gravity*) est *center*.

Preview du positionnement

Le *layout* décrit ci-avant ressemble à:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_height="match_parent" android:orientation="horizontal"
     android:layout_width="match_parent"
     android:gravity="center">
     <TextView ...></TextView>
     <LinearLayout android:layout_height="wrap_content"
             android:orientation="horizontal"
             android:layout_width="match_parent"
             android:gravity="right"
             android:layout_gravity="center">
     <Image .../>
</LinearLayout></LinearLayout>
```

Ce qui produit dans l'interface de preview, en orientation portrait:



3.4 Les listes

Au sein d'un gabarit, on peut implanter une liste que l'on pourra dérouler si le nombre d'éléments est important. Si l'on souhaite faire une liste plein écran, il suffit juste de poser un layout linéaire et d'y implanter une **ListView**. Le XML du gabarit est donc:

```
<LinearLayout ...>
<ListView android:id="@+id/listView1" ...>
</ListView></LinearLayout>
```

Etant donné qu'une liste peut contenir des éléments graphiques divers et variés, les éléments de la liste doivent être insérés dans un **ListAdapter** et il faut aussi définir le gabarit qui sera utilisé pour afficher chaque élément du **ListAdapter**. Prenons un exemple simple: une liste de chaine de caractères. Dans ce cas, on créé un nouveau gabarit **montexte** et on ajoute dynamiquement un **ArrayAdapter** à la liste **listView1**. Le gabarit suivant doit être placé dans *montexte.xml*:

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

Le code de l'application qui créé la liste peut être:

```
ListView list = (ListView)findViewById(R.id.listView1);
ArrayAdapter<String> tableau = new ArrayAdapter<String>(list.getContext(),
                                                        R.layout.montexte);
for (int i=0; i<40; i++) {
  tableau.add("coucou " + i); }
list.setAdapter(tableau);
```

Démonstration

Video

(cf ANX_Listes-pour-des-items-texte)

Liste de layouts plus complexes

Lorsque les listes contiennent un layout plus complexe qu'un texte, il faut utiliser un autre constructeur de **ArrayAdapter** (ci-dessous) où **resource** est l'id du layout à appliquer à chaque ligne et **textViewResourceId** est l'id de la zone de texte inclu dans ce layout complexe. A chaque entrée de la liste, la vue générée utilisera le layout complexe et la zone de texte contiendra la *string* passée en argument à la méthode **add**.

```
ArrayAdapter (Context context, int resource, int textViewResourceId)
```

Le code de l'exemple précédent doit être adapté comme ceci:

```
ListView list = (ListView)findViewById(R.id.maliste);
ArrayAdapter<String> tableau = new ArrayAdapter<String>(
                  list.getContext(), R.layout.ligne, R.id.monTexte);
for (int i=0; i<40; i++) {
  tableau.add("coucou " + i); }
list.setAdapter(tableau);
```

Avec le layout de liste suivant (ligne.xml):

```
<LinearLayout ...>
   <TextView ... android:id="@+id/monTexte"/>
```

```
   <LinearLayout> <ImageView /> </LinearLayout>
</LinearLayout>
```

Une autre solution consiste à hériter de la classe **BaseAdapter**, ce qui oblige à coder la méthode **getView()** mais évite la complexité du constructeur de **ArrayAdapter**.

Interface résultat

(cf ANX_Listes-de-layouts-complexes)

3.5 Les Fragments

A partir d'Android 3.0, on dispose d'un composant graphique similaire aux **Activity**: les **Frament**s. Un fragment est une sorte de sous activité qui possède son propre cycle de vie. On peut intégrer plusieurs fragments dans une activités, et changer un fragment par un autre dynamiquement, sans changer l'activité principale. Cela rend la programmation graphique dynamique plus aisée.

On peut intégrer un **Fragment** dans le layout de l'activité à l'aide du tag **fragment**. L'*id* permet de cibler le fragment à l'exécution, et le tag *name* permet de spécifier la classe à utiliser à l'instanciation, si on le sait à l'avance à la conception:

```
<LinearLayout ...>
  <fragment
      android:id="@+id/fragmentstatic"
      android:name="andro.jf.MonFragmentStatic" />
</LinearLayout>
```

Et la classe **MonFragmentStatique** contient le code permettant de générer la **View**:

```
public class MonFragmentStatique extends Fragment {
@Override
```

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                                 Bundle savedInstanceState) {
View v = inflater.inflate(R.layout.fragment_main, container, false);
return v;
}}
```

Fragments dynamiques

Le générateur d'exemples du SDK propose de créer un template de projet contenant un fragment dynamique ajouté à la création de l'activité (cf. ANX_Fragments):

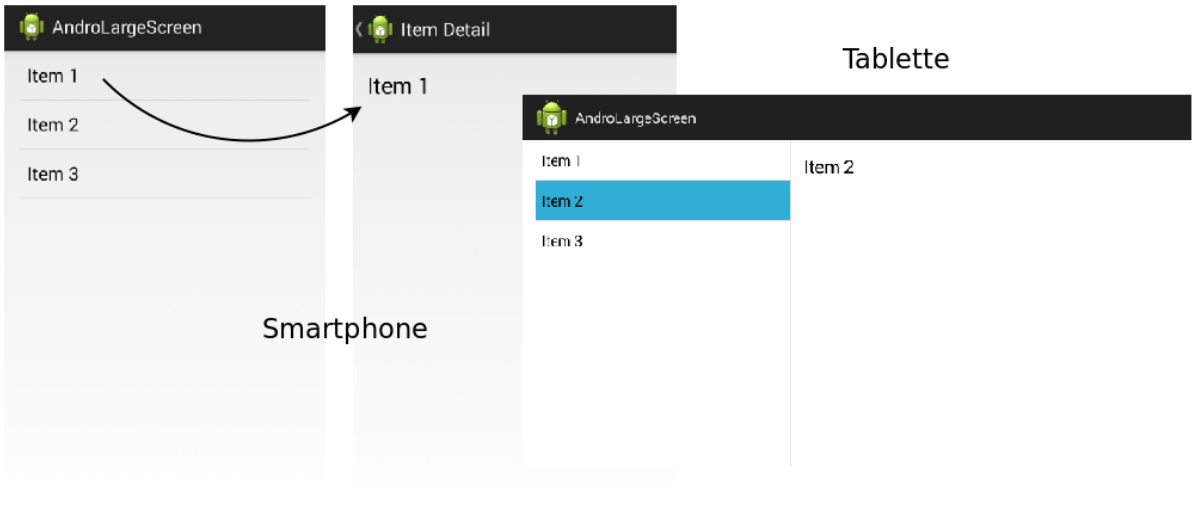
```
public class MainActivity extends Activity {

protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
              setContentView(R.layout.activity_main);
              if (savedInstanceState == null) {
    getFragmentManager().beginTransaction()
      .add(R.id.container, new PlaceholderFragment()).commit();
  }
}
public View onCreateView(LayoutInflater inflater, ViewGroup container,
  Bundle savedInstanceState) {
  View rootView = inflater.inflate(R.layout.fragment_main_dyn, container, false);
  return rootView;
  }
      }}
```

Les fragments d'une activité sont gérés par le *fragment manager*, ce qui complexifie la réalisation du code. Les bénéfices apportés par le *fragment manager* sont les suivants:

- il peut conserver en mémoire l'état du fragment.
- il va conserver chaque *commit*, ce qui permet ensuite à l'utilisateur de revenir en arrière et de retrouver l'état de l'IHM au *commit* précédent.

Gerér la diversité des appareils

Les fragments permettent aussi de gérer la diversité des appareils qui ne disposent pas des mêmes tailles d'écran. Par exemple, on peut créer un fragment de navigation, à gauche d'une tablette, qui contrôle le contenu d'un autre fragment à droite. Il sera affiché en deux écrans successifs sur un téléphone. Cet exemple est disponible dans l'assistant de création d'applications (Master Detail/Flow).



Exemple Master Detail/Flow

Pour contrôler le fragment de droite, on code alors deux comportements dans une *callback*, au niveau de l'activité (l'évènement remonte depuis le fragment versl 'activité). En fonction du type d'appareil, on démarre une activité ou on change le fragment:

```
public void onItemSelected(String id) {
  if (mTwoPane) {
    // In two-pane mode, show the detail view in this activity by
    // adding or replacing the detail fragment using a fragment transaction.
    Bundle arguments = new Bundle();
    arguments.putString(ItemDetailFragment.ARG_ITEM_ID, id);
    ItemDetailFragment fragment = new ItemDetailFragment();
    fragment.setArguments(arguments);
    getFragmentManager().beginTransaction()
      .replace(R.id.item_detail_container, fragment).commit();
  } else {
    // In single-pane mode, simply start the detail activity
    // for the selected item ID.
    Intent detailIntent = new Intent(this, ItemDetailActivity.class);
    detailIntent.putExtra(ItemDetailFragment.ARG_ITEM_ID, id);
    startActivity(detailIntent);
  }}
```

La détection de la taille de l'écran se fait grâce à une valeur de *values-large/refs.xml* qui n'est chargée que si l'écran est large.

(cf. ANX_Interfaces-graphiques)

Cycle de vie d'un fragment

Différentes *callbacks* sont appelées en fonction de l'état de l'activité principale. Un fragment est d'abord attaché à l'activité, puis créé, puis sa vue est créée. On peut même intercaler du code quand la vue de l'activité est créée.

Lorsqu'on change un fragment en préparant une transaction, et que l'on finit par appeler *commit* le fragment précédent est détruit, à moins que l'on est appelé **addToBackStack()** avant le *commit*. Dans ce cas, il est sauvegardé, et si l'utilisateur appuis sur la touche retour, il retrouvera l'interface précédente.

Cela peut donner par exemple:

```
public void loadFragment(Fragment fragmentB, String tag) {
  FragmentManager fm = getSupportFragmentManager();
  FragmentTransaction ft = fm.beginTransaction();
  ft.replace(R.id.fragment_container, fragmentB, tag);

  ft.addToBackStack(null);

  ft.commit();
}
```

Représentation du cycle de vie d'un fragment, par rapport à celui d'une activité:

3.6 Les onglets

La réalisation d'onglets permet de mieux utiliser l'espace réduit de l'écran. Pour réaliser les onglets, les choses se sont considérablement simplifiées depuis l'API 11. Dans l'activité principale, il faut ajouter à

l'*action bar* existante des onglet des objets de type **Tab** ayant obligatoirement un listener de type **TabListener**, comme présenté dans la documentation sur les onglets:

```
final ActionBar actionBar = getActionBar();

  // Specify that tabs should be displayed in the action bar.
  actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
  // Create a tab listener that is called when the user changes tabs.
  ActionBar.TabListener tabListener = new ActionBar.TabListener() {
      public void onTabSelected(ActionBar.Tab tab, FragmentTransaction ft) {
          // show the given tab
      }
      public void onTabUnselected(ActionBar.Tab tab, FragmentTransaction ft) {
          // hide the given tab
      }
      public void onTabReselected(ActionBar.Tab tab, FragmentTransaction ft) {
          // probably ignore this event
  }};
  // Add 3 tabs, specifying the tab's text and TabListener
  for (int i = 0; i < 3; i++) {
      Tab t = actionBar.newTab().setText("Tab " + (i + 1));
      t.setTabListener(tabListener);
      actionBar.addTab(t);
  }
```

Il faut ensuite coder le changement d'écran lorsque l'évenement *onTabSelected* survient. On utilise pour cela l'objet **FragmentTransaction** pour lequel on passe l'*id* du composant graphique à remplacer par un objet de type **Tab**:

```
public void onTabSelected(ActionBar.Tab tab, FragmentTransaction ft) {
FragmentTab newft = new FragmentTab();
if (tab.getText().equals("Tab 2")) {
  newft.setChecked(true);
}
ft.replace(R.id.fragmentContainer, newft);
}
```

La classe **FragmentTab** hérite de **Fragment**: c'est une sous partie d'activité qui a son propre cycle de vie. Dans cet exemple, au lieu de faire 2 fragments différents, on surcharge la méthode *onCreateView* pour cocher la case à cocher:

```
public class FragmentTab extends Fragment {
...
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
          Bundle savedInstanceState)   {
    View view =  inflater.inflate(R.layout.fragment1, container, false);
    CheckBox cb = (CheckBox) view.findViewById(R.id.checkBox1);
    cb.setChecked(this.checked);
    return view;
  }}
```

Le gabarit de l'activité, est donc:

```
<RelativeLayout>
 <TextView
   android:id="@+id/textView1"
   android:text="@string/hello_world" />
  <FrameLayout
   android:id="@+id/fragmentContainer" />
  </FrameLayout>
</RelativeLayout>
```

Et celui du fragment fragment1.xml:

```
<LinearLayout
  <CheckBox android:id="@+id/checkBox1" />
</LinearLayout>
```

Il faut bien remarquer que:

- R.id.fragment1 est l'*id* d'un composant de type **ViewGroup** de l'activité.
- la méthode *inflate* crée la vue en se basant sur le layout *R.layout.fragment1*.

Démonstration

Video

4 Les Intents

4.1 Principe des Intents

Les *Intents* permettent de gérer l'envoi et la réception de messages afin de faire coopérer les applications. Le but des *Intents* est de déléguer une action à un autre composant, une autre application ou une autre activité de l'application courante.

Un objet **Intent** contient les information suivantes:

- le nom du composant ciblé (facultatif)
- l'action à réaliser, sous forme de chaine de caractères
- les données: contenu MIME et URI
- des données supplémentaires sous forme de paires clef/valeur
- une catégorie pour cibler un type d'application
- des drapeaux (informations supplémentaires)

On peut envoyer des *Intents* informatifs pour faire passer des messages. Mais on peut aussi envoyer des *Intents* servant à lancer une nouvelle activité.

4.2 Intents pour une nouvelle activité

Il y a plusieurs façons de créer l'objet de type *Intent* qui permettra de lancer une nouvelle activité. Si l'on passe la main à une activité interne à l'application, on peut créer l'Intent et passer la classe de l'activité ciblée par l'Intent:

```
Intent login = new Intent(this, GiveLogin.class);
startActivity(login);
```

Le premier paramètre de construction de l'Intent est en fait le contexte de l'application. Dans certain cas, il ne faut pas mettre **this** mais faire appel à **getApplicationContext()** si l'objet manipulant l'*Intent* n'hériste pas de **Context**.

S'il s'agit de passer la main à une autre application, on donne au constructeur de l'Intent les données et l'URI cible: l'OS est chargé de trouver une application pouvant répondre à l'Intent.

```
Button b = (Button)findViewById(R.id.Button01);
b.setOnClickListener(new OnClickListener() {
public void onClick(View v) {
  Uri telnumber = Uri.parse("tel:0248484000");
  Intent call = new Intent(Intent.ACTION_DIAL, telnumber);
  startActivity(call);
}
});
```

Sans oublier de déclarer la nouvelle activité dans le Manifest.

Retour d'une activité

Lorsque le bouton *retour* est pressé, l'activité courante prend fin et revient à l'activité précédente. Cela permet par exemple de terminer son appel téléphonique et de revenir à l'activité ayant initié l'appel.

Au sein d'une application, une activité peut vouloir récupérer un code de retour de l'activité "enfant". On utilise pour cela la méthode **startActivityForResult** qui envoie un code de retour à l'activité enfant. Lorsque l'activité parent reprend la main, il devient possible de filtrer le code de retour dans la méthode **onActivityResult** pour savoir si l'on revient ou pas de l'activité enfant.

L'appel d'un *Intent* devient donc:

```
public void onCreate(Bundle savedInstanceState) {
  Intent login = new Intent(getApplicationContext(), GivePhoneNumber.class);
  startActivityForResult(login,48);
... }
```

Le filtrage dans la classe parente permet de savoir qui avait appelé cette activité enfant:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
  if (requestCode == 48)
    Toast.makeText(this, "Code de requête récupéré (je sais d'ou je viens)",
                   Toast.LENGTH_LONG).show();
}
```

Résultat d'une activité

Il est aussi possible de définir un résultat d'activité, avant d'appeler explicitement la fin d'une activité avec la méthode **finish()**. Dans ce cas, la méthode **setResult** permet d'enregistrer un code de retour qu'il sera aussi possible de filtrer dans l'activité parente.

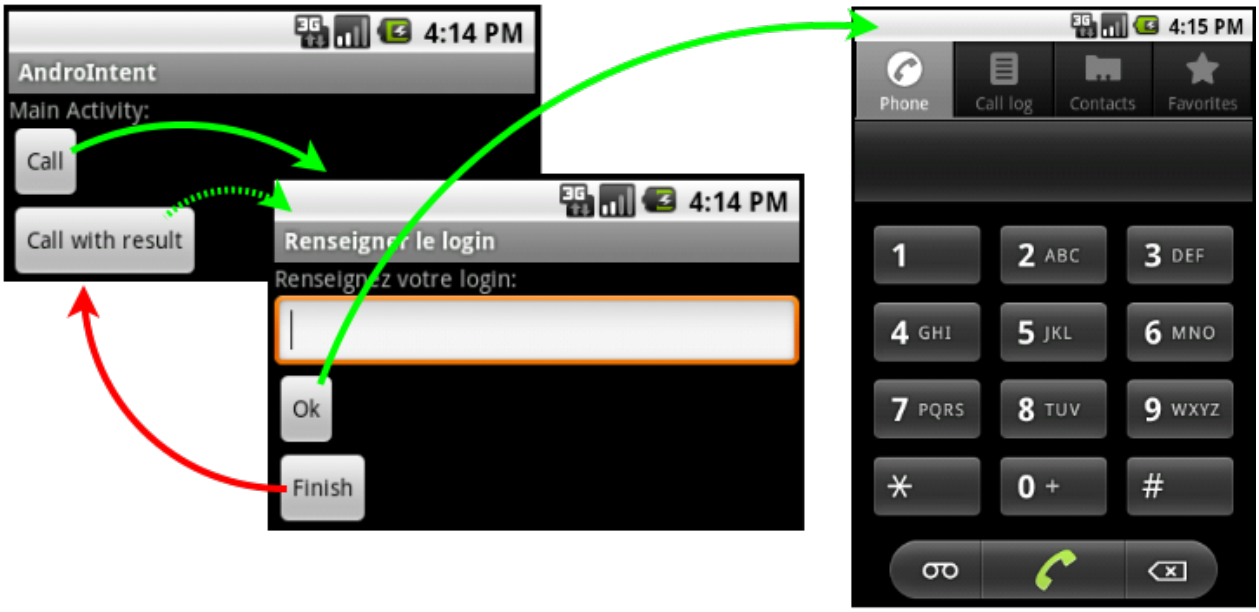
Dans l'activité enfant, on met donc:

```
Button finish = (Button)findViewById(R.id.finish);
  finish.setOnClickListener(new OnClickListener() {

  @Override
  public void onClick(View v) {
    setResult(50);
    finish();
  }});
```

Et la classe parente peut filtrer ainsi:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
  if (requestCode == 48)
    Toast.makeText(this, "Code de requête récupéré (je sais d'ou je viens)",
                   Toast.LENGTH_LONG).show();
  if (resultCode == 50)
    Toast.makeText(this, "Code de retour ok (on m'a renvoyé le bon code)",
                   Toast.LENGTH_LONG).show();
}
```

Principe de la Démonstration



(cf ANX_Intents)

Démonstration

Video

4.3 Ajouter des informations

Les *Intent* permettent de transporter des informations à destination de l'activité cible. On appelle ces informations des *Extra*: les méthodes permettant de les manipuler sont **getExtra** et **putExtra**. Lorsqu'on prépare un *Intent* et que l'on souhaite ajouter une information de type "clef -> valeur" on procéde ainsi:

```
Intent callactivity2 = new Intent(getApplicationContext(), Activity2.class);
callactivity2.putExtra("login", "jfl");
startActivity(callactivity2);
```

Du côté de l'activité recevant l'Intent, on récupère l'information de la manière suivante:

```
Bundle extras = getIntent().getExtras();
String s = new String(extras.getString("login"));
```

4.4 Types d'Intent: actions

Le premier paramètre de construction de l'*Intent* est le type de l'action véhiculé par cet Intent. Ces types d'actions peuvent être les actions natives du système ou des actions définies par le développeur.

Plusieurs actions natives existent par défaut sur Android. La plus courante est l'action **Intent.ACTION_VIEW** qui permet d'appeler une application pour visualiser un contenu dont on donne l'URI.

Voici un exemple d'envoi d'email (issu de [E-mail]). Dans ce cas ou l'on utilise l'action native, il faut ajouter des informations supplémentaires à l'aide de **putExtra**:

```
Intent emailIntent = new Intent(android.content.Intent.ACTION_SEND);
String[] recipients = new String[]{"my@email.com", "",};
emailIntent.putExtra(android.content.Intent.EXTRA_EMAIL, recipients);
emailIntent.putExtra(android.content.Intent.EXTRA_SUBJECT, "Test");
emailIntent.putExtra(android.content.Intent.EXTRA_TEXT, "Message");
emailIntent.setType("text/plain");
startActivity(Intent.createChooser(emailIntent, "Send mail..."));
finish();
```

Pour définir une action personelle, il suffit de créer une chaine unique:

```
Intent monIntent = new Intent("andro.jf.nom_du_message");
```

Types d'Intent: catégories

Les catégories d'*Intent* permettent de grouper les applications par grands types de fonctionnalités (clients emails, navigateurs, players de musique, etc...). Par exemple, on trouve les catégories suivantes qui permettent de lancer:

- **DEFAULT**: catégorie par défaut
- **BROWSABLE**: une activité qui peut être invoquée depuis un clic sur un navigateur web, ce qui permet d'implémenter des nouveaux types de lien, e.g. foo://truc

- **APP_MARKET**: une activité qui permet de parcourir le market de télécharger des applications
- **APP_MUSIC**: une activité qui permet de parcourir et jouer de la musique
- ...

4.5 Broadcaster des informations

Il est aussi possible d'utiliser un objet **Intent** pour broadcaster un message à but informatif. Ainsi, toutes les applications pourront capturer ce message et récupérer l'information.

```
Intent broadcast = new Intent("andro.jf.broadcast");
broadcast.putExtra("extra", "test");
sendBroadcast(broadcast);
```

La méthode **putExtra** permet d'enregistrer une paire clef/valeur dans l'Intent. On peut récupérer les données à l'aide de la méthode **getExtras** dans l'objet **Bundle** qui est dans l'Intent:

```
Bundle extra = intent.getExtras();
String val = extra.getString("extra");
```

4.6 Recevoir et filtrer les Intents

Etant donné la multitude de messages véhiculés par des *Intent*, chaque application doit pouvoir facilement "écouter" les *Intents* dont elle a besoin. Android dispose d'un système de filtres déclaratifs permettant de définir dans le Manifest des filtres.

Un filtre peut utiliser plusieurs niveaux de filtrage:

- action: identifie le nom de l'Intent. Pour éviter les collisions, il faut utiliser la convention de nommage de Java
- category: permet de filtrer une catégorie d'action (DEFAULT, BROWSABLE, ...)
- data: filtre sur les données du message par exemple en utilisant android:host pour filtrer un nom de domaine particulier

Filtrage d'un Intent par l'activité

En déclarant un filtre au niveau du tag **activity**, l'application déclare les types de messsage qu'elle sait gérer et qui l'invoquent.

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

Ainsi, l'application répond à la sollicitation des *Intents* envoyés par:

```
Button autoinvoc = (Button)findViewById(R.id.autoinvoc);
autoinvoc.setOnClickListener(new OnClickListener() {
@Override
```

```
public void onClick(View v) {
  Intent intent = new Intent("andro.jf.nom_du_message");
  startActivity(intent);
}});
```

Utilisation de catégories pour le filtrage

Les catégories d'*Intent* évoquées précédemment permettent de répondre à des évènements dans un contexte particulier. L'exemple le plus utilisé est l'interception d'un clic lors d'une navigation sur le web ou un protocole particulier est prévu. Par exemple, les liens vers le market sont de la forme **market://**.

Pour par exemple répondre à un clic sur un lien **foo://ensi-bourges.fr/mapage**, il faut construire un filtre d'*Intent* utilisant la catégorie définissant ce qui est "navigable" tout en combinant cette catégorie avec un type d'action signifiant que l'on souhaite "voir" la ressource. Il faut en plus utiliser le tag *data* dans la construction du filtre:

```
<intent-filter>
  <action android:name="android.intent.action.VIEW" />
  <category android:name="android.intent.category.DEFAULT" />
  <category android:name="android.intent.category.BROWSABLE" />
  <data android:scheme="foo" android:host="ensi-bourges.fr"/>
</intent-filter>
```

L'attribut *host* définit l'URI de l'autorité (on évite ainsi le fishing). L'attribut *scheme* définit la partie de gauche de l'URI. On peut ainsi lancer une activité lors d'un clic sur:

```
<a href="foo://ensi-bourges.fr/mapage">lien</a>
```

Réception par un BroadcastReceiver

Les messages broadcastés par les applications sont réceptionnées par une classe héritant de **BroadcastReceiver**. Cette classe doit surcharger la méthode **onReceive**. Dans le Manifest, un filtre doit être inséré dans un tag **receiver** qui pointe vers la classe se chargeant des messages.

Dans le Manifest:

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
  <receiver android:name="MyBroadcastReceiver">
    <intent-filter>
            <action android:name="andro.jf.broadcast" />
            <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
  </receiver>
</application>
```

La classe héritant de **BroadcastReceiver**:

```
public final class MyBroadcastReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    Bundle extra = intent.getExtras();
```

```
    if (extra != null)
    {
      String val = extra.getString("extra");
      Toast.makeText(context, "Broadcast message received: " + val,
                     Toast.LENGTH_SHORT).show();
}}}
```

Récepteur d'Intent dynamique

Il est possible de créer un récepteur et d'installer le filtre correspondant dynamiquement.

```
private MyBroadcastReceiverDyn myreceiver;

public void onCreate(Bundle savedInstanceState) {
  // Broadcast receiver dynamique
  myreceiver = new MyBroadcastReceiverDyn();
  IntentFilter filtre = new IntentFilter("andro.jf.broadcast");
  registerReceiver(myreceiver, filtre);
}
```

Lorsque ce receiver n'est plus utile, il faut le désenregistrer pour ainsi libérer les ressources:

```
unregisterReceiver(myreceiver);
```

C'est particulièrement important puisque un filtre de réception reste actif même si l'application est éteinte (elle peut être relancée si un message la concernant survient).

Les messages natifs

Un certain nombre de messages sont diffusés par l'OS:

- **ACTION_BOOT_COMPLETED**: diffusé lorsque le système a fini son boot
- **ACTION_SHUTDOWN**: diffusé lorsque le système est en cours d'extinction
- **ACTION_SCREEN_ON / OFF**: allumage / exctinction de l'écran
- **ACTION_POWER_CONNECTED / DISCONNECTED**: connexion / perte de l'alimentation
- **ACTION_TIME_TICK**: une notification envoyée toutes les minutes
- **ACTION_USER_PRESENT**: notification recue lorsque l'utilisateur délock son téléphone
- ...

Tous les messages de broadcast se trouvent dans la documentation des *Intents*.

D'autres actions permettent de lancer des applications tierces pour déléguer un traitement:

- **ACTION_CALL (ANSWER, DIAL)**: passer/réceptionner/afficher un appel
- **ACTION_SEND**: envoyer des données par SMS ou E-mail
- **ACTION_WEB_SEARCH**: rechercher sur internet

Principe de la Démonstration

Démonstration

Video

(cf ANX_Receveur-de-broadcasts)

5 Persistance des données

5.1 Différentes persistances

Android fournit plusieurs méthodes pour faire persister les données applicatives:

- la persistance des données de l'activité (cf Le SDK Android)
- un mécanisme de sauvegarde clé/valeur, utilisé pour les fichiers de préférences (appelé préférences partagées)
- des entrées sorties de type fichier
- une base de donnée basé sur SQLite

La persistance des données des activités est géré par l'objet **Bundle** qui permet de restaurer les **View** qui possède un *id*. S'il est nécessaire de réaliser une sauvegarde plus personnalisée, il suffit de recoder les méthodes **onSaveInstanceState** et **onCreate** et d'utiliser les méthodes qui permettent de lire/écrire des données sur l'objet **Bundle**.

Android fournit aussi automatiquement, la persistance du chemin de navigation de l'utilisateur, ce qui le renvoie à la bonne activité lorsqu'il appuie sur la touche Retour. La navigation vers le parent (bouton "up") n'est pas automatique car c'est le concepteur qui doit décider vers quelle activitée l'application doit retourner quand on appuie sur "up". Elle peut être programmée dans le Manifest avec l'attribut android:parentActivityName.

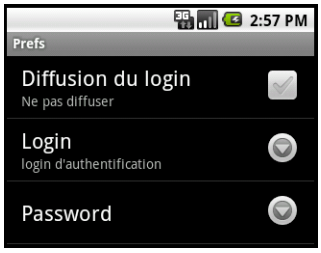
5.2 Préférences partagées

La classe **SharedPreferences** permet de gérer des paires de clé/valeurs associées à une activité. On récupère un tel objet par l'appel à **getPreferences**:

```
SharedPreferences prefs = getPreferences(Context.MODE_PRIVATE);
String nom = prefs.getString("login", null);
Long nom = prefs.getLong("taille", null);
```

La méthode **getPreferences(int)** appelle en fait **getPreferences(String, int)** à partir du nom de la classe de l'activité courante. Le mode **MODE_PRIVATE** restreint l'accès au fichier créé à l'application. Les modes d'accès **MODE_WORLD_READABLE** et **MODE_WORLD_WRITABLE** permettent aux autres applications de lire/écrire ce fichier.

L'intérêt d'utiliser le système de préférences prévu par Android réside dans le fait que l'interface graphique associé à la modification des préférences est déjà programmé: pas besoin de créer l'interface de l'activité pour cela. L'interface sera générée automatiquement par Android et peut ressembler par exemple à cela:



Représentation XML d'un menu de préférences

Une activité spécifique a été programmée pour réaliser un écran d'édition de préférences. Il s'agit de **PreferenceActivity**. A partir d'une description XML des préférences, la classe permet d'afficher un écran composé de modificateurs pour chaque type de préférences déclarées.

Voici un exemple de déclarations de préférences XML, à stocker dans *res/xml/preferences.xml*:

```
<PreferenceScreen
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:key="first_preferencescreen">
  <CheckBoxPreference
          android:key="wifi enabled"
          android:title="WiFi" />
  <PreferenceScreen
          android:key="second_preferencescreen"
          android:title="WiFi settings">
      <CheckBoxPreference
              android:key="prefer wifi"
              android:title="Prefer WiFi" />
      ... other preferences here ...
  </PreferenceScreen>
</PreferenceScreen>
```

Activité d'édition de préférences

Pour afficher l'écran d'édition des préférences correspondant à sa description XML, il faut créer une nouvelle activité qui hérite de **PreferenceActivity** et simplement appeler la méthode **addPreferencesFromResource** en donnant l'id de la description XML:

```
public class MyPrefs extends PreferenceActivity {
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.preferences);
}}
```

Pour lancer cette activité, on crée un bouton et un **Intent** correspondant.

Attributs des préférences

Les attributs suivants sont utiles:

- **android:title**: La string apparaissant comme nom de la préférence
- **android:summary**: Une phrase permettant d'expliciter la préférence
- **android:key**: La clef pour l'enregistrement de la préférence

Pour accéder aux valeurs des préférences, on utiliser la méthode **getDefaultSharedPreferences** sur la classe **PreferenceManager**. C'est la clef spécifiée par l'attribut **android:key** qui est utilisée pour récupérer la valeur choisie par l'utilisateur.

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(
      getApplicationContext());
String login = prefs.getString("login","");
```

Des attributs spécifiques à certains types de préférences peuvent être utilisés, par exemple **android:summaryOn** pour les cases à cocher qui donne la chaine à afficher lorsque la préférence est cochée. On peut faire dépendre une préférence d'une autre, à l'aide de l'attribut **android:dependency**. Par exemple, on peut spécifier dans cet attribut le nom de la clef d'une préférence de type case à cocher:

```
<CheckBoxPreference android:key="wifi" ... />
<EditTextPreference android:dependency="wifi" ... />
```

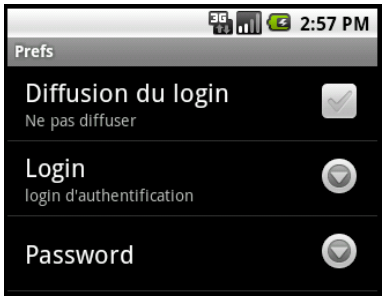
Préférences: toggle et texte

Une case à cocher se fait à l'aide de **CheckBoxPreference**:

```
<CheckBoxPreference android:key="wifi"
  android:title="Utiliser le wifi"
  android:summary="Synchronise l'application via le wifi."
  android:summaryOn="L'application se synchronise via le wifi."
  android:summaryOff="L'application ne se synchronise pas."
/>
```

Un champs texte est saisi via **EditTextPreference**:

```
<EditTextPreference android:key="login&"
  android:title="Login utilisateur"
  android:summary="Renseigner son login d'authentification."
  android:dialogTitle="Veuillez saisir votre login"
/>
```



Préférences: listes

Une entrée de préférence peut être liée à une liste de paires de clef-valeur dans les ressources:

```
<resources>
<array name="key"> <!-- Petite=1, Moyenne=5, Grande=20 -->
              <item>"Petite"</item>
              <item>"Moyenne"</item>
              <item>"Grande"</item>
      </array>
      <array name="value">
              <item>"1"</item>
              <item>"5"</item>
              <item>"20"</item>
      </array>
</resources>
```

qui se déclare dans le menu de préférences:

```
<ListPreference android:title="Vitesse"
  android:key="vitesse"
  android:entries="@array/key"
  android:entryValues="@array/value"
  android:dialogTitle="Choisir la vitesse:"
  android:persistent="true">
</ListPreference>
```

Lorsque l'on choisit la valeur "Petite", la préférence *vitesse* est associée à "1".

Ecriture de préférences

Il est aussi possible d'écraser des préférences par le code, par exemple si une action fait changer le paramétrage de l'application ou si l'on recoit le paramétrage par le réseau.

L'écriture de préférences est plus complexe: elle passe au travers d'un éditeur qui doit réaliser un *commit* des modifications (*commit* atomique, pour éviter un mix entre plusieurs écritures simultanées):

```
SharedPreferences prefs = getPreferences(Context.MODE_PRIVATE);
Editor editor = prefs.edit();
editor.putString("login", "jf");
editor.commit();
```

Il est même possible de réagir à un changement de préférences en installant un écouteur sur celles-ci:

```
prefs.registerOnSharedPreferenceChangeListener(
      new OnSharedPreferenceChanged () {
            ...
});
```

Démonstration

Video

(cf ANX_Gestion-des-préférences)

5.3 Les fichiers

Android fournit aussi un accès classique au système de fichier pour tous les cas qui ne sont pas couverts par les préférences ou la persistance des activités, c'est à dire de nombreux cas. Le choix de Google est de s'appuyer sur les classes classiques de Java EE tout en simplifiant la gestion des permissions et des fichiers embarqués dans l'application.

Pour la gestion des permissions, on retrouve comme pour les préférences les constantes **MODE__PRIVATE** et **MODE_WORLD_READABLE/WRITABLE** à passer en paramètre de l'ouverture du fichier. En utilisant ces constantes comme un masque, on peut ajouter **| MODE_APPEND** pour ajouter des données.

```
try {
  FileOutputStream out = openFileOutputStream("fichier", MODE_PRIVATE);
  ...
} catch (FileNotFoundException e) { ... }
```

Les ressources permettent aussi de récupérer un fichier embarqué dans l'application:

```
Resources res = getResources();
InputStream is = res.openRawResource(R.raw.fichier);
```

A noter: les fichier sont à éviter. Mieux vaut alléger l'application au maximum et prévoir le téléchargement de ressources nécessaires et l'utilisation de fournisseurs de contenu.

5.4 BDD SQLite

Android dispose d'une base de donnée relationelle basée sur SQLite. Même si la base doit être utilisée avec parcimonie, cela fournit un moyen efficace de gérer une petite quantité de données.

[DBSQL] présente un exemple de création de base de données, ce qui se fait en héritant de **SQLiteOpenHelper**:

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

  private static final int DATABASE_VERSION = 2;
  private static final String DICTIONARY_TABLE_NAME = "dictionary";
  private static final String DICTIONARY_TABLE_CREATE =
              "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
```

```
                KEY_WORD + " TEXT, " +
                KEY_DEFINITION + " TEXT);";

    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

Lecture / Ecriture dans la BDD

Pour réaliser des écritures ou lectures, on utilise les méthodes **getWritableDatabase()** et **getReadableDatabase()** qui renvoient une instance de **SQLiteDatabase**. Sur cet objet, une requête peut être exécutée au travers de la méthode **query()**:

```
public Cursor query (boolean distinct, String table, String[] columns,
                     String selection, String[] selectionArgs, String groupBy,
                     String having, String orderBy, String limit)
```

L'objet de type **Cursor** permet de traiter la réponse (en lecture ou écriture), par exemple:

- **getCount()**: nombre de lignes de la réponse
- **moveToFirst()**: déplace le curseur de réponse à la première ligne
- **getInt(int columnIndex)**: retourne la valeur (int) de la colonne passée en paramètre
- **getString(int columnIndex)**: retourne la valeur (String) de la colonne passée en paramètre
- **moveToNext()**: avance à la ligne suivante
- **getColumnName(int)**: donne le nom de la colonne désignée par l'index
- ...

5.5 XML

[XML] Sans surprise, Android fournit plusieurs parsers XML (*Pull parser*, *Document parser*, *Push parser*). SAX et DOM sont disponibles. L'API de *streaming* (StAX) n'est pas disponible (mais pourrait le devenir). Cependant, une librairie équivalente est disponible: **XmlPullParser**. Voici un exemple simple:

```
String s = new String("<plop><blup attr=\"45\">Coucou !</blup></plop>");
InputStream f = new ByteArrayInputStream(s.getBytes());
XmlPullParser parser = Xml.newPullParser();
try {
    // auto-detect the encoding from the stream
    parser.setInput(f, null);

    parser.next();
    Toast.makeText(this, parser.getName(), Toast.LENGTH_LONG).show();
```

```
    parser.next();
    Toast.makeText(this, parser.getName(), Toast.LENGTH_LONG).show();
    Toast.makeText(this, parser.getAttributeValue(null, "attr"),
                                        Toast.LENGTH_LONG).show();
    parser.nextText();
    Toast.makeText(this, parser.getText(), Toast.LENGTH_LONG).show();
} ...
```

Video

SAX parser

SAX s'utilise très classiquement, comme en *Java SE*. Voici un exemple attaquant du XML au travers d'une connexion http:

```
URI u = new URI("https://www.site.com/api/xml/list?apikey=845ef");
DefaultHttpClient httpclient = new DefaultHttpClient();
HttpGet httpget = new HttpGet(u);
HttpResponse response = httpclient.execute(httpget);
HttpEntity entity = response.getEntity();
InputStream stream = entity.getContent();
InputSource source = new InputSource(stream);

SAXParserFactory spf = SAXParserFactory.newInstance();
SAXParser sp = spf.newSAXParser();
XMLReader xr = sp.getXMLReader();
DefaultHandler handler = new DefaultHandler() {
        @Override
        public void startElement(String uri, String localName,
                    String qName, Attributes attributes) throws SAXException {

        Vector monitors = new Vector();

        if (localName.equals("monitor")) {
            monitors.add(attributes.getValue("displayname"));
        }}
};
xr.setContentHandler(handler);
xr.parse(source);
```

DOM parser

Enfin, le paser DOM permet de naviguer assez facilement dans la représentation arborescente du document XML. Ce n'est évidemment pas préconisé si le XML en question est volumineux. L'exemple suivant permet de chercher tous les tags "monitor" dans les sous-tags de la racine.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
try {
  DocumentBuilder builder = factory.newDocumentBuilder();
  Document dom = builder.parse(source);
  Element root = dom.getDocumentElement();
```

```
// Récupère tous les tags du descendant de la racine s'appelant monitor
NodeList items = root.getElementsByTagName("monitor");
for (int i=0;i<items.getLength();i++){
  Node item = items.item(i);
  // Traitement:
  // item.getNodeName()
  // item.getTextContent()
  // item.getAttributes()
  ...
  }
} catch (...)
```

6 Programmation concurrente

6.1 Composants d'une application

Une application Android n'est pas qu'une simple activité. Dans de nombreux cas, une application Android est amenée à tirer partie de la programmation concurrente afin de réaliser des tâches parallèles.

Dans ce chapitre, on s'intéresse à la notion de processus:

- processus "lourd" appelé par la suite "processus"
- processus "léger" appelé par la suite *thread*

Une fois ces deux notions présentées, nous verrons comment réaliser des tâches particulières qui sont souvent concurrente à l'activité principale. Ces tâches s'appuient principalement sur les *threads*, mais aussi sur un nouveau composant d'une application appelé "service".

6.2 Processus

[PT] Par défaut, une application android s'exécute dans un processus unique, le processus dit "principal". L'interface graphique s'exécute elle aussi au travers de ce processus principal. Ainsi, plus une application réalisera de traitement, plus la gestion de la programmation devient délicate car on peut arriver très rapidement à des problèmes de blocage de l'interface graphique par exemple.

Il est possible de séparer les composants d'une application en plusieurs processus. Ces composants sont les codes attachés aux tags <activity>, <service>, <receiver>, et <provider> de l'application. Par défaut, l'application (tag <application>) s'exécute dans le processus principal portant le même nom que le *package* de l'application. Bien que cela soit inutile, on peut le préciser dans le Manifest à l'aide de l'attribut **android:process**, par exemple:

```
<application android:process="andro.jf">
```

Les sous-tags du manifest, c'est-à-dire les composants de l'application héritent de cet attribut et s'exécutent donc dans le même processus. Si l'on souhaite créer un nouveau processus indépendant, on peut utiliser l'attribut **android:process** en préfixant le nom du processus par ":", par exemple:

```
<service android:name=".AndroService" android:process=":p2">
```

Pour cet exemple, le service et l'application sont indépendants. L'interface graphique pourra s'afficher indépendamment.

Vie des processus

Android peut devoir arrêter un processus à cause d'autres processus qui requièrent plus d'importance. Par exemple, un service peut être détruit car une application gourmande en ressources (navigateur web) occupe de plus en plus de mémoire. Plus classiquement, le processus affecté à une activité qui n'est plus visible a de grandes chances d'être détruit.

Ainsi, une hiérarchie permet de classer le niveau d'importance des processus:

1. Processus en avant plan (activité en interaction utilisateur, service attaché à cette activité, **BroadCastReceiver** exécutant **onReceive()**)
2. Processus visible: il n'interagit pas avec l'utilisateur mais peut influer sur ce que l'on voit à l'écran (activité ayant affiché une boite de dialogue (**onPause()** a été appelé), service lié à ces activité "visibles").

3. Processus de service
4. Processus tâche de fond (activité non visible (**onStop()** a été appelé))
5. Processus vide (ne comporte plus de composants actifs, gardé pour des raisons de *cache*)

Ainsi, il faut préférer l'utilisation d'un service à la création d'un thread pour accomplir une tâche longue, par exemple l'*upload* d'une image. On garantit ainsi d'avoir le niveau 3 pour cette opération, même si l'utilisateur quitte l'application ayant initié l'*upload*.

6.3 Threads

Dans le processus principal, le système crée un *thread* d'exécution pour l'application: le *thread* principal. Il est, entre autre, responsable de l'interface graphique et des messages/notifications/événements entre composants graphiques. Par exemple, l'événement générant l'exécution de **onKeyDown()** s'exécute dans ce *thread*.

Ainsi, si l'application doit effectuer des traitements longs, elle doit éviter de les faire dans ce thread. Cependant, il est interdit d'effectuer des opérations sur l'interface graphique en dehors du thread principal (aussi appelé *UI thread*), ce qui se résume dans la documentation sur les threads par deux règles:

- *Do not block the UI thread*
- *Do not access the Android UI toolkit from outside the UI thread*

L'exemple à ne pas faire est donné dans la documentation et recopié ci-dessous. Le comportement est imprévisible car le toolkit graphique n'est pas *thread-safe*.

```
public void onClick(View v) { // DO NOT DO THIS !
  new Thread(new Runnable() {
    public void run() {
      Bitmap b = loadImageFromNetwork("http://example.com/image.png");
      mImageView.setImageBitmap(b); }
  }).start(); }
```

Une exception peut parfois (pas toujours) être levée quand cela est détecté: **CalledFromWrongThreadException**.

Threads et interface graphique

Android fournit des méthodes pour résoudre le probème précédemment évoqué. Il s'agit de créer des objets exécutables dont la partie affectant l'interface graphique n'est pas exécutée mais déléguée à l'*UI thread* pour exécution ultérieure.

Par exemple, un appel à la méthode **View.post(Runnable)** permet de réaliser cela et donne, pour l'exemple précédent:

```
public void onClick(View v) {
new Thread(new Runnable() {
    public void run() {
    final Bitmap bitmap =
      loadImageFromNetwork("http://example.com/image.png");
    mImageView.post(new Runnable() {
      public void run() {
      mImageView.setImageBitmap(bitmap);
      }
```

```
      });
    }
    }).start(); }
```

La documentation donne les quelques méthodes utiles pour ces cas délicats:

- **Activity.runOnUiThread(Runnable)**
- **View.post(Runnable)**
- **View.postDelayed(Runnable, long)**

6.4 Services

La structure d'une classe de service ressemble à une activité. Pour réaliser un service, on hérite de la classe **Service** et on implémente les méthodes de création/démarrage/arrêt du service. La nouvelle méthode qu'il faut implémenter ici est **onBind** qui permet aux IPC de faire des appels à des méthodes distantes.

```
public class MyService extends Service {
  public void onCreate() {
    // Création du service
  }
  public void onDestroy() {
    // Destruction du service
  }
  @deprecated // utile pour les versions antérieures d'Android 2.0
  public void onStart(Intent intent, int startId) {
    // Démarrage du service
  }
  public int onStartCommand(Intent intent, int flags, int startId) {
    // Démarrage du service
    return START_STICKY;
  }
  public IBinder onBind(Intent arg0) {
    return null;
}}
```

Le service se déclare dans le Manifest dans le tag *application*:

```
<service android:name=".MyService"/>
```

Démarrer / Arrêter un service

Les *Intents* fournissent un moyen de démarrer/arrêter un service en utilisant le nom de la classe du service ou un identifiant d'action:

```
startService(new Intent(this, AndroService.class));
stopService(new Intent(this, AndroService.class));
startService(new Intent("andro.jf.manageServiceAction"));
```

Ces méthodes sont à invoquer dans l'activité de l'application développée. On dit alors que le service est *local* à l'application: il va même s'exécuter dans le *thread* de l'application. Un bouton de démarrage d'un service local est simple à coder:

```
Button b = (Button) findViewById(R.id.button1);
b.setOnClickListener(new OnClickListener() {

  public void onClick(View v) {
    Intent startService = new Intent("andro.jf.manageServiceAction");
    startService(startService);
  }
});
```

Auto démarrage d'un service

Il est aussi possible de faire démarrer un service au démarrage du système. Pour cela, il faut créer un **BroadcastReceiver** qui va réagir à l'action **BOOT_COMPLETED** et lancer l'**Intent** au travers de la méthode **startService**.

Le Manifest contient donc:

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
<service android:name=".AndroService"></service>
<receiver android:name=".AutoStart">
<intent-filter>
  <action android:name="android.intent.action.BOOT_COMPLETED" />
</intent-filter>
</receiver>
</application>
```

Et la classe **AutoStart** doit être:

```
public class AutoStart extends BroadcastReceiver {
  public void onReceive(Context context, Intent intent) {
    Intent startServiceIntent = new Intent(context, AndroService.class);
    context.startService(startServiceIntent);
  }
}
```

onStartCommand()

Un problème subsiste après que l'application ait démarré le service. Typiquement, le service démarré exécute **onCreate()** puis **onStart()** qui va par exemple lancer un **Thread** pour réaliser la tâche de fond. Si le service doit être détruit par la plate-forme (pour récupérer de la mémoire), puis est récréé, seule la méthode **onCreate()** est appelée [API-service].

Afin de résoudre le problème précédent, une nouvelle méthode a fait son apparition dans les versions ultérieures à la version 5 de l'API. La méthode **onStartCommand()** est très similaire à **onStart()** mais cette méthode renvoie un entier qui permet d'indiquer au système ce qu'il faut faire au moment de la ré-instanciation du service, si celui-ci a dû être arrêté. La méthode peut renvoyer (cf [API-service]):

- **START_STICKY**: le comportement est similaire aux API<5. Si le service est tué, il est ensuite redémarré. Cependant, le système prend soin d'appeler à nouveau la méthode **onStartCommand(Intent intent)** avec un *Intent* null, ce qui permet au service qu'il vient d'être démarré après un *kill* du système.
- **START_NOT_STICKY**: le service n'est pas redémarré en cas de *kill* du système. C'est utile lorsque le service réagit à l'envoi d'un *Intent* unique, par exemple une alarme qui envoie un *Intent* toutes les 15 minutes.
- **START_REDELIVER_INTENT**: similaire à **START_NOT_STICKY** avec en plus le rejoue d'un *Intent* si le service n'a pas pu finir de le traiter et d'appeler **stopSelf()**.

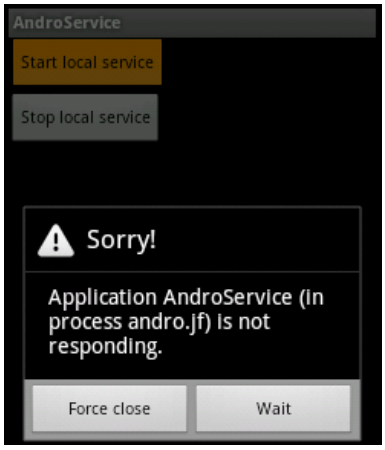
Dans les versions ultérieures à l'API 5, la commande **onStart()** se comporte comment **onStartCommand()** avec un ré-appel de la méthode avec un *Intent* null. Pour conserver le comportement obsolète de l'API (pas d'appel à **onStart()**), on peut utiliser le flag **START_STICKY_COMPATIBILITY**.

Service dans le thread principal

Le service invoqué par l'*Intent* de l'application s'exécute dans le thread de l'application. Cela peut être particulièrement ennuyeux si les traitements à effectuer sont couteux, par exemple si l'on souhaite faire:

```
boolean blop = true;
while (blop == true)
  ;
```

On obtient:



Service dans un processus indépendant

Une manière de résoudre le problème précédent est de demander à Android d'exécuter le service dans un processus indépendant en le déclarant dans le Manifest:

```
<service android:name=".AndroService" android:process=":p2">
```

Cependant, cela n'autorise pas le service à réaliser un traitement long dans la méthode **onStart()** (comme une boucle infinie). La méthode **onStart()** sert à créer des threads, des tâches asynchrones, des tâches programmées, qui vont s'exécuter en tâche de fond.

En fait, isoler le service dans un processus à part rend la programmation plus difficile, car la programmation de la communication entre l'activité principale et le service devient plus difficile (il faut

utiliser AIDL). Il est donc plus simple de laisser le service dans le processus principal de l'application et de lui faire lancer des threads ou tâches asynchrones.

Démonstration

Dans cette démonstration, le service, qui comporte une boucle infinie, s'exécute dans un processus indépendant de l'interface principale. Android va tuer le service qui occupe trop de temps processeur, ce qui n'affectera pas l'application et son interface (cf ANX_Processus-indépendants).

Video

6.5 Tâches concurrentes

En plus des classes de programmation concurrentes de Java (**Thread**, **Executor**, **ThreadPoolExecutor**, **FutureTask**, **TimerTask**, etc...), Android fournit quelques classes supplémentaires pour programmer des tâches concurrentes (**ASyncTask**, **IntentService**, etc...).

Si la tâche concurrente est lancée:

- par l'activité principale: sa vie durera le temps de l'activité
- par un service: la tâche survivra à l'activité principale

Tâche régulière

S'il s'agit de faire une tâche répétitive qui ne consomme que très peu de temps CPU à intervalles réguliers, une solution ne nécessitant pas de processus à part consiste à programmer une tâche répétitive à l'aide d'un **TimerTask**.

```
final Handler handler = new Handler();
task = new TimerTask() {
  public void run() {
    handler.post(new Runnable() {
      public void run() {
        Toast.makeText(AndroService.this, "plop !", Toast.LENGTH_SHORT).show();
      } });
}};
timer.schedule(task, 0, 5000);
```

La création du **TimerTask** est particulièrement tarabiscotée mais il s'agissait de résoudre le problème du lancement du Toast. Pour des cas sans *toast*, un simple appel à **new TimerTask() { public void run() { code; } };** suffit.

Attention à bien détruire la tâche programmée lorsqu'on détruit le service ! (sinon, le *thread* associé à la tâche programmée reste actif).

```
public void onDestroy() { // Destruction du service
              timer.cancel(); }
```

Démonstration

Dans cette démonstration, le service local à l'application est lancé: il affiche toutes les 5 secondes un *toast* à l'aide d'un **TimerTask**. Il est visible dans la section *running services*. En cliquant une seconde fois sur le bouton start, deux services s'exécutent en même temps. Enfin, les deux services sont stoppés. (cf ANX_Demarrage-de-services)

Video

Tâches asynchrones

Dans [AT], Gilles Debunne présente comment gérer le chargement d'image de façon asynchrone afin d'améliorer la fluidité d'une application. Dans le cas d'une liste d'image chargée depuis le web, c'est même obligatoire, sous peine de voir planter l'application.

Pour réaliser cela, il faut se baser sur la classe **AsyncTask<U,V,W>** basée sur 3 types génériques, cf [ATK]:

- U: le type du paramètre envoyé à l'exécution
- V: le type de l'objet permettant de notifier de la progression de l'exécution
- W: le type du résultat de l'exécution

A partir de l'implémentation d'[AT], nous proposons une implémentation utilisant 3 paramètres: l'url de l'image à charger, un entier représentant les étapes d'avancement de notre tâche et la classe Bitmap renvoyant l'image quand celle-ci est chargée, soit la classe **AsyncTask<String, Integer, Bitmap>**.

La classe BitmapDownloaderTask

Notre classe héritant de **AsyncTask** est donc dédiée à la gestion d'une tâche qui va impacter l'interface graphique. Dans notre exemple, la classe doit charger une image, et donc modifier un **ImageView**. Elle doit aussi gérer l'affichage de la progression de la tâche, ce que nous proposons de faire dans un **TextView**. Le constructeur de la classe doit donc permettre d'accéder à ces deux éléments graphiques:

```
public class BitmapDownloaderTask extends AsyncTask<String, Void, Bitmap> {

private final WeakReference<ImageView> imageViewReference;
private final WeakReference<TextView> textViewReference;

public BitmapDownloaderTask(ImageView imageView, TextView textView) {
  imageViewReference = new WeakReference<ImageView>(imageView);
  textViewReference = new WeakReference<TextView>(textView);
}
```

On notera l'emploi de *weak references* qui permet au *garbage collector* de détruire les objets graphiques même si le téléchargement de l'image est encore en cours.

Méthodes de BitmapDownloaderTask

void onPreExecute(): invoquée juste après que la tâche soit démarrée. Elle permet de modifier l'interface graphique juste après le démarrage de la tâche. Cela permet de créer une barre de progression ou de charger un élément par défaut. Cette méthode sera exécutée par l'*UI thread*.

```
protected void onPreExecute() {
if (imageViewReference != null) {
  ImageView imageView = imageViewReference.get();
  if (imageView != null) {
    imageView.setImageResource(R.drawable.interro);
}}}
```

W doInBackground(U...): invoquée dans le thread qui s'exécute en tâche de fond, une fois que **onPreExecute()** est terminée. C'est cette méthode qui prendra un certain temps à s'exécuter. L'objet reçu en paramètre est de type **U** et permet de gérer la tâche à accomplir. Dans notre exemple, l'objet est l'url où télécharger l'image. A la fin, la tâche doit renvoyer un objet de type **W**.

```
protected Bitmap doInBackground(String... params) {
  String url = params[0];
  publishProgress(new Integer(0));
  AndroidHttpClient client = AndroidHttpClient.newInstance("Android");
  HttpGet getRequest = new HttpGet(url);
  HttpResponse response = client.execute(getRequest);
  publishProgress(new Integer(1));
  ...  }
```

Méthodes de BitmapDownloaderTask

void onProgressUpdate(V...): invoquée dans le thread qui gère l'UI, juste après que la méthode **doInBackground(U...)** ait appelé **publishProgress(V...)**. On reçoit donc l'objet **V** qui contient les informations permettant de mettre à jour l'UI pour notifier de la progression de la tâche.

```
protected void onProgressUpdate(Integer... values) {
  Integer step = values[0];
  if (textViewReference != null) {
  textViewReference.get().setText("Step: " + step.toString());
}}
```

onPostExecute(W): invoquée quand la tâche est terminée et qu'il faut mettre à jour l'UI avec le résultat calculé dans l'objet de la classe **W**.

```
protected void onPostExecute(Bitmap bitmap) {
  if (imageViewReference != null) {
    ImageView imageView = imageViewReference.get();
    if (imageView != null) {
      imageView.setImageBitmap(bitmap);
}}}
```

Démonstation

(cf ANX_Tâches-asynchrones) La démonstration suivante montre la notification au travers de l'entier passant de 0 à 3. Une image par défaut est chargée avant que l'image soit complètement téléchargée depuis le web et affichée à son tour.

Video

IntentService

[II] Un autre *design pattern* a été simplifié dans Android: l'exécution dans un processus isolé de *jobs* provenant d'une autre partie de l'application. Les *jobs* sont provisionnés au travers d'un **Intent** et l'**IntentService** exécute la tâche dans la méthode **onHandleIntent()**, indépendamment de l'*UI thread*.

```
public class MonServiceLourd extends IntentService {
{
  public MonServiceLourd() {
    super("MonServiceLourd");
  }
  @Override
  protected void onHandleIntent(Intent intent) {
    ... // Tâche longue, par exemple un téléchargement
  }
}
```

Le bénéfice, qui n'est pas visible ici, est que les **Intent** sont mis dans une file d'attente et traités les uns après les autres, évitant une surcharge du système. Chaque **Intent** est traité par **onHandleIntent()** et le service s'arrête tout seul quand il n'y a plus rien à traiter.

Loaders

Encore un autre *design pattern* est simplifié: le chargement au travers de **Loaders** et du **LoaderManager**.

La classe **Loader** permet de gérer le chargement en tâche de fond, si l'on utilise un **AsyncTaskLoader<D>**. Combinée à un **Cursor** cela permet de gérer indépendemment une interrogation de base de donnée au travers d'un **AsyncTaskLoader<Cursor>**. Une classe existe déjà pour ce cas: **CursorLoader**. Un **Loaders** peut-être créé *from scratch* mais il faut implémenter de nombreuses méthodes, et notamment celles qui gèrent le cas ou les données surveillées changent.

La classe **LoaderManager** permet d'instancier un **Loader** (il faut surcharger des *callbacks* pour cela), qui va gérer les chargement. L'intérêt du **LoaderManager** est de conserver les données même si l'activité est rechargée par exemple à cause d'un changement de configuration. Cela économise le rechargement de ces données.

Le bénéfice attendu est le déchargement du thread principal et d'éviter tout ralentissement lorsque les données changent beaucoup ou sont volumineuses à traiter.

L'implémentation d'un *Loader* est assez technique. On peut se référer à [LLB] pour une explication assez complète.

6.6 Bilan: processus et threads

Par défaut, un service s'exécute donc dans le même processus que l'application, dans l'*UI thread*. Dans ce cas, le service doit programmer des tâches concurrentes qui s'exécuteront alors dans des threads indépendants. Ces threads survivront à l'activité principale car ils sont lancés depuis la classe héritant de **Service**.

On peut toutefois vouloir absolument caser un service dans un processus indépendant. Dans ce cas, le service et l'application sont séparés. La difficulté de la séparation du service et de l'application réside dans la difficulté de l'interaction de ces deux composants. Il faut alors utiliser les mécanismes prévus dans Android pour faire coopérer ces deux entités à l'aide de l'objet **IBinder** ou d'IPC AIDL.

Je ne résiste pas au plaisir de vous citer une discussion du googlegroup android-developer [LSAS], à propos des services persistants, ressemblant à des démons systèmes:

R. Ravichandran> *I have a need to create a background service that starts up during the system boot up, and keeps running until the device is powered down. There is no UI or Activity associated with this.*

Dianne Hackborn> *Mark answered how to do this, but please: think again about whether you really need to do this. Then think another time. And think once more. And if you are really really absolutely positively sure, this what you want to do, fine, but realize --*

On current Android devices, we can keep only a small handful of applications running at the same time. Having your application do this is going to going to take resources from other things that at any particular point in time would be better used elsewhere. And in fact, you can be guaranteed that your service will -not- stay running all of the time, because there is so much other stuff that wants to run (other background services that are only running when needed will be prioritized over yours), or needs more memory for what the user is actually doing (running the web browser on complicated web pages is a great way to kick background stuff out of memory).

We have lots of facilities for implementing applications so they don't need to do this, such as alarms, and various broadcasts from events going on in the system. Please please please use them if at all possible. Having a service run forever is pretty close to the side of evil.

6.7 Coopération service/activité

Un service est souvent paramétré par une activité. L'activité peut être en charge de (re)démarrer le service, d'afficher les résultats du service, etc. Trois possibilités sont offertes pour faire coopérer service et activité *frontend*:

- utiliser des *Intents* pour envoyer des informations
- lier service et activité d'un même processus
- définir une interface de service en utilisant le langage AIDL

Lier le service et l'activité peut se faire assez simplement en définissant une interface (le contrat de communication entre le service et l'activité) et en utilisant la classe **Binder**. Cependant, cela n'est réalisable que si le service et l'activité font partie du même processus.

Pour deux processus séparés, il faut passer par la définition de l'interface en utilisant le langage AIDL. La définition de l'interface permettra à l'outil aidl de générer les stubs de communication inter processus permettant de lier une activité et un service.

Lier activité et service: IBinder

[BAS] Pour associer un service à une activité, le service doit implémenter la méthode **onBind()** qui renverra à l'application un objet de type **IBinder**. Au travers de l'objet **IBinder**, l'application aura accès au service.

On implémente donc dans le service:

```
private final IBinder ib = new MonServiceBinder();
public IBinder onBind(Intent intent) {
  return ib;
}
```

et on crée une nouvelle classe **MyServiceBinder**, de préférence classe interne à l'activité, qui renvoie la classe associée au service:

```
private int infoOfService = 0; // La donnée à transmettre à l'activité
private class MonServiceBinder extends Binder implements AndroServiceInterface {
     // Cette classe qui hérite de Binder implémente une méthode
     // définie dans l'interface AndroServiceInterface
```

```
    public int getInfo() {
        return infoOfService;
}}
```

L'interface, à définir, déclare les méthodes qui seront accessibles à l'application:

```
public interface AndroServiceInterface {
  public int getInfo();
}
```

Lier activité et service: ServiceConnexion

Pour lier l'activité au service, un objet de la classe **ServiceConnexion** doit être instancié à partir de l'application. C'est cet objet qui fera le lien entre l'activité et le service. Deux méthodes de cet objet doivent être surchargées:

- **onServiceConnected** qui sera appelé lorsque le service sera connecté à l'application et qui permettra de récupérer un pointeur sur le binder.
- **onServiceDisconnected** qui permet de nettoyer les choses créées à la connexion, par exemple un pointeur d'attribut de classe sur l'objet **Binder**.

```
private int infoFromService = 0;
private ServiceConnection maConnexion = new ServiceConnection() {
    public void onServiceConnected(ComponentName name, IBinder service) {
        AndroServiceInterface myBinder = (AndroServiceInterface)service;
        infoFromService = myBinder.getInfo();
    }
    public void onServiceDisconnected(ComponentName name) { }
};
```

Puis, dans le code de l'activité, on initie la connexion:

```
Intent intentAssociation =
       new Intent(AndroServiceBindActivity.this, AndroService.class);
bindService(intentAssociation, maConnexion, Context.BIND_AUTO_CREATE);
Toast.makeText(getApplicationContext(), "Info lue dans le service: "
                    + infoFromService, Toast.LENGTH_SHORT).show();
unbindService(maConnexion);
```

Démonstration

Dans cette démonstration, un service lance un thread qui attendra 4 secondes avant d'enregistrer la valeur 12 comme résultat du service. A chaque click du bouton, l'activité se connecte au service pour *toaster* le résultat. (cf ANX_Binding-entre-service-et-activite-pour-un-meme-processus)

Video

Service au travers d'IPC AIDL

Le langage AIDL est très proche de la définition d'interfaces Java EE (mais pas tout à fait identique). Il s'agit de définir ce que les service est capable de faire et donc, quelles méthodes peuvent être appelées.

Un peu comme un web service, la définition de l'interface permettra à l'outil AIDL de générer des stubs de communication pour les IPC. Ainsi, une autre application android pourra appeler le service au travers d'IPC.

Une interface en AIDL peut par exemple ressembler à:

```
package andro.jf;

interface AndroServiceInterface {
  int getInfo();
}
```

L'outil aidl va générer les *stubs* correspondants à l'interface dans un fichier .java portant le même nom que le fichier aidl, dans le répertoire *gen*.

```
// This file is auto-generated.  DO NOT MODIFY.
package andro.jf;
public interface AndroServiceInterface extends android.os.IInterface {
public static abstract class Stub extends android.os.Binder
  implements andro.jf.AndroServiceInterface {
private static final java.lang.String DESCRIPTOR =
  "andro.jf.AndroServiceInterface";
...
```

Service exposant l'interface

Dans le service, on utilise alors l'implémentation générée par aidl pour implémenter ce que fait le service dans la méthode déclarée par l'interface:

```
private int infoOfService = 0; // La donnée à transmettre à l'activité
private AndroServiceInterface.Stub ib = new AndroServiceInterface.Stub() {
    @Override
    public int getInfo() throws RemoteException {
        return infoOfService;
    }
};
public IBinder onBind(Intent arg0) {
    return ib; }
```

Du côté de l'application, il faut aussi utiliser le code de la *stub* pour récupérer l'objet implémentant l'interface (cf ANX_Binding-inter-processus-utilisant-le-langage-AIDL):

```
private ServiceConnection maConnexion = new ServiceConnection() {
  public void onServiceConnected(ComponentName name, IBinder service) {
    AndroServiceInterface myBinder =
        AndroServiceInterface.Stub.asInterface(service);
    try { // stockage de l'information provevant du service
      infoFromService = myBinder.getInfo();
    }
```

Il devient alors possible de déclarer le service comme appartenant à un autre processus que celui de l'activité, i.e. **android:process=":p2"** (cf. Processus).

6.8 Etude de cas

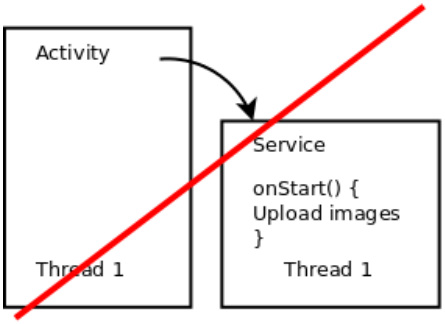
Cette section présente une étude de cas sur les solutions possible au problème suivant: on souhaite réaliser l'*upload* d'une image volumineuse en tâche de fond d'une activité qui peut éventuellement être fermée. Ce type de mécanisme est couramment employé par les applications qui chargent des images dans le *cloud*. Le but n'est pas d'étudier comment réaliser l'upload mais plutôt *où* le réaliser dans le code.

Les caractéristiques visées sont les suivantes:

- depuis l'application, on souhaite choisir une image et lancer l'upload
- l'upload doit se poursuivre si on ferme l'application
- si on lance plusieurs upload, l'idéal serait qu'ils se fassent les uns derrière les autres
- une notification sur l'avancement du chargement dans l'application principale serait un plus

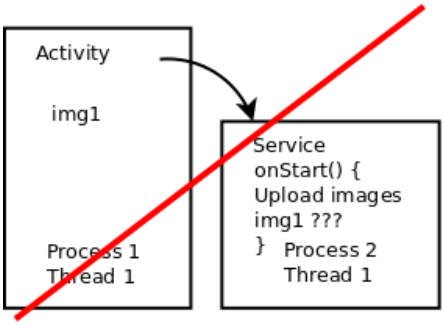
L'utilisation d'un service et d'une tâche programmée semble tout indiqué...

Service opérant dans onStart()



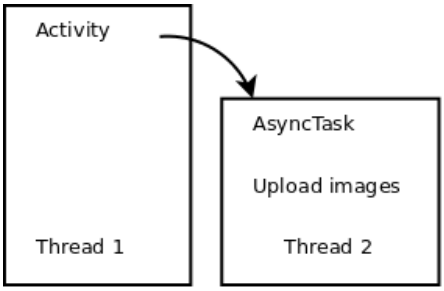
Cette solution n'est pas bonne: elle bloque l'activité principale, c'est-à-dire l'*UI thread* et donc l'interface utilisateur !

Service opérant dans onStart() + processus



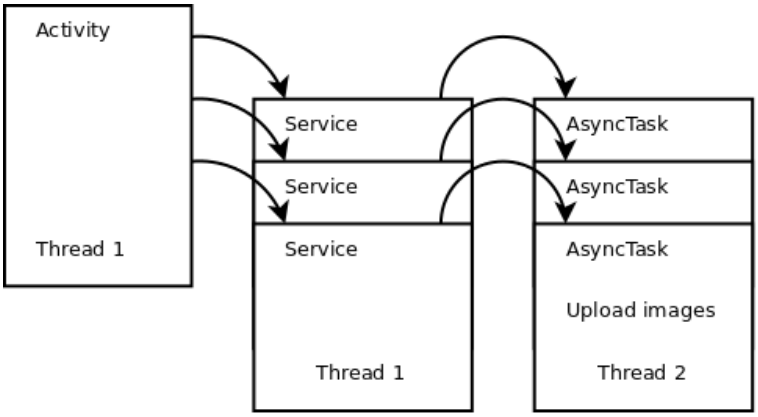
Cette solution est un peu mieux: l'interface de l'activité n'est plus bloquée. Cependant, en isolant le processus du service de celui de l'activité, il sera difficile de récupérer l'image.

Service opérant dans une tâche asynchrone



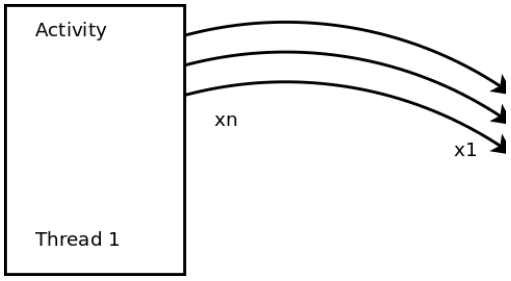
Dans cette solution, la tâche asynchrone aura tout loisir de charger l'image et de notifier l'activité principale. Cependant, si l'utilisateur quitte l'activité, l'*upload* s'arrête.

Service opérant dans onStart() + processus



En introduisant un service entre l'activité et la tâche synchronisée on résout le problème précédemment évoqué: la tâche survivra à l'activité. Dans une telle configuration, à chaque *upload* sera généré un nouveau service qui vivra le temps de la tâche.

Service opérant dans onStart() + processus



La manière la plus élégante d'opérer est d'utiliser un **IntentService**. L'**IntentService** va gérer toutes les requêtes d'*upload* dans une *working queue* qui sera dans un *thread* à part inclu dans un service.

7 Connectivité

7.1 Téléphonie

Les fonctions de téléphonie sont relativement simples à utiliser. Elles permettent de récupérer l'état de la fonction de téléphonie (appel en cours, appel entrant, ...), d'être notifié lors d'un changement d'état, de passer des appels et de gérer l'envoi et réception de SMS.

L'état de la téléphonie est géré par la classe **TelephonyManager** qui permet de récupérer le nom de l'opérateur, du téléphone, et l'état du téléphone. Pour lire ces informations, il est nécessaire de disposer de la permission **android.permission.CALL_PHONE**.

```
TelephonyManager tel =
  (TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
int etat = tel.getCallState();
if (etat == TelephonyManager.CALL_STATE_IDLE)
   // RAS
if (etat == TelephonyManager.CALL_STATE_RINGING)
   // Le téléphone sonne
String SIMnb = tel.getSimSerialNumber();
```

Il est aussi possible d'être notifié d'un changement d'état en utilisant un écouteur:

```
public class Ecouteur extends PhoneStateListener {
  public void onCallStateChanged(int etat, String numero) {
```

```
    super.onCallStateChanaged(etat, numero)
      if (etat == TelephonyManager.CALL_STATE_OFFHOOK)
      // Le téléphone est utilisé
}
```

Passer ou déclarer savoir passer un appel

Il est bien sûr possible de passer un appel ou de déléguer l'appel, ces deux actions étant réalisées avec un *Intent* (attention aux permissions):

```
Uri telnumber = Uri.parse("tel:0248484000");
Intent call = new Intent(Intent.ACTION_CALL, telnumber);
startActivity(call);
```

```
Uri telnumber = Uri.parse("tel:0248484000");
Intent call = new Intent(Intent.ACTION_DIAL, telnumber);
startActivity(call);
```

Les applications qui peuvent passer des appels doivent filtrer ce type d'*Intent* pour pouvoir être invoquée lorsque l'*Intent* est lancé:

```
<receiver android:name=".ClasseGerantLAppel">
  <intent-filter>
    <action android:name="Intent.ACTION_CALL"/>
  </intent-filter>
</receiver>
```

Envoyer et recevoir des SMS

Si la permission **android.permission.SEND_SMS** est disponible, il est possible d'envoyer des SMS au travers de **SmsManager**:

```
SmsManager manager = SmsManager.getDeault();
manager.sendTextMessage("02484840000", null, "Coucou !", null, null);
```

Inversement, il est possible de créer un filtre d'*Intent* pour recevoir un SMS qui sera géré par un *broadcast receiver*. L'action à préciser dans le filtre d'Intent du receveur est **android.provider.Telephony.SMS_RECEIVED**:

```
<receiver android:name=".SMSBroadcastReceiver">
  <intent-filter>
    <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
  </intent-filter>
</receiver>
```

Puis, le début du code du *broadcast receiver* est par exemple:

```
public final class MyBroadcastReceiver extends BroadcastReceiver {
  public void onReceive(Context context, Intent intent) {
```

```
    if (intent.getAction().equals(android.provider.Telephony.SMS_RECEIVED)) {
      String val = extra.getString("extra");
      Object[] pdus = (Object[]) intent.getExtras().get("pdus");
      SmsMessage[] messages = new SmsMessage[pdus.length];
      for (int i=0; i < pdus.length; i++)
        messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
}}}
```

7.2 Réseau

(cf ANX_Réseau) Le réseau peut être disponible ou indisponible, suivant que le téléphone utilise une connexion Wifi, 3G, bluetooth, etc. Si la permission **android.permission;ACCESS_NETWORK_STATE** est déclarée, la classe **NetworkInfo** (depuis **ConnectivityManager**) permet de lire l'état de la connexion réseau parmi les constantes de la classe **State**: CONNECTING, CONNECTED, DISCONNECTING, DISCONNECTED, SUSPENDED, UNKNOWN.

```
ConnectivityManager manager =
  (ConnectivityManager)getSystemService(CONNECTIVITY_SERVICE);
NetworkInfo net = manager.getActiveNetworkInfo();
if (net.getState().compareTo(State.CONNECTED)
  // Connecté
```

Il est possible de connaître le type de la connexion:

```
int type = net.getType();
```

Le type est un entier correspondant, pour l'instant, au wifi ou à une connexion de type mobile (GPRS, 3G, ...).

- **ConnectivityManager.TYPE_MOBILE**: connexion mobile
- **ConnectivityManager.TYPE_WIFI**: wifi

Gérer le réseau Wifi/Mobile

Le basculement entre les types de connexion est possible si la permission **WRITE_SECURE_SETTINGS** est disponible. On utilise alors la méthode **setNetworkPreference** sur l'objet **ConnectivityManager** pour lui donner l'entier correspondant au type de connexion voulu. Par exemple:

```
manager.setNetworkPreference(ConnectivityManager.TYPE_WIFI);
```

L'accès au réseau wifi est gérable depuis une application, ce qui permet d'allumer ou de couper le wifi. L'objet **WifiManager** permet de réaliser cela.

```
WifiManager wifi = (WifiManager)getSystemService(Context.WIFI_SERVICE);
if (!wifi.isWifiEnabled())
  wifi.setWifiEnabled(true);
```

Les caractéristiques de la connexion Wifi sont accessibles par des appels statiques à des méthodes de **WifiManager**:

- force du signal projeté sur une échelle [0,levels]: **WifiManager.calculateSignalLelvel(RSSI ?, levels)**
- vitesse du lien réseau: **info.getLinkSpeed()**
- les points d'accès disponibles: **List<ScanResult> pa = manager.getScanResults()**

7.3 Bluetooth

Le bluetooth se gère au travers de principalement 3 classes:

- **BluetoothAdapter**: similaire au **WifiManager**, cette classe permet de gérer les autres appareils bluetooth et d'initier les communications avec ceux-ci.
- **BluetoothDevice**: objet représentant l'appareil distant.
- **BluetoothSocket** et **BluetoothServerSocket**: gère une connexion établie.

Pour pouvoir utiliser les fonctionnalités bluetooth, il faut activer les permissions **android.permission.BLUETOOTH** et **android.permission.BLUETOOTH_ADMIN** pour pouvoir chercher des appareils ou changer la configuration bluetooth du téléphone.

```
BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();
if (!bluetooth.isEnabled())
{
  Intent launchBluetooth = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
  startActivity(launchBluetooth);
}
```

S'associer en bluetooth

Pour pouvoir associer deux appareils en bluetooth, il faut que l'un d'eux soit accessible (s'annonce) aux autres appareils. Pour cela, l'utilisateur doit autoriser le mode "découverte". L'application doit donc le demander explicitement via un **Intent**:

```
Intent discoveryMode = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discoveryMode.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 60);
startActivity(discoveryMode);
```

A l'inverse, si un appareil externe diffuse une annonce de découverte, il faut capturer les intents recus en broadcast dans le mobile:

```
public final class BluetoothBroadcastReceiver extends BroadcastReceiver {
  public void onReceive(Context context, Intent intent) {
    String action = intent.getAction();
    BluetoothDevice appareil = null;
      if (action.equals(BluetoothDevice.ACTION_FOUND))
      appareil = (BluetoothDevice)intent.getParcelableExtra(
                 BluetoothDevice.EXTRA_DEVICE);
  }
}
```

Enfin les appareils associés se parcourent à l'aide d'un **Set<BluetoothDevice>**:

```
Set<BluetoothDevice> s = bluetooth.getBondedDevices();
  for (BluetoothDevice  ap : s)
      Toast.makeText(getApplicationContext(), "" + ap.getName(),
      Toast.LENGTH_LONG).show();
```

Utiliser le réseau

De nombreuses méthodes de développement permettent d'exploiter le réseau. Elles ne sont pas rappelées en détail ici (ce n'est pas l'objet du cours) et sont de toutes façons déjà connues:

- HTTP: **HttpClient**, **httpResponse**
- SOAP: **SoapObjet**, **SoapSerializationEnvelope**
- REST:
  - JSON: **JSONObject**
  - XML: **DocumentBuilder**
- Sockets: **Socket**, **ServerSocket**
- Bluetooth: **BluetoothSocket**, **BluetoothServerSocket**

7.4 Localisation

(cf ANX_Localisation) Comme pour le réseau, Android permet d'utiliser plusieurs moyens de localisation. Cela permet de rendre transparent l'utilisation du GPS, des antennes GSM ou des accès au Wifi. La classe **LocationManger** permet de gérer ces différents fournisseurs de position.

- **LocationManager.GPS_PROVIDER**: fournisseur GPS
- **LocationManager.NETWORK_PROVIDER**: fournisseur basé réseau

La liste de tous les fournisseurs s'obtient au travers de la méthode **getAllProviders()** ou **getAllProviders(true)** pour les fournisseurs activés:

```
LocationManager manager =
  (LocationManager)getSystemService(Context.LOCATION_SERVICE);
List<String> fournisseurs = manager.getAllProviders();
for (String f : fournisseurs) {
  Toast.makeText(getApplicationContext(), "" + f, Toast.LENGTH_SHORT).show();
  if (f.equals.(LocationManager.GPS_PROVIDER))
      ...
}
```

Les permissions associées pour la localisation sont:

- **android.permission.ACCESS_FINE_LOCATION** via le GPS
- **android.permission.ACCESS_COARSE_LOCATION** via le réseau

Coordonnées

A partir du nom d'un fournisseur de position actif, il est possible d'interroger la dernière localisation en utilisant l'objet **Location**.

```
Location localisation = manager.getLastKnownLocation("gps");
Toast.makeText(getApplicationContext(), "Latitude" +
  localisation.getLatitude(), Toast.LENGTH_SHORT).show();
Toast.makeText(getApplicationContext(), "Longitude" +
  localisation.getLongitude(), Toast.LENGTH_SHORT).show();
```

Il est possible de réagir à un changement de position en créant un écouteur qui sera appelé à intervalles réguliers et pour une distante minimum donnée:

```
manager.requestLocationUpdates("gps", 6000, 100, new LocationListener() {

  public void onStatusChanged(String provider, int status, Bundle extras) {
  }
  public void onProviderEnabled(String provider) {
  }
  public void onProviderDisabled(String provider) {
  }
  public void onLocationChanged(Location location) {
    // TODO Auto-generated method stub
  }
});
```

Alerte de proximité

Il est possible de préparer un événement en vu de réagir à la proximité du téléphone à une zone. Pour cela, il faut utiliser la méthode **addProximityAlert** de **LocationManager** qui permet d'enregistrer un *Intent* qui sera envoyé lorsque des conditions de localisation sont réunies. Cette alerte possède une durée d'expiration qui la désactive automatiquement. La signature de cette méthode est:

addProximityAlert(double latitude, double longitude, float radius, long expiration, PendingIntent intent)

Il faut ensuite filtrer l'intent préparé:

```
IntentFilter filtre = new IntentFilter(PROXIMITY_ALERT);
registerReceiver(new MyProximityAlertReceiver(), filtre);
```

La classe gérant l'alerte est alors:

```
public class MyProximityAlertReceiver extends BroadcastReiceiver {
  public void onReceive(Context context, Intent intent) {
    String key = LocationManager.KEY_PROXIMITY_ENTERING;
    Boolean entering = intent.getBooleanExtra(key, false);
}}
```

Carte google map

Google fournit une librairie qui permet d'inclure une carte google map dans son application. Voici quelques éléments pour utiliser la version 2 de cette API.

Le manifest doit tout d'abord déclarer la librairie et plusieurs permissions, dont notamment celle de l'API. Il faut donc ajouter au tag application un tag signifant l'utilisation des google play services (à intégrer comme un projet Eclipse et à déclarer comme une dépendance) ainsi que votre clef d'accès à l'API:

```
<meta-data android:name="com.google.android.gms.version"
 android:value="@integer/google_play_services_version" />
<meta-data android:name="com.google.android.geo.API_KEY"
 android:value="AIzaS....."/>
```

La clef d'API est une clef à générer sur le serveur de google gérant le service des cartes à partir de votre clef de signature d'application. Votre clef de signature de travail générée par Eclipse suffit pour la phase de développement.

Ensuite, il faut un certain nombre de permissions, et déclarer une permission dépendante du nom de package et que l'on utilise soi même:

```
<permission
      android:name="jf.andro.mapv2.permission.MAPS_RECEIVE"
      android:protectionLevel="signature" />
<uses-permission android:name="jf.andro.mapv2.permission.MAPS_RECEIVE"/>

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="com.google.android.providers.gsf.permission.
   READ_GSERVICES"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
```

On peut alors utiliser un fragment, déclarant une carte:

```
    <fragment
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:map="http://schemas.android.com/apk/res-auto"
  android:name="com.google.android.gms.maps.MapFragment"
  android:id="@+id/map"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  map:mapType="normal" />
```

Reverse Geocoding

Une autre classe intéressante fournie par Android permet de retrouver une adresse à partir d'une localisation. Il s'agit de la classe **Geocoder** qui permet d'interroger un service Google à partir de coordonnées. Le code est assez simple à mettre en oeuvre:

```
Geocoder geocoder = new Geocoder(context, Locale.getDefault());
List<Address> addresses = geocoder.getFromLocation(loc.getLatitude(),
                                                   loc.getLongitude(), 1);
String addressText = String.format("%s, %s, %s",
  address.getMaxAddressLineIndex() > 0 ? address.getAddressLine(0) : "",
  address.getLocality(),
  address.getCountryName());
```

Attention à utiliser l'émulateur contenant les "Google APIs" pour pouvoir utiliser ce service. Pour savoir si l'OS dispose du *backend* permettant d'utiliser la méthode **getFromLocation**, on peut appeler la méthode **isPresent()** qui doit renvoyer *true* dans ce cas. Un exemple de code utilisant une tâche asynchrone est donné dans ANX_Geocoding.

Video

7.5 Capteurs

(cf ANX_Capteurs) Android introduit la gestion de multiples capteurs. Il peut s'agir de l'accéléromètre, du gyroscope (position angulaire), de la luminosité ambiante, des champs magnétiques, de la pression ou température, etc. En fonction, des capteurs présents, la classe **SensorManager** permet d'accéder aux capteurs disponibles. Par exemple, pour l'accéléromètre:

```
SensorManager manager = (SensorManager)getSystemService(SENSOR_SERVICE);
manager.registerListener( new SensorEventListener() {
    public void onSensorChanged(SensorEvent event) { // TODO method stub
    }
    public void onAccuracyChanged(Sensor sensor, int accuracy) { // TODO
  }}
  , manager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
  , SensorManager.SENSOR_DELAY_UI );
}
```

Quand l'accéléromètre n'est plus utilisé, il doit être désenregistré à l'aide de:

```
manager.unregisterListener( pointeur ecouteur ,
            manager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) );
```

La fréquence d'interrogation influe directement sur l'énergie consommée. Cela va du plus rapide possible pour le système à une fréquence faible: SENSOR_DELAY_FASTEST < SENSOR_DELAY_GAME < SENSOR_DELAY_NORMAL < SENSOR_DELAY_UI.

Hardware

Pour déclarer l'utilisateur d'un capteur et avoir le droit d'accéder aux valeurs lues par le *hardware*, il ne faut pas oublier d'ajouter dans le Manifest la déclaration adéquat à l'aide du tag *uses-feature* [UF]. Cela permet à Google Play de filtrer les applications compatibles avec l'appareil de l'utilisateur. Par exemple, pour l'accéléromètre:

```
<uses-feature android:name="android.hardware.sensor.accelerometer"></uses-feature>
```

La directive n'est cependant pas utilisée lors de l'installation, ce qui signifie qu'il est possible d'installer une application utilisant le bluetooth sans posséder de *hardware* bluetooth. Evidemment, il risque d'y avoir une exception ou des dysfonctionnements. Un booléen supplémentaire permet alors d'indiquer si ce *hardware* est indispensable au fonctionnement de l'application:

```
<uses-feature android:name="xxx" android:required="true"></uses-feature>
```

L'outil aapt permet d'apprécier la façon dont Google Play va filtrer l'application, en donnant un dump des informations collectées:

```
./aapt dump badging ~/workspace/AndroSensors2/bin/AndroSensors2.apk

package: name='jf.andro.as' versionCode='1' versionName='1.0'
sdkVersion:'8'
targetSdkVersion:'14'
uses-feature:'android.hardware.sensor.acceleromedter'
```

Lecture des données

Les valeurs sont récupérées au travers de la classe **SensorEvent**. Toutes les données sont dans un tableau, dont la taille dépend du type de capteur utilisé. Par exemple pour l'accéléromètre:

```
if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
{
  float x,y,z;
  x = event.values[0];
  y = event.values[1];
  z = event.values[2];
}
```

Le code est similaire pour le capteur d'orientation:

```
float azimuth, pitch, roll;
azimuth = event.values[0];
pitch = event.values[1];
roll = event.values[2];
}
```

Pour le capteur de lumière, on se limite à une seule valeur:

```
float lumiere;
lumiere = event.values[0];
```

7.6 Caméra

La documentation proposée par Google dans [CAM] permet de bien comprendre comment fonctionne la caméra. Il est souvent nécessaire de construire une zone de *preview* de ce que la caméra voit, avant par exemple de prendre une photo. Avant toute chose, il faut prévoir dans le *manifest* la permission nécessaire i.e. **android.permission.CAMERA**. Ensuite les grandes étapes sont les suivantes:

Dans l'activité principale on ajoute une **ViewGroup** spéciale et on ouvre la caméra (arrière, d'*id* 0, frontale, d'*id* 1):

```
Camera mCamera;
Preview mPreview = new Preview(this);
setContentView(mPreview);
Camera.open(id);
mPreview.setCamera(mPreview);
```

Avec une classe **Preview** héritant de **ViewGroup** et implémentant **SurfaceHolder.Callback**. Cela oblige à recoder les méthodes **surfaceChanged**, **surfaceCreated**, **surfaceDestroyed** et **onLayout** qui seront appelées quand la surface de rendu est effectivement créée. Notamment, il faut accrocher un objet **SurfaceHolder** à l'objet **Camera**, puis dire à l'objet **SurfaceHolder** que le **callback** est l'objet **Preview**. Plus de détails sont observables en annexes [ANX_Camera].

8 Développement client serveur

8.1 Architectures

Les applications clientes interragissant avec un serveur distant, qu'il soit dans un *cloud* ou non, peuvent permettre de déporter des traitements sur le serveur. La plupart du temps, on déporte les traitements pour:

- garder et protéger le savoir métier
- rendre l'utilisateur captif
- améliorer les performances lors de traitements lourds
- économiser les ressources du client

Le prix à payer est le temps de latence induit par le réseau, voire l'incapacité de l'application à fonctionner correctement si l'utilisateur n'a plus de réseau ou un réseau dégradé (e.g. 2G). L'expérience utilisateur peut aussi être largement impacté.

Losqu'un développeur réalise une application cliente, il utilise presque de manière inconsciente le modèle MVC. S'il ajoute la dimension réseau, il peut alors choisir de déporter des élements du modèle MVC sur le serveur. En effet, il faut décider quels éléments du modèle MVC sont du côté client ou du côté serveur.

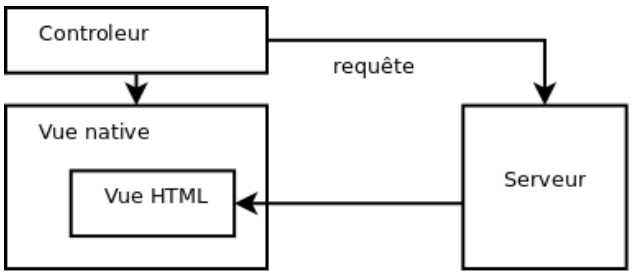
Types d'applications

Lorsqu'on réalise une application client-serveur, on peut considérer que trois choix d'architectures sont possibles pour l'emplacement du modèle MVC:

- application native: MVC sont codés en java, dans une machine virtuelle Dalvik
- application hybride: certains éléments du Contrôle sont codés en java, mais Modèles et Vues sont codés par le serveur
- application web: tout est réalisé sur le serveur

Dans ce cours, jusqu'à maintenant, il a toujours été présenté des applications natives. Une application hybride utilise le langage HTML5 et les capacités d'Android à afficher des pages web pour intégrer des composants de visualisation dans une application native. Ainsi, une partie de l'application est

classiquement native, tandis que certaines parties de la vue sont des affichages de documents HTML calculés coté serveur.



8.2 Applications Natives

Même si elle native, une application peut tout à fait dépendre d'un serveur distant. L'interface graphique utilise des composants Android, mais la localisation et le traitement des données est partagé entre le téléphone et le serveur.

On peut décrire plusieurs cas différents:

- Native offline: tout est sur le client: interface graphique, données, traitements.
- Native + Cloud/API online ou partiellement offline: interface graphique sur le client mais données dans le cloud ou via un accès à une API avec des traitements possibles des deux côtés. Le mode partiellement offline nécessite de faire persister les données sur le téléphone, au moins partiellement.
- Native + Cloud/API online: interface graphique sur le client mais données à l'extérieur avec des traitements possibles des deux côtés.

8.3 Applications Hybrides

Pour réaliser une application hybride, il suffit d'utiliser un objet de type **WebView** qui, comme son nom l'indique, est un object graphique fait pour visualiser une page web. Après l'avoir intégré dans le gabarit, il faut lui faire charger une page:

```
WebView wv = (WebView)findViewById(R.id.webView1);
wv.getSettings().setJavaScriptEnabled(true);
wv.loadUrl("http://www.univ-orleans.fr/lifo/Members/Jean-Francois.Lalande/");
```

Cependant, les liens web appelent tout de même le navigateur, ce qui est assez génant:

Video

Surcharge du WebClient

Afin de ne pas lancer le navigateur par défaut, il faut surcharger le client web afin de recoder la méthode agissant lorsqu'un lien est cliqué. La classe est très simple:

```
private class MyWebViewClient extends WebViewClient {
  @Override
  public boolean shouldOverrideUrlLoading(WebView view, String url) {
    view.loadUrl(url);
    Toast.makeText(getApplicationContext(), "A link has been clicked! ",
                   Toast.LENGTH_SHORT).show();
```

```
    return true;
  }}
```

Du côté de l'activité, il faut remplacer le comportement du client web par défaut. On peut aussi prévoir un bouton afin de pouvoir revenir en arrière car le bouton *back* du téléphone va tuer l'activité.

```
wv.setWebViewClient(new MyWebViewClient());
Button back = (Button)findViewById(R.id.back);
back.setOnClickListener(new OnClickListener() {
  public void onClick(View v) {
    wv.goBack();
    }
});
```

Démonstration

Dans cet exemple (cf ANX_Applications-hybrides), on navigue sur une page web incluse dans une activité. Un bouton transparent "Back" permet de controler la **WebView**:

Video

JQueryMobile

Evidemment, une page web classique n'est pas très adaptée à un téléphone ou une tablette. Il existe des technologies pour calculer le rendu javascript d'une page web en fonction du navigateur, ce qui est particulièrement pratique. On peut par exemple utiliser JQueryMobile pour cela, comme expliqué dans [JQM].

```
<!DOCTYPE html>
<html>
<head>
<title>My Page</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.2.0/
 jquery.mobile-1.2.0.min.css" />
<script src="http://code.jquery.com/jquery-1.8.2.min.js"></script>
<script src="http://code.jquery.com/mobile/1.2.0/jquery.mobile-1.2.0.min.js"/>
</head>
<body>

<div data-role="page">

<div data-role="header">
<h1>Jquery example</h1>
</div>

<div data-role="content">
<p>Hello world !</p>
</div><!-- /content -->

</div> </body> </html>
```

JQueryMobile: exemple de liste

Par exemple, une liste d'items se transforme facilement en éléments facilement cliquable dans un environnement tactile:

```
<ul data-role="listview" data-inset="true" data-filter="true">
 <li><a href="http://www.univ-orleans.fr/lifo/Members/Jean-Francois.Lalande">
  Home</a></li>
 <li><a href="publications.html">Publications</a></li>
 <li><a href="teaching.html">Enseignement</a></li>
 <li><a data-role="button" data-icon="star" data-theme="b"
         href="jquerymobiletest2.html">page 2</a></li>
</ul>
```

Video

JQueryMobile: intégration hybride

La combinaison de jquerymobile avec une webview donne alors tout son potentiel:

Video

8.4 Architectures REST

Les applications clientes Android peuvent tirer partie d'une architecture de type REST car de nombreuses technologies côté serveur sont disponibles. Les principes de REST sont bien résumés sur la page wikipedia leurs étant dédiée:

- les données sont stockées dans le serveur permettant au client de ne s'occuper que de l'affichage
- chaque requête est *stateless* c'est à dire qu'elle est exécutable du côté serveur sans que celui-ci ait besoin de retenir l'état du client (son passé i.e. ses requêtes passées)
- les réponses peuvent parfois être mises en cache facilitant la montée en charge
- les ressources (services) du serveur sont clairement définies; l'état du client est envoyé par "morceaux" augmentant le nombre de requêtes.

Pour bâtir le service du côté serveur, un serveur PHP ou Tomcat peut faire l'affaire. Il pourra fournir un web service, des données au format XML ou JSON. Du côté client, il faut alors implémenter un parseur SAX ou d'objet JSON, comme montré dans [JSONREST].

Exemple de client JSON

```
HttpClient httpclient = new DefaultHttpClient();
HttpGet httpget = new HttpGet(url);
HttpResponse response;
try {
  response = httpclient.execute(httpget);
  // Examine the response status
  Log.i("Praeda",response.getStatusLine().toString());

  // Get hold of the response entity
  HttpEntity entity = response.getEntity();
  if (entity != null) {
```

```
    // A Simple JSON Response Read
    InputStream instream = entity.getContent();
    String result= convertStreamToString(instream);
    Log.i("Praeda",result);

    // A Simple JSONObject Creation
    JSONObject json=new JSONObject(result);
    Log.i("Praeda","<jsonobject>\n"+json.toString()+"\n</jsonobject>");

    // A Simple JSONObject Parsing
    JSONArray nameArray=json.names();
    JSONArray valArray=json.toJSONArray(nameArray);
    for(int i=0;i<valArray.length();i++)
    {
    Log.i("Praeda","<jsonname"+i+">\n"+nameArray.getString(i)+
      "\n</jsonname"+i+">\n"
    +"<jsonvalue"+i+">\n"+valArray.getString(i)+"\n</jsonvalue"+i+">");
    }
```

Exemple de serveur Tomcat

Du côté serveur, on peut par exemple utiliser une servlet tomcat pour capturer la reqûete.

```
public class SimpleServlet extends HttpServlet {
  public void init(ServletConfig c) throws ServletException {
    // init
  }
  public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    Vector myVector = new Vector();
    String param = req.getParameter("param");
    PrintWriter out = response.getWriter();
    // traitements
  }
  public void destroy() { }
  public String getServletInfo() { }
}
```

La servlet peut même redéléguer la partie "Vue" à une JSP, en fin de traitement:

```
<title>
  <!-- get a value that has been set by the servlet -->
  <%= request.getAttribute("title") %>
</title>
<jspx:foreach collection="<%= myVector %>">
  <jspx:item ref="myItem" type="java.lang.Integer">
    <element> <%= myItem %> </element>
  </jspx:item>
</jspx:foreach>
```

9 Divers

9.1 Librairies natives: JNI

Dans [JNI], un excellent tutoriel présente les grandes étapes pour programmer des appels natifs vers du code C, ce qui, en java s'appelle **JNI** pour *Java Native Interface*. Le but recherché est multiple:

- les appels natifs permettent de s'appuyer sur du code C déjà développé et évite de tout recoder en java
- le code ainsi déporté est davantage protégé contre la décompilation des .dex
- le code C peut éventuellement être utilisé ailleurs, par exemple sous iOS

Evidemment, ce type de développement est tout à fait orthogonal au but d'Android, c'est à dire à la programmation Java s'appuyant sur les fonctionnalités offertent par l'API Android. Dans votre code C, vous n'aurez accès à quasiment rien, ce qui restreint grandement l'intérêt de programmer des méthodes natives. Cependant, cette possibilité est très appréciée pour le développement des jeux car cela permet un accès facile aux primitives graphiques openGL.

Pour travailler avec des appels **JNI**, il faudra utiliser un outil complémentaire de Google permettant de compiler votre programme C vers une librairie partagée .so: le Android NDK.

Classe d'interface

La première étape consiste à réaliser le code Java permettant de faire le pont entre votre activité et les méthodes natives. Dans l'exemple suivant, on déclare une méthode statique (en effet, votre programme C ne sera pas un objet), appelant une méthode native dont l'implémentation réalise l'addition de deux nombres:

```
package andro.jf.jni;

public class NativeCodeInterface {

  public static native int calcul1(int x, int y);

  public static int add(int x, int y)
  {
      int somme;
      somme = calcul1(x,y);
      return somme;
  }

  static {
      System.loadLibrary("testmodule");
```

```
  }
}
```

Un appel natif depuis une activité ressemblera à:

```
TextView text = (TextView)findViewById(R.id.texte);
text.setText("5+7 = " + NativeCodeInterface.add(5, 7));
```

Génération du .h

A partir de la classe précédemment écrite, il faut utiliser l'outil **javah** pour générer le fichier .h correspondant aux méthodes natives déclarées dans le .java. En dehors du répertoire **src/** de votre projet, vous pouvez créer un répertoire **jni/** afin d'y placer les fichiers de travail de la partie C. On réalise donc la compilation, puis l'extraction du .h:

```
javac javac -d ./jni ./src/andro/jf/jni/NativeCodeInterface.java
cd ./jni
javah -jni andro.jf.jni.NativeCodeInterface
```

On obtient alors le fichier **andro_jf_jni_NativeCodeInterface.h** suivant:

```
// Header for class andro_jf_jni_NativeCodeInterface

#ifndef _Included_andro_jf_jni_NativeCodeInterface
#define _Included_andro_jf_jni_NativeCodeInterface
#ifdef __cplusplus
extern "C" {
#endif
JNIEXPORT jint JNICALL Java_andro_jf_jni_NativeCodeInterface_calcul1
(JNIEnv *, jclass, jint, jint);

#ifdef __cplusplus
  }
#endif
#endif
```

Ecriture du .c et compilation

A partir du fichier .h, il faut maintenant écrire le code de l'appel natif:

```
#include "andro_jf_jni_NativeCodeInterface.h"
JNIEXPORT jint JNICALL Java_andro_jf_jni_NativeCodeInterface_calcul1
     (JNIEnv * je, jclass jc, jint a, jint b) {
  return a+b;
}
```

Il faut ensuite créer le Makefile approprié pour la compilation du fichier .c à l'aide du script **ndk-build**. Le fichier de Makefile doit s'appler Android.mk:

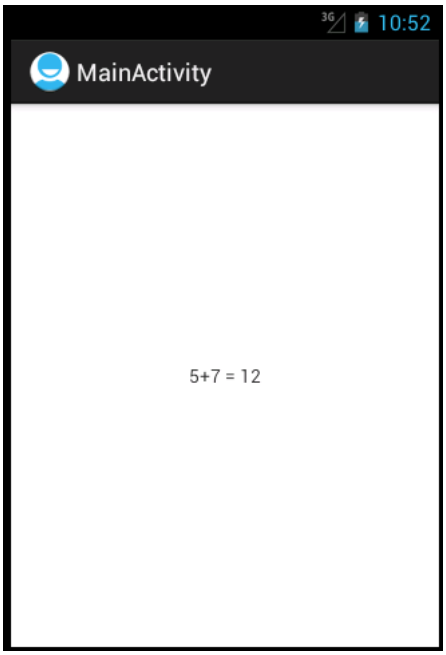
```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)

LOCAL_MODULE    := testmodule
LOCAL_CFLAGS    := -Werror
LOCAL_SRC_FILES := test.c
LOCAL_LDLIBS    := -llog

include $(BUILD_SHARED_LIBRARY)
```

Et la phase de compilation doit ressembler à:

```
/usr/local/android-ndk-r8b/ndk-build
Compile thumb  : testmodule <= test.c
SharedLibrary  : libtestmodule.so
Install        : libtestmodule.so => libs/armeabi/libtestmodule.so
```

Démonstration

L'annexe [ANX_JNI] presente le code complet d'un appel natif à une librairie C réalisant l'addition de deux nombres. Le résultat obtenu est le suivant:



10 Annexes: outils

10.1 Outils à télécharger

Les outils minimum:

- JDK 6 ou 7
- Eclipse
- ADT plugin, via l'*update manager* avec l'url https://dl-ssl.google.com/android/eclipse/
- Android SDK
- ou bien directement l'ADT bundle qui regroupe les 3 précédents.

Les outils complémentaires:

- Android apktool
- Dex2Jar
- JD-GUI
- Smali

10.2 L'émulateur Android

L'émulateur Android résulte d'une compilation des sources d'Android permettant d'exécuter le système sur un PC classique. En fonction de ce qui a été installé par le SDK, on peut lancer un émulateur compatible avec telle ou telle version de l'API.

L'émulateur reste limité sur certains aspects. Par exemple, il est difficile de tester une application utilisant l'accéléromètre ou le wifi. Il devient alors nécessaire de réaliser des tests avec un téléphone réel.

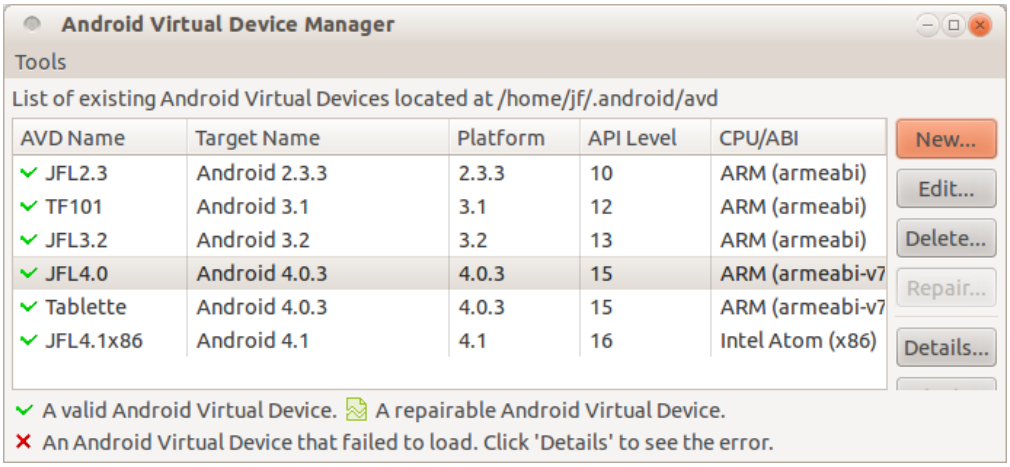
L'émulateur peut maintenant s'exécuter directement de manière native, car il est possible de le compiler pour une architecture x86. Dans les anciennes versions de l'émulateur, seuls les processeurs ARM étaient supportés car ce sont eux qui équipent la plupart de nos smartphones. Dans ce cas, l'émulateur utilise la virtualisation pour s'exécuter sur du x86.

Une version particulière de l'émulateur, nommée "Google APIs", permet de tester une application qui interroge l'un des nombreux services Google (non open-source), par exemple le service Google Map. L'émulateur "Google APIs" n'est disponible qu'en version ARM.

AVD: le gestionnaire d'appareils

[Emulator] Pour tester une application, Google fournit un émulateur Android, abrégé par AVD pour *Android Virtual Device*. Il est possible, en fonction de ce qui est présent dans le SDK installé, de créer plusieurs configurations différentes de téléphones:

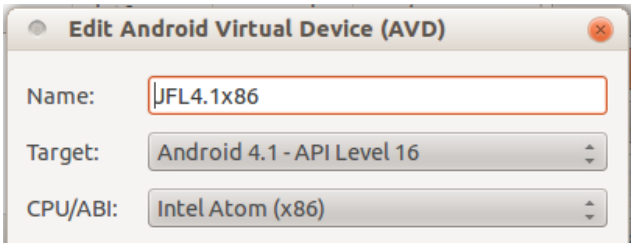
- l'outil **tools/android** permet de mettre à jour et installer différentes version du SDK
- l'outil **tools/android avd** (ajouter "avd" en ligne de commande) liste les différentes configurations d'appareils de l'utilisateur



Accéleration matérielle pour l'émulateur

Depuis Android 4.0 (ICS), Google fournit une image **kvm** de l'émulateur, compatible avec les plate-forme x86. Cette image permet de se passer de qemu qui permettait d'émuler un processeur ARM sur une architecture x86: on gagne l'*overhead* d'émulation !

Pour se faire, il faut bien entendu disposer de **kvm** et installer depuis le SDK manager l'outil *Intel x86 Atom System Image*, disponible dans ICS. Pour utiliser l'accélération, il faut créer un nouvel appareil utilisant un CPU de type *Intel Atom (x86)*, comme montré ici:



Lancer l'émulateur

[Emulator] Dans le répertoire **tools**, l'exécutable **emulator** permet de lancer l'émulateur Android en précisant la configuration d'appareil à l'aide de l'option **-avd**. La syntaxe de la ligne de commande est donc:

```
emulator -avd <avd_name> [-<option> [<value>]]
```

Par défaut, la commande sans option démarre l'émulateur.

L'émulateur peut aussi être paramétré en ligne de commande. Les options possibles permettent par exemple de:

- spécifier un serveur DNS (-dns-server <IP>)
- ralentir le CPU (-cpu-delay <value>)
- charger une partition de SDcard (-ramdisk <filepath>)
- effacer des données utilisateur (-wipe-data, -initdata)
- ...

10.3 ADB: Android Debug Bridge

L'outil **adb** (Android Debug Bridge) permet d'intéragir avec l'émulateur pour effectuer des opérations de configuration, d'installation et de débugging. Par défaut, **adb** se connecte à la seule instance de l'émulateur qui est lancée (comportement de l'option **-e**) pour exécuter les commandes demandées.

Voici quelques commandes utiles:

Installation/Désinstallation d'une application

```
./adb install newDialerOne.apk
./adb uninstall kz.mek.DialerOne
```

Pour connaitre le nom de *package* d'une application dont on est pas le développeur, on peut utiliser la commande ./aapt dump badging app.apk.

Envoi/Récupération d'un fichier

```
adb push file /sdcard/
adb pull /sdcard/file ./
```

Exécution d'un shell dans le téléphone

```
adb shell
```

Debugging

Les commandes classiques **System.out.xxx** sont redirigées vers **/dev/null**. Il faut donc débugguer autrement. Pour ce faire, le SDK fournit un moyen de visualiser les logs du système et de les filtrer au travers de l'outils **adb** situé dans le répertoire **platform-tools** du SDK. L'outil permet de se connecter à l'émulateur et d'obtenir les messages de logs. On obtient par exemple pour une exception:

```
./adb logcat
I/System.out(  483): debugger has settled (1441)
W/System.err(  483): java.io.FileNotFoundException: /test.xml
                     (No such file or directory)
                     ...
W/System.err(  483):  at andro.jfl.AndroJFLActivity.onCreate(
                     Andro7x24Activity.java:38)
```

Cependant, il est beaucoup plus propre d'utiliser la classe **Log** qui permet de classifier les logs par niveaux et par tags. Le niveau dépend des méthodes statiques utilisées, à savoir: **.v** (verbose), **.d** (debug), **.e** (error), **.w** (warning), etc. Par exemple:

```
Log.w("monTAG", "Mon message à logguer");
```

On peut alors filtrer ce log, en posant un filtre avec **adb**:

```
adb logcat TODO
```

Tester sur son téléphone

Il est extrêmement simple d'utiliser un appareil réel pour tester une application. Il faut tout d'abord activer le débogage Android afin qu'*adb* puisse se connecter à l'appareil lorsque le cable USB est branché. On voit alors apparaître l'appareil, par exemple:

```
./adb devices
List of devices attached
363234B---------      device
```

Dans Eclipse, il suffit de changer la configuration du "Run" pour "Active Devices", ce qui lancera automatiquement l'application sur le téléphone si aucun émulateur n'est lancé.

Les tests sur un appareil réel sont particulièrement utiles lorsque l'on travaille avec les capteurs.

10.4 Simuler des sensors

[SS] (cf ANX_Capteurs) Afin de simuler les sensors, vous pouvez utiliser l'outil Sensor Simulator qui permet de simuler le changement de valeurs des senseurs à partir d'un client Java en dehors de l'émulateur. Pour réaliser cela il faut:

- Téléchargez Sensor Simulator
- Lancez l'outil *bin/sensorsimulator-x.x.x.jar* (*java -jar bin/sensorsimulator-x.x.x.jar*)
- Ajoutez *bin/sensorsimulator-lib-x.x.x.jar* à votre projet Eclipse
- Ajoutez *bin/SensorSimulatorSettings-x.x.x.apk* à votre device virtuel (utiliser la commande *adb install SensorSimulatorSettings-x.x.x.apk*)

Vous pouvez, pour commencer, tester la connexion entre l'outil externe et l'outil de configuration interne de Sensor Simulator. Dans l'émulateur, lancez Sensor Simulator et après avoir vérifié la concordance entre les adresses IP, vous pouvez connecter les deux outils depuis l'onglet Testing. Vous pouvez ensuite faire bouger le device virtuel dans le client java (téléphone en haut à gauche).

Video

Adaptation de votre application au simulateur

Le principe du simulateur est de remplacer la déclaration de votre SensorManager par celui du simulateur dans le code de votre application. Ainsi, les appels à la lecture des senseurs appelera la librairie du simulateur qui, à travers une connexion réseau, ira lire les données dans le client externe Java. Remplacez votre déclaration ainsi:

```
// SensorManager manager = (SensorManager)getSystemService(SENSOR_SERVICE);
SensorManagerSimulator manager =
          SensorManagerSimulator.getSystemService(this, SENSOR_SERVICE);
manager.connectSimulator();
```

Attention aux directives import qui, par défaut, importent les classes de *android.hardware* et doivent désormais utiliser *org.openintents.sensorsimulator.hardware*. Ne pas oublier non plus d'ajouter la permission permettant à votre application d'accéder à internet. Quelques adaptations du code peuvent être nécessaires...

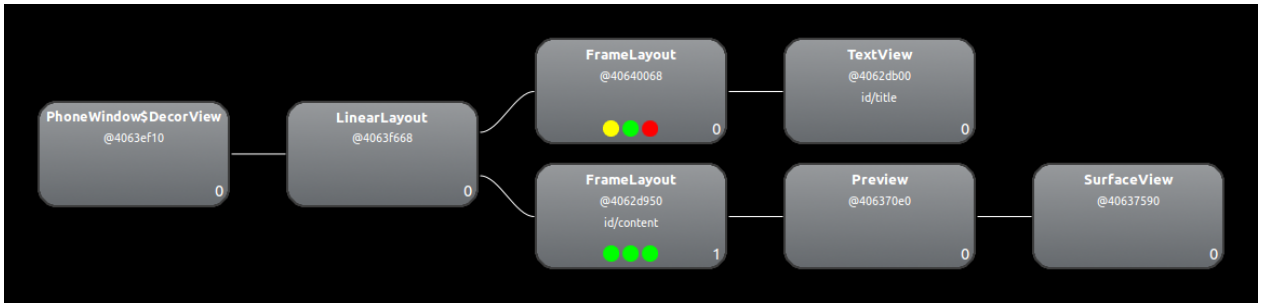
Video

10.5 HierarchyViewer

Android fournit un outil d'analyse *live* de la hiérarchie des objets graphiques. Ainsi, il est possible de mieux comprendre sa propre application lors de la phase de débugging. Pour l'utiliser, il faut lancer: **./hierarchyviewer**.

Dans l'exemple suivant, on voit deux **FrameLayout** qui sont enfants de l'éléments de plus haut niveau **LinearLayout**. Le premier **FrameLayout** a été décroché de l'élément *root* et remplacé par un nouveau **FrameLayout** contenant l'élément **Preview** permettant de visualiser ce que voit la caméra arrière (cf [ANX_Camera]).

Les trois points de couleur représentent la vitesse de rendu. De gauche à droite on a le temps utilisé pour mesurer (calculer les dimensions), positionner (calculer la position des enfants) et dessiner . L'interprétation est: vert: 50% plus rapide que les autres éléments graphiques, jaune: moins de 50%, rouge: le plus lent de la hiérarchie.



11 Annexes: codes sources

11.1 Interfaces graphiques

Main.java

```
public class Main extends Activity {

  @Override
  protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    Toast.makeText(this, "SAUVEGARDE", Toast.LENGTH_LONG).show();
  }

  public void onCreate(Bundle savedInstanceState) {

    setContentView(R.layout.acceuil);
    super.onCreate(savedInstanceState);
    TextView texte = (TextView)findViewById(R.id.le_texte);
```

```
    texte.setText("Here we go !");

    Button b = (Button)findViewById(R.id.Button01);
    b.setOnClickListener(new OnClickListener() {

      @Override
      public void onClick(View v) {
        Toast.makeText(v.getContext(), "Stop !", Toast.LENGTH_LONG).show();
      }
    });

    // Changing layout
    LinearLayout l = (LinearLayout)findViewById(R.id.accueilid);
    l.setBackgroundColor(Color.GRAY);
  }
}
```

Layout

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



  <TextView android:id="@+id/le_texte" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:text="@string/hello"
    android:layout_gravity="center" />


  <EditText android:text="" android:id="@+id/EditText01"
    android:layout_width="fill_parent" android:layout_height="wrap_content">
  </EditText>
  <EditText android:text="" android:layout_width="fill_parent"
    android:layout_height="wrap_content">
  </EditText>
  <ImageView android:id="@+id/logoEnsi" android:src="@drawable/ensi"
    android:layout_width="100px" android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal">
  </ImageView>

  <Button android:text="Go !" android:id="@+id/Button01"
    android:layout_width="wrap_content" android:layout_height="wrap_content">
  </Button>
</LinearLayout>
```

Activity2.java

```
public class Activity2 extends Activity {
  public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout gabarit = new LinearLayout(this);
        gabarit.setOrientation(LinearLayout.VERTICAL);
        gabarit.setGravity(Gravity.CENTER);

        // Label
        TextView texte = new TextView(this);
        texte.setText("Programming creation of interface !");
        gabarit.addView(texte);

        // Zone de texte
        EditText edit = new EditText(this);
        edit.setText("Edit me");
        gabarit.addView(edit);

        // Image
        ImageView image = new ImageView(this);
        image.setImageResource(R.drawable.ensi);
        gabarit.addView(image);

        setContentView(gabarit);
    }
}
```

11.2 Listes pour des items texte

AndroListsSimpleActivity.java

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

        ListView list = (ListView)findViewById(R.id.maliste);
        ArrayAdapter<String> tableau = new ArrayAdapter<String>(
                list.getContext(), R.layout.montexte);
        for (int i=0; i<40; i++) {
            tableau.add("coucou " + i);
        }
        list.setAdapter(tableau);

    }
}
```

Layout

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

Layout d'une ligne de liste

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

11.3 Listes de layouts complexes

AndroListsActivity.java

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

    ListView list = (ListView)findViewById(R.id.maliste);
    ArrayAdapter<String> tableau = new ArrayAdapter<String>(
        list.getContext(), R.layout.ligne, R.id.monTexte);
    for (int i=0; i<40; i++) {
      tableau.add("coucou " + i);
    }
    list.setAdapter(tableau);
  }
}
```

Layout

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

Layout d'une ligne de liste

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_height="fill_parent" android:orientation="horizontal"
  android:layout_width="fill_parent" android:baselineAligned="true"
  android:gravity="center">
  <TextView android:layout_height="wrap_content" android:id="@+id/monTexte"
    android:text="TextView"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:layout_width="wrap_content">
  </TextView>
  <LinearLayout android:layout_height="wrap_content"
    android:orientation="horizontal" android:layout_width="fill_parent"
    android:gravity="right" android:layout_gravity="center">
    <ImageView android:id="@+id/monImage" android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:src="@android:drawable/ic_menu_compass">
    </ImageView>

  </LinearLayout>
</LinearLayout>
```

11.4 Fragments

MainActivity.java

```
public class MainActivity extends Activity {

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

    if (savedInstanceState == null) {
      getFragmentManager().beginTransaction()
      .add(R.id.container, new PlaceholderFragment()).commit();
    }
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar
    // if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
  }

  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
```

```
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();
    if (id == R.id.action_settings) {
      return true;
    }
    return super.onOptionsItemSelected(item);
  }

  /**
   * A placeholder fragment containing a simple view.
   */
  public static class PlaceholderFragment extends Fragment {

    public PlaceholderFragment() {
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
      View rootView = inflater.inflate(R.layout.fragment_main_dyn, container,
          false);
      return rootView;
    }
  }
}
```

MainTabActivity.java

```
public class MonFragmentStatique extends Fragment {
  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {

    View v = inflater.inflate(R.layout.fragment_main, container, false);
    return v;
  }
}
```

Layout du Main

```
<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
```

```
    tools:context="andro.jf.androfragments.MainActivity$PlaceholderFragment" >

  <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Fragment statique" />

</RelativeLayout>
```

Layout du fragment statique

```
<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="andro.jf.androfragments.MainActivity$PlaceholderFragment" >

  <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Fragment statique" />

</RelativeLayout>
```

Layout du fragment dynamique

```
<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="andro.jf.androfragments.MainActivity$PlaceholderFragment" >

  <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Fragment dynamique !" />

</RelativeLayout>
```

11.5 Onglets

AndroTabs2Activity.java

```
public class MainTabActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main_tab);


    final ActionBar actionBar = getActionBar();

    // Specify that tabs should be displayed in the action bar.
    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

    // Create a tab listener that is called when the user changes tabs.
    ActionBar.TabListener tabListener = new ActionBar.TabListener() {
      public void onTabSelected(ActionBar.Tab tab, FragmentTransaction ft) {

        FragmentTab newft = new FragmentTab();
        if (tab.getText().equals("Tab 2")) {
          newft.setChecked(true);
        }

        ft.replace(R.id.fragmentContainer, newft);
      }

      public void onTabUnselected(ActionBar.Tab tab, FragmentTransaction ft) {
        // hide the given tab
      }

      public void onTabReselected(ActionBar.Tab tab, FragmentTransaction ft) {
        // probably ignore this event
      }
    };

    // Add 3 tabs, specifying the tab's text and TabListener
    for (int i = 0; i < 3; i++) {
      Tab t = actionBar.newTab().setText("Tab " + (i + 1));
      t.setTabListener(tabListener);
      actionBar.addTab(t);
    }




  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
```

```
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main_tab, menu);
    return true;
  }

  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();
    if (id == R.id.action_settings) {
      return true;
    }
    return super.onOptionsItemSelected(item);
  }
}
```

Layout du Main

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="jf.andro.androtabs3.MainTabActivity" >

  <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />


  <FrameLayout
        android:id="@+id/fragmentContainer"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@+id/textView1"
        android:layout_marginRight="22dp" >

  </FrameLayout>

</RelativeLayout>
```

FragmentTab.java

```java
public class FragmentTab extends Fragment {

  boolean checked = false;

  public void setChecked(boolean checked) {
    this.checked = checked;
  }

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

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    View view =  inflater.inflate(R.layout.fragment1, container, false);
    CheckBox cb = (CheckBox) view.findViewById(R.id.checkBox1);
    cb.setChecked(this.checked);

    return view;
  }

}
```

Layout du fragment

```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" >

  <CheckBox
        android:id="@+id/checkBox1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="CheckBox" />

</LinearLayout>
```

11.6 Intents

Main.java

```java
public class Main extends Activity {

  private MyBroadcastReceiverDyn myreceiver;
```

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

    Button call = (Button)findViewById(R.id.call);
    call.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        Intent login = new Intent(getApplicationContext(),
            GivePhoneNumber.class);
        startActivity(login);
      }});
    Button info = (Button)findViewById(R.id.info);
    info.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        Intent login = new Intent(getApplicationContext(),
            GivePhoneNumber.class);
        startActivityForResult(login,48);
      }});

    // Broadcast receiver dynamique
    myreceiver = new MyBroadcastReceiverDyn();
    IntentFilter filtre = new IntentFilter("andro.jf.broadcast");
    registerReceiver(myreceiver, filtre);
  }
  @Override
  protected void onActivityResult(int requestCode, int resultCode,
              Intent data) {
    if (requestCode == 48)
    Toast.makeText(this, "Code de requête récupéré (je sais d'ou je viens)",
          Toast.LENGTH_LONG).show();
    if (resultCode == 50)
    Toast.makeText(this, "Code de retour ok (on m'a renvoyé le bon code)",
          Toast.LENGTH_LONG).show();
  }
}
```

Layout du Main

```
<?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="Main Activity:" />
  <Button android:text="Call" android:id="@+id/call"
    android:layout_width="wrap_content" android:layout_height="wrap_content">
  </Button>
  <Button android:text="Call with result" android:id="@+id/info"
    android:layout_width="wrap_content" android:layout_height="wrap_content">
```

```
  </Button>
</LinearLayout>
```

Seconde activité

```
public class GivePhoneNumber extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.givephonenumber);

    // Bouton ok
    Button b = (Button)findViewById(R.id.ok);
    b.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        EditText number = (EditText)findViewById(R.id.number);
        Uri telnumber = Uri.parse("tel:" + number.getText().toString());
        Intent call = new Intent(Intent.ACTION_DIAL, telnumber);
        startActivity(call);
      }
    });
    // Bouton finish
    Button finish = (Button)findViewById(R.id.finish);
    finish.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        setResult(50);
        finish();
      }
    });

    // Bouton appel direct
    Button direct = (Button)findViewById(R.id.direct);
    direct.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        EditText number = (EditText)findViewById(R.id.number);
        Uri telnumber = Uri.parse("tel:" + number.getText().toString());
        Intent call = new Intent(Intent.ACTION_CALL, telnumber);
        startActivity(call);
      }});

    // Bouton broadcast
    Button broad = (Button)findViewById(R.id.broadcast);
    broad.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        Intent broadcast = new Intent("andro.jf.broadcast");
        broadcast.putExtra("extra", "test");
        sendBroadcast(broadcast);
      }});
```

```
    // Bouton pour s'auto appeler
    Button autoinvoc = (Button)findViewById(R.id.autoinvoc);
    autoinvoc.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        Intent call = new Intent("andro.jf.nom_du_message");
        call.putExtra("extra", "test");
        startActivity(call);
      }});
  }
}
```

Layout de la seconde activité

```
<?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:text="Renseignez votre login:" android:id="@+id/TextView01"
    android:layout_width="wrap_content" android:layout_height="wrap_content"
    xmlns:android="http://schemas.android.com/apk/res/android">
  </TextView>

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

  <Button android:text="Ok" android:id="@+id/ok"
    android:layout_width="wrap_content" android:layout_height="wrap_content">
  </Button>
  <Button android:text="Finish" android:id="@+id/finish"
    android:layout_width="wrap_content" android:layout_height="wrap_content">
  </Button>
  <Button android:text="Direct Call" android:id="@+id/direct"
    android:layout_width="wrap_content" android:layout_height="wrap_content">
  </Button>
  <Button android:text="Broadcast" android:id="@+id/broadcast"
    android:layout_width="wrap_content" android:layout_height="wrap_content">
  </Button>
  <Button android:text="Auto call" android:id="@+id/autoinvoc"
    android:layout_width="wrap_content" android:layout_height="wrap_content">
  </Button>
</LinearLayout>
```

11.7 Receveur de broadcasts

MyBroadcastReceiver.java

```java
public class MyBroadcastReceiver extends BroadcastReceiver {

  @Override
  public void onReceive(Context context, Intent intent) {
    Bundle extra = intent.getExtras();
    if (extra != null)
    {
      String val = extra.getString("extra");
      Toast.makeText(context, "Broadcast message received: " + val,
          Toast.LENGTH_SHORT).show();
    }
  }
}
```

Manifest

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="andro.jf"
      android:versionCode="1"
      android:versionName="1.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".Main"
                android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
      <intent-filter>
        <action android:name="andro.jf.nom_du_message" />
        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </activity>
    <activity android:name=".GivePhoneNumber"
                android:label="Renseigner le login"/>
    <receiver android:name="MyBroadcastReceiver">
      <intent-filter>
        <action android:name="andro.jf.broadcast" />
        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </receiver>
  </application>
  <uses-permission android:name="android.permission.RECEIVE_SMS"/>
  <uses-permission android:name="android.permission.CALL_PHONE"/>

  <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="4" />


</manifest>
```

11.8 Gestion des préférences

Main.java

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

    // Passé à l'activité des préférences
    View button = findViewById(R.id.prefs);
    button.setOnClickListener( new View.OnClickListener() {

      @Override
      public void onClick(View v) {
        Intent go_to_prefs = new Intent(getApplicationContext(),
            MesPreferences.class);
        startActivity(go_to_prefs);

      }
    });

    // Récupération d'une valeur de préférence
    View recup_login = findViewById(R.id.recup_login);
    recup_login.setOnClickListener( new View.OnClickListener() {

      @Override
      public void onClick(View v) {
        SharedPreferences prefs =
            PreferenceManager.getDefaultSharedPreferences(
            getApplicationContext());

        Toast.makeText(getApplicationContext(), "Recup login: " +
        prefs.getString("login",""),
            Toast.LENGTH_SHORT).show();
        Toast.makeText(getApplicationContext(), "Recup vitesse: " +
            prefs.getString("vitesse",""),
            Toast.LENGTH_SHORT).show();
      }
    });
  }
}
```

MesPreferences.java

```
public class MesPreferences extends PreferenceActivity {

  @Override
```

```
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.prefs);
  }

}
```

Preferences XML

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


  <CheckBoxPreference android:title="Diffusion du login"
  android:summaryOff="Ne pas diffuser" android:summaryOn="Diffuser"
  android:key="diffuser">
  </CheckBoxPreference>
  <EditTextPreference android:summary="login d'authentification"
    android:title="Login" android:key="login" android:dependency="diffuser">
  </EditTextPreference>
  <EditTextPreference android:title="Password"
    android:key="password" android:dependency="diffuser">
  </EditTextPreference>
  <ListPreference android:title="Vitesse"
              android:key="vitesse"
              android:entries="@array/key"
                android:entryValues="@array/value"
                android:dialogTitle="Choisir la vitesse:"
                android:persistent="true">
  </ListPreference>
</PreferenceScreen>
```

11.9 Demarrage de services

AutoStart.java

```
public class AutoStart extends BroadcastReceiver {
  public void onReceive(Context context, Intent intent) {
    Intent startServiceIntent = new Intent(context, AndroService.class);
    context.startService(startServiceIntent);
  }
}
```

Manifest

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

```
  package="andro.jf" android:versionCode="1" android:versionName="1.0">
  <uses-sdk android:minSdkVersion="10" />

  <application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".LocalStartUI" android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <service android:name=".AndroService">
      <intent-filter>
        <action android:name="andro.jf.manageServiceAction" />
      </intent-filter>
    </service>
    <receiver android:name=".AutoStart">
      <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
      </intent-filter>
    </receiver>
  </application>

</manifest>
```

LocalStartUI.java

```
public class LocalStartUI extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    // TODO Auto-generated method stub
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    Button b = (Button) findViewById(R.id.button1);
    b.setOnClickListener(new OnClickListener() {

      @Override
      public void onClick(View v) {
        Intent startService = new Intent("andro.jf.manageServiceAction");
        startService(startService);

      }
    });
    Button b2 = (Button) findViewById(R.id.button2);
    b2.setOnClickListener(new OnClickListener() {

      @Override
      public void onClick(View v) {
        Intent stopService = new Intent("andro.jf.manageServiceAction");
        stopService(stopService);
```

```
      }
    });
  }


}
```

AndroService.java

```
public class AndroService extends Service {

  Timer timer;
  TimerTask task;

  public void onCreate() {
    // Création du service
    Toast.makeText(this, "Service Created",
        Toast.LENGTH_SHORT).show();
    timer = new Timer();
  }
  public void onDestroy() {
    // Destruction du service
    Toast.makeText(this, "Destruction du service",
        Toast.LENGTH_SHORT).show();
    timer.cancel();
  }
  public int onStartCommand(Intent intent, int flags, int startId) {
    // Démarrage du service
    Toast.makeText(this, "Démarrage du service", Toast.LENGTH_SHORT).show();
    /* boolean blop = true; // Vraiment pas une bonne idée !
    while (blop == true)
      ; */
    final Handler handler = new Handler();
    task = new TimerTask() {
      public void run() {
        handler.post(new Runnable() {
          public void run() {
            Toast.makeText(AndroService.this, "plop !",
                Toast.LENGTH_SHORT).show();
          }
        });
      }
    };
    timer.schedule(task, 0, 5000);

    return START_STICKY;
  }
  public IBinder onBind(Intent arg0) {
    return null;
```

```
  }
}
```

11.10 Binding entre service et activite pour un meme processus

AndroService.java

```
public class AndroService extends Service {

  /* For binding this service */
  //La donnée à transmettre à l'activité
  private int infoOfService = 0;
  //L'implémentation de l'objet de binding
  private final IBinder ib = new MonServiceBinder();
  private class MonServiceBinder extends Binder
        implements AndroServiceInterface {
    // Cette classe qui hérite de Binder
    // implémente une méthode définie dans l'interface
    public int getInfo() {
      return infoOfService;
    }
  }

  public IBinder onBind(Intent arg0) {
    return ib;
  }

  /* Service stuff */
  public void onCreate() {
    // Création du service
    Toast.makeText(this, "Service Created", Toast.LENGTH_SHORT).show();
  }
  public void onDestroy() {
    // Destruction du service
  }
  public int onStartCommand(Intent intent, int flags, int startId) {
    // Démarrage du service
    Thread t = new Thread(new Runnable() {

      @Override
      public void run() {

        try {
          Thread.sleep(4000);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
        infoOfService = 12;
      }
    });
```

```
    t.start();

    return START_STICKY;
  }

}
```

AndroServiceInterface.java

```
package andro.jf;

/* C'est l'interface utilisée de part et d'autre (activité / service).
 * Le service va créer un objet de type Binder qui implémente cette interface.
 * L'activité va recevoir cet objet (sans connaitre son type) mais sait qu'il
 * implémentera cette interface.
 */
public interface AndroServiceInterface {

  public int getInfo();

}
```

AndroServiceBindActivity.java

```
public class AndroServiceBindActivity extends Activity {
  /** Called when the activity is first created. */

  private int infoFromService = 0;
  /* Code disponible lorsque l'activité sera connectée au service */
  private ServiceConnection maConnexion = new ServiceConnection() {

    /* Code exécuté lorsque la connexion est établie: */
    public void onServiceConnected(ComponentName name, IBinder service) {
      AndroServiceInterface myBinder = (AndroServiceInterface)service;
      // stockage de l'information provevant du service
      infoFromService = myBinder.getInfo();

    }
    public void onServiceDisconnected(ComponentName name) {
    }
  };


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

    Intent serv = new Intent(this,AndroService.class);
    startService(serv);
```

```
    Button b = (Button)findViewById(R.id.button1);
    b.setOnClickListener(new OnClickListener() {

      @Override
      public void onClick(View v) {
        /* Intent pour la connexion au service */
        Intent intentAssociation =
            new Intent(AndroServiceBindActivity.this, AndroService.class);
        /* Bind au service: à refaire à chaque
         * appui sur le bouton pour rafraichir la valeur */
        bindService(intentAssociation, maConnexion,
            Context.BIND_AUTO_CREATE);
        Toast.makeText(getApplicationContext(),
            "Info lue dans le service: "  + infoFromService,
            Toast.LENGTH_SHORT).show();
        /* On se déconnecte du service */
        unbindService(maConnexion);

      }
    });



  }
}
```

Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="andro.jf" android:versionCode="1" android:versionName="1.0">
  <uses-sdk android:minSdkVersion="10" />

  <application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".AndroServiceBindActivity"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <service android:name=".AndroService" />
    <!-- not possible: android:process=":p2" -->
  </application>
</manifest>
```

11.11 Binding inter processus utilisant le langage AIDL

AndroService.java

```
public class AndroService extends Service {

  /* For binding this service */
  // La donnée à transmettre à l'activité
  private int infoOfService = 0;
  private AndroServiceInterface.Stub ib =
      new AndroServiceInterface.Stub() {
    @Override
    public int getInfo() throws RemoteException {
      // Cette classe qui hérite de Binder
      // implémente une méthode définie dans l'interface
      return infoOfService;
    }
  };

  public IBinder onBind(Intent arg0) {
    return ib;
  }

  /* Service stuff */
  public void onCreate() {
    // Création du service
    Toast.makeText(this, "Service Created",
        Toast.LENGTH_SHORT).show();
  }
  public void onDestroy() {
    // Destruction du service
  }
  public int onStartCommand(Intent intent, int flags, int startId) {
    // Démarrage du service
    Thread t = new Thread(new Runnable() {

      @Override
      public void run() {

        try {
          Thread.sleep(4000);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
        infoOfService = 12;
      }
    });
    t.start();

    return START_STICKY;
  }

}
```

AndroServiceInterface.aidl

```
package andro.jf;

interface AndroServiceInterface {

    int getInfo();
}
```

AndroServiceInterface.java généré

```
/*
 * This file is auto-generated.  DO NOT MODIFY.
 * Original file: /home/jf/workspace/AndroServiceBind2/src/andro/jf/AndroServiceInterfac
 */
package andro.jf;
public interface AndroServiceInterface extends android.os.IInterface
{
/** Local-side IPC implementation stub class. */
public static abstract class Stub extends android.os.Binder implements andro.jf.AndroSer
{
private static final java.lang.String DESCRIPTOR = "andro.jf.AndroServiceInterface";
/** Construct the stub at attach it to the interface. */
public Stub()
{
this.attachInterface(this, DESCRIPTOR);
}
/**
 * Cast an IBinder object into an andro.jf.AndroServiceInterface interface,
 * generating a proxy if needed.
 */
public static andro.jf.AndroServiceInterface asInterface(android.os.IBinder obj)
{
if ((obj==null)) {
return null;
}
android.os.IInterface iin = (android.os.IInterface)obj.queryLocalInterface(DESCRIPTOR);
if (((iin!=null)&&(iin instanceof andro.jf.AndroServiceInterface))) {
return ((andro.jf.AndroServiceInterface)iin);
}
return new andro.jf.AndroServiceInterface.Stub.Proxy(obj);
}
public android.os.IBinder asBinder()
{
return this;
}
@Override public boolean onTransact(int code, android.os.Parcel data, android.os.Parcel
{
switch (code)
{
case INTERFACE_TRANSACTION:
{
reply.writeString(DESCRIPTOR);
```

```
return true;
}
case TRANSACTION_getInfo:
{
data.enforceInterface(DESCRIPTOR);
int _result = this.getInfo();
reply.writeNoException();
reply.writeInt(_result);
return true;
}
}
return super.onTransact(code, data, reply, flags);
}
private static class Proxy implements andro.jf.AndroServiceInterface
{
private android.os.IBinder mRemote;
Proxy(android.os.IBinder remote)
{
mRemote = remote;
}
public android.os.IBinder asBinder()
{
return mRemote;
}
public java.lang.String getInterfaceDescriptor()
{
return DESCRIPTOR;
}
public int getInfo() throws android.os.RemoteException
{
android.os.Parcel _data = android.os.Parcel.obtain();
android.os.Parcel _reply = android.os.Parcel.obtain();
int _result;
try {
_data.writeInterfaceToken(DESCRIPTOR);
mRemote.transact(Stub.TRANSACTION_getInfo, _data, _reply, 0);
_reply.readException();
_result = _reply.readInt();
}
finally {
_reply.recycle();
_data.recycle();
}
return _result;
}
}
static final int TRANSACTION_getInfo = (android.os.IBinder.FIRST_CALL_TRANSACTION + 0);
}
public int getInfo() throws android.os.RemoteException;
}
```

AndroServiceBind2Activity.java

```
public class AndroServiceBind2Activity extends Activity {

  private int infoFromService = 0;
  /* Code disponible lorsque l'activité sera connectée au service */
  private ServiceConnection maConnexion = new ServiceConnection() {

    /* Code exécuté lorsque la connexion est établie: */
    public void onServiceConnected(ComponentName name, IBinder service) {
      AndroServiceInterface myBinder =
          AndroServiceInterface.Stub.asInterface(service);
      try {
        // stockage de l'information provevant du service
        infoFromService = myBinder.getInfo();
      } catch (RemoteException e) {
        e.printStackTrace();
      }
    }
    public void onServiceDisconnected(ComponentName name) {
    }
  };


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

    Intent serv = new Intent(this,AndroService.class);
    startService(serv);

    Button b = (Button)findViewById(R.id.button1);
    b.setOnClickListener(new OnClickListener() {

      @Override
      public void onClick(View v) {
        /* Intent pour la connexion au service */
        Intent intentAssociation = new
            Intent(AndroServiceBind2Activity.this, AndroService.class);
        /* Bind au service: à refaire à chaque
         * appui sur le bouton pour rafraichir la valeur */
        bindService(intentAssociation, maConnexion,
Context.BIND_AUTO_CREATE);
        Toast.makeText(getApplicationContext(),
            "Info lue dans le service: " + infoFromService,
            Toast.LENGTH_SHORT).show();
        /* On se déconnecte du service */
        unbindService(maConnexion);

      }
    });
```

```
  }
}
```

Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="andro.jf" android:versionCode="1" android:versionName="1.0">
  <uses-sdk android:minSdkVersion="10" />

  <application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".AndroServiceBind2Activity"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <service android:name=".AndroService" android:process=":p2" />
    <!-- Now possible -->
  </application>
</manifest>
```

11.12 Processus indépendants

Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="andro.jf" android:versionCode="1" android:versionName="1.0">
  <uses-sdk android:minSdkVersion="7" />

  <application android:icon="@drawable/icon"
               android:label="@string/app_name"
    android:process="andro.jf">
    <activity android:name=".AndroProcessActivity"
          android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <service android:name=".AndroService" android:process=":p2">
    </service>

  </application>
</manifest>
```

AndroService.java

```java
public class AndroService extends Service {

    public void onCreate() {
    }
    public void onDestroy() {
    }
    public int onStartCommand(Intent intent, int flags, int startId) {
        // Démarrage du service
        boolean b = true;
        while (b)
            ;
        return START_STICKY;
    }
    public IBinder onBind(Intent arg0) {
        return null;
    }
}
```

11.13 Tâches asynchrones

MainActivity

```java
public class MainActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ImageView img = (ImageView) findViewById(R.id.img);
        TextView txt = (TextView) findViewById(R.id.textView1);
        BitmapDownloaderTask task = new BitmapDownloaderTask(img, txt);
        task.execute("http://www.univ-orleans.fr/lifo/Members/" +
        "Jean-Francois.Lalande/enseignement/android/images/androids.png");
    }
}
```

BitmapDownloaderTask

```java
public class BitmapDownloaderTask
    extends AsyncTask<String, Integer, Bitmap> {

    private final WeakReference<ImageView> imageViewReference;
    private final WeakReference<TextView> textViewReference;

    public BitmapDownloaderTask(ImageView imageView, TextView textView) {
        imageViewReference = new WeakReference<ImageView>(imageView);
        textViewReference = new WeakReference<TextView>(textView);
    }

    // Actual download method, run in the task thread
```

```
    protected Bitmap doInBackground(String... params) {
        // params comes from the execute() call: params[0] is the url.
        String url = params[0];

        publishProgress(new Integer(0));

        AndroidHttpClient client = AndroidHttpClient.newInstance("Android");
        HttpGet getRequest = new HttpGet(url);

        try {
            Thread.sleep(2000); // To simulate a slow downloading rate
            HttpResponse response = client.execute(getRequest);
            publishProgress(new Integer(1));
            Thread.sleep(1000); // To simulate a slow downloading rate
            final int statusCode = response.getStatusLine().getStatusCode();
            if (statusCode != HttpStatus.SC_OK) {
                Log.w("ImageDownloader", "Error " + statusCode
                        + " while retrieving bitmap from " + url);
                return null;
            }

            HttpEntity entity = response.getEntity();
            if (entity != null) {
                InputStream inputStream = null;
                try {
                    inputStream = entity.getContent();
                    Bitmap bitmap = BitmapFactory.decodeStream(inputStream);
                    publishProgress(new Integer(3));
                    return bitmap;
                } finally {
                    if (inputStream != null) {
                        inputStream.close();
                    }
                    entity.consumeContent();
                }
            }
        } catch (Exception e) {
            // Could provide a more explicit error message
            // for IOException or IllegalStateException
            getRequest.abort();
            Log.w("ImageDownloader",
              "Error while retrieving bitmap from " + url);
        } finally {
            if (client != null) {
                client.close();
            }
        }
        return null;
    }

    @Override
    protected void onProgressUpdate(Integer... values) {
```

```
        Integer step = values[0];
        if (textViewReference != null) {
            textViewReference.get().setText("Step: " + step.toString());
        }
    }

    @Override
    // Once the image is downloaded, associates it to the imageView
    protected void onPostExecute(Bitmap bitmap) {
        if (isCancelled()) {
            bitmap = null;
        }

        if (imageViewReference != null) {
            ImageView imageView = imageViewReference.get();
            if (imageView != null) {
                imageView.setImageBitmap(bitmap);
            }
        }
    }

    @Override
    protected void onPreExecute() {
        if (imageViewReference != null) {
            ImageView imageView = imageViewReference.get();
            if (imageView != null) {
                imageView.setImageResource(R.drawable.interro);
            }
        }
    }
}
```

11.14 Réseau

Main.java

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

    // Réseau
    ConnectivityManager manager =
       (ConnectivityManager)getSystemService(CONNECTIVITY_SERVICE);
    NetworkInfo net = manager.getActiveNetworkInfo();
    Toast.makeText(getApplicationContext(), "" + net.getType(),
          Toast.LENGTH_LONG).show();

    // Wifi
    WifiManager wifi = (WifiManager)getSystemService(Context.WIFI_SERVICE);
```

```
    if (!wifi.isWifiEnabled())
      wifi.setWifiEnabled(true);
    manager.setNetworkPreference(ConnectivityManager.TYPE_WIFI);

    // Bluetooth
    BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();
    if (!bluetooth.isEnabled()){
      Intent launchBluetooth =
        new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
      startActivity(launchBluetooth);
    }

    Intent discoveryMode =
      new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
    discoveryMode.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 60);
    startActivity(discoveryMode);

    Set<BluetoothDevice> s = bluetooth.getBondedDevices();
    for (BluetoothDevice  ap : s)
      Toast.makeText(getApplicationContext(), "" + ap.getName(),
            Toast.LENGTH_LONG).show();

  }
}
```

Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="andro.jf"
      android:versionCode="1"
      android:versionName="1.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".Main"
                android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
      <receiver android:name=".ClasseGerantLAppel">
        <intent-filter>
          <action android:name="Intent.ACTION_CALL"/>
        </intent-filter>
      </receiver>
    </activity>

  </application>
  <uses-sdk android:minSdkVersion="5" />


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

```
  </uses-permission>
  <uses-permission android:name="android.permission.WRITE_SECURE_SETTINGS">
  </uses-permission>
  <uses-permission android:name="android.permission.WRITE_SETTINGS">
  </uses-permission>
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION">
  </uses-permission>
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION">
  </uses-permission>
</manifest>
```

Bluetooth

```
public class BluetoothBroadcastReceiver extends BroadcastReceiver {
    public void onReceive(Context context, Intent intent) {
      String action = intent.getAction();
      BluetoothDevice appareil = null;
  if (action.equals(BluetoothDevice.ACTION_FOUND))
        appareil = (BluetoothDevice)intent.getParcelableExtra(
          BluetoothDevice.EXTRA_DEVICE);
    }
}
```

11.15 Localisation

Main.java

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

        // Fournisseurs de service
        LocationManager manager =
         (LocationManager)getSystemService(Context.LOCATION_SERVICE);
        List<String> fournisseurs = manager.getAllProviders();
        for (String f : fournisseurs)
          Toast.makeText(getApplicationContext(), "" + f,
                Toast.LENGTH_SHORT).show();

        // Récupération de la localisation
        Location localisation = manager.getLastKnownLocation("gps");
        Toast.makeText(getApplicationContext(), "Lattitude" +
          localisation.getLatitude(), Toast.LENGTH_SHORT).show();
        Toast.makeText(getApplicationContext(), "Longitude" +
          localisation.getLongitude(), Toast.LENGTH_SHORT).show();
```

```
        // Ecouteur de changement de localisation
        manager.requestLocationUpdates("gps", 6000, 100,
                new LocationListener() {
      public void onStatusChanged(String provider,
                                  int status, Bundle extras) {
      }
      public void onProviderEnabled(String provider) {
      }
      public void onProviderDisabled(String provider) {
      }
      public void onLocationChanged(Location location) {
        // TODO Auto-generated method stub
      }
    });
    }
}
```

Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="andro.jf"
      android:versionCode="1"
      android:versionName="1.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".Main"
                  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="5" />

</manifest>
```

11.16 Geocoding

AndroReverseGeocodingActivity.java

```
public class AndroReverseGeocodingActivity extends Activity {
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    TextView address = (TextView)findViewById(R.id.address);

    ReverseGeocodingTask rgt = new ReverseGeocodingTask(this, address);
```

```
    Location ensib = new Location("JFL provider");
    ensib.setLatitude(47.082687);
    ensib.setLongitude(2.415916);
    rgt.execute(new Location[] {ensib});
  }
}
```

ReverseGeocodingTask.java

```
public class ReverseGeocodingTask extends AsyncTask<Location, Void, String> {
  Context mContext;
  TextView addresseResult;
  protected void onPostExecute(String result) {
    addresseResult.setText(result);
  }

  public ReverseGeocodingTask(Context context, TextView result) {
    super();
    mContext = context;
    this.addresseResult = result;
  }

  @Override
  protected String doInBackground(Location... params) {
    String addressText = null;
    Geocoder geocoder = new Geocoder(mContext, Locale.getDefault());
    System.out.println("Geocoder is present ? " + geocoder.isPresent());

    Location loc = params[0];
    List<Address> addresses = null;
    try {
      // Call the synchronous getFromLocation() method
      // by passing in the lat/long values.
      addresses = geocoder.getFromLocation(loc.getLatitude(),
          loc.getLongitude(), 1);
    } catch (IOException e) {
      e.printStackTrace();
      System.out.println("Erreur: " + e.toString());
    }
    System.out.println("Adresses: " + addresses);
    if (addresses != null && addresses.size() > 0) {
      Address address = addresses.get(0);
      // Format the first line of address (if available),
      // city, and country name.
      addressText = String.format("%s, %s, %s",
          address.getMaxAddressLineIndex() > 0 ? address.getAddressLine(0) :
              "",
              address.getLocality(),
              address.getCountryName());
      System.out.println("Adresse: " + addressText);
```

```
    }
    return addressText;
  }
}
```

11.17 Applications hybrides

MainActivity.java

```
public class MainActivity extends Activity {
  WebView wv = null;
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    wv = (WebView)findViewById(R.id.webView1);
    wv.setWebViewClient(new MyWebViewClient());
    wv.getSettings().setJavaScriptEnabled(true);
    wv.loadUrl("http://www.univ-orleans.fr/lifo/Members/"
        + "Jean-Francois.Lalande/jquerymobiletest.html");

    Button back = (Button)findViewById(R.id.back);
    back.setOnClickListener(new OnClickListener() {

      @Override
      public void onClick(View v) {
        wv.goBack();
      }
    });
  }

  private class MyWebViewClient extends WebViewClient {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
      view.loadUrl(url);
      Toast.makeText(getApplicationContext(),
          "A link has been clicked! ", Toast.LENGTH_SHORT).show();
      return true;
    }
  }


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

11.18 Capteurs

Main.java

```
public class Main extends Activity {

  @Override
  protected void onStop() {
    // Surtout, ne pas oublier de détacher l'écouteur !
    manager.unregisterListener(myListener);
    super.onStop();
  }
  SensorManagerSimulator manager;
  SensorEventListener myListener;

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

    //SensorManager manager = (SensorManager)getSystemService(SENSOR_SERVICE);
    manager = SensorManagerSimulator.getSystemService(this, SENSOR_SERVICE);
    manager.connectSimulator();

    //   Création d'un écouteur qui réagit aux événements de l'accéléromètre
    myListener = new SensorEventListener() {
      public void onSensorChanged(SensorEvent event) {

        if (event.type == Sensor.TYPE_ACCELEROMETER)
        {
          float x,y,z;
          x = event.values[0];
          y = event.values[1];
          z = event.values[2];
          TextView t = (TextView)findViewById(R.id.text1);
          t.setText("x = " + x);

        }
      }
      public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // TODO Auto-generated method stub
      }
    };

    // Ajout d'un écouteur pour l'acceleromètre
    manager.registerListener(myListener
        , manager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
        , SensorManager.SENSOR_DELAY_UI);
```

```
  }
}
```

11.19 Camera

MainActivity.java

```
public class MainActivity extends Activity {
    Camera mCamera;
    Preview mPreview;

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

        mPreview = new Preview(this);
        setContentView(mPreview);

        // A faire dans un thread à part en principe
        boolean ok = safeCameraOpen(0);
        if (ok)
            mPreview.setCamera(mCamera);
    }

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

    private boolean safeCameraOpen(int id) {
        boolean qOpened = false;

        try {
            releaseCameraAndPreview();
            mCamera = Camera.open(id);
                    qOpened = (mCamera != null);
        } catch (Exception e) {
            Log.e(getString(R.string.app_name), "failed to open Camera");
            e.printStackTrace();
        }

        return qOpened;
    }

    private void releaseCameraAndPreview() {
        mPreview.setCamera(null);
        if (mCamera != null) {
            mCamera.release();
            mCamera = null;
```

```
        }
    }
}
```

Preview.java

```
package jf.andro.ac;

import java.io.IOException;
import java.util.List;

import android.content.Context;
import android.hardware.Camera;
import android.hardware.Camera.Size;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.ViewGroup;

class Preview extends ViewGroup implements SurfaceHolder.Callback {

  SurfaceView mSurfaceView;
  SurfaceHolder mHolder;
  Camera mCamera;
  List<Size> mSupportedPreviewSizes;
  Size mPreviewSize;

  Preview(Context context) {
    super(context);

    mSurfaceView = new SurfaceView(context);
    addView(mSurfaceView);

    // Install a SurfaceHolder.Callback so we get notified
    // when the
    // underlying surface is created and destroyed.
    mHolder = mSurfaceView.getHolder();
    mHolder.addCallback(this);
    mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
  }

  public void setCamera(Camera camera) {
    if (mCamera == camera) { return; }

    stopPreviewAndFreeCamera();

    mCamera = camera;

    if (mCamera != null) {
      List<Size> localSizes =
```

```
            mCamera.getParameters().getSupportedPreviewSizes();
        mSupportedPreviewSizes = localSizes;
        requestLayout();

        try {
          mCamera.setPreviewDisplay(mHolder);
        } catch (IOException e) {
          e.printStackTrace();
        }

        /*
                Important: Call startPreview() to start
                updating the preview surface. Preview must
                be started before you can take a picture.
         */
        mCamera.startPreview();
      }
    }

    /**
     * When this function returns, mCamera will be null.
     */
    private void stopPreviewAndFreeCamera() {

      if (mCamera != null) {
        /*
               Call stopPreview() to stop updating
               the preview surface.
         */
        mCamera.stopPreview();

        /*
               Important: Call release() to release the
               camera for use by other applications.
               Applications should release the camera
               immediately in onPause() (and re-open() it in
               onResume()).
         */
        mCamera.release();

        mCamera = null;
      }
    }

    @Override
    public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {
      // TODO Auto-generated method stub

    }

    @Override
    public void surfaceCreated(SurfaceHolder arg0) {
```

```
    // The Surface has been created, acquire the camera and tell it where
    // to draw.
    try {
      if (mCamera != null) {
        mCamera.setPreviewDisplay(mHolder);
      }
    } catch (IOException exception) {
      Log.e("W", "IOException caused by setPreviewDisplay()", exception);
    }
  }

  @Override
  public void surfaceDestroyed(SurfaceHolder holder) {
    // Surface will be destroyed when we return, so stop the preview.
    if (mCamera != null) {
      /*
              Call stopPreview() to stop updating the preview surface.
       */
      mCamera.stopPreview();
    }
  }

  @Override
  protected void onLayout(boolean changed,  int l, int t, int r, int b) {
    if (changed && getChildCount() > 0) {
      final View child = getChildAt(0);

      final int width = r - l;
      final int height = b - t;

      int previewWidth = width;
      int previewHeight = height;
      if (mPreviewSize != null) {
        previewWidth = mPreviewSize.width;
        previewHeight = mPreviewSize.height;
      }

      // Center the child SurfaceView within the parent.
      if (width * previewHeight > height * previewWidth) {
        final int scaledChildWidth = previewWidth * height / previewHeight;
        child.layout((width - scaledChildWidth) / 2, 0,
            (width + scaledChildWidth) / 2, height);
      } else {
        final int scaledChildHeight = previewHeight * width / previewWidth;
        child.layout(0, (height - scaledChildHeight) / 2,
            width, (height + scaledChildHeight) / 2);
      }
    }

  }

}
```

11.20 JNI

MainActivity.java

```
public class MainActivity extends Activity {

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

        TextView text = (TextView)findViewById(R.id.texte);
        text.setText("5+7 = " + NativeCodeInterface.add(5, 7));
    }

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

NativeCodeInterface.java

```
public class NativeCodeInterface {

    public static native int calcul1(int x, int y);

    public static int add(int x, int y)
    {
        int somme;
        somme = calcul1(x,y);
        return somme;
    }

    static {
        System.loadLibrary("testmodule");
    }
}
```

test.c

```
#include "andro_jf_jni_NativeCodeInterface.h"
JNIEXPORT jint JNICALL Java_andro_jf_jni_NativeCodeInterface_calcul1
  (JNIEnv * je, jclass jc, jint a, jint b)
{
   return a+b;
}
```

andro_jf_jni_NativeCodeInterface.h

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class andro_jf_jni_NativeCodeInterface */

#ifndef _Included_andro_jf_jni_NativeCodeInterface
#define _Included_andro_jf_jni_NativeCodeInterface
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     andro_jf_jni_NativeCodeInterface
 * Method:    calcul1
 * Signature: (II)I
 */
JNIEXPORT jint JNICALL Java_andro_jf_jni_NativeCodeInterface_calcul1
  (JNIEnv *, jclass, jint, jint);

#ifdef __cplusplus
}
#endif
#endif
```

11.21 Permission pour broadcast receivers

MainActivity.java

```
public class MainActivity extends Activity {

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

    // Bouton broadcast
    Button broad = (Button)findViewById(R.id.send);
    broad.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        Intent broadcast = new Intent("andro.jf.broadcast");
        broadcast.putExtra("extra", "test");
        sendBroadcast(broadcast, "andro.jf.mypermission");
      }
    });

  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
```

```
    return true;
  }

}
```

MyBroadcastReceiver.java

```
public class MyBroadcastReceiver extends BroadcastReceiver {

  @Override
  public void onReceive(Context context, Intent intent) {
    Bundle extra = intent.getExtras();
    if (extra != null)
    {
      String val = extra.getString("extra");
      Toast.makeText(context, "Broadcast message received: " + val,
          Toast.LENGTH_SHORT).show();
    }
  }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="andro.jf.ABR"
    android:versionCode="1"
    android:versionName="1.0" >

  <!-- Declaration de la permission speciale -->
  <permission
        android:name="andro.jf.mypermission"
        android:label="my_permission"
        android:protectionLevel="dangerous" >
  </permission>

  <!-- J'utilise moi-même cette permission -->
  <uses-permission android:name="andro.jf.mypermission" />

  <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

  <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
    <activity
          android:name="andro.jf.ABR.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>

    <receiver
            android:name="MyBroadcastReceiver"
            android:exported="false" >
      <intent-filter>
        <action android:name="andro.jf.broadcast" />

        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </receiver>
  </application>

</manifest>
```

.

12 Bibliographie

PA	Programmation Android, de la conception au déploiement avec le SDK Google Android, Damien Guignard, Julien Chable, Emmanuel Robles, Eyrolles, 2009.
DA	Decompiling Android, Godfrey Nolan, Apress, 2012.
Cookbook	Android Cookbook, Ian F. Darwin, O'Reilly Media, decembre 2011.
AAS	Android Apps Security, Sheran Gunasekera, Apress, septembre 2012.
ASI	*Android Security Internals: An In-Depth Guide to Android's Security Architecture <http://www.nostarch.com/androidsecurity>*, Nikolay Elenkov, 2014.
Dalvik	Article Wikipedia, from Wikipedia, the free encyclopedia.
E-mail	Email sending in Android, Oleg Mazurashu, The Developer's Info, October 22, 2009.
SO	Android permissions: Phone Calls: read phone state and identity, stackoverflow.
SD	Utiliser les services sous Android, Nicolas Druet, Developpez.com, Février 2010.
API-Service	Service API changes starting with Android 2.0, Dianne Hackborn, 11 Février 2010.
XML	Working with XML on Android, Michael Galpin, IBM, 23 Jun 2009.
VL	How to Position Views Properly in Layouts, jwei512, Think Android, Janvier 2010.
BAS	Basics of Android : Part III – Android Services, Android Competency Center, Janvier 2009.
MARA	Implementing Remote Interface Using AIDL, Marko Gargenta, novembre 2009.
AT	Multithreading For Performance, Gilles Debunne, 19 juillet 2010.
HWM	MapView Tutorial, 20 aout 2012.
JSONREST	How-to: Android as a RESTful Client, Cansin, 11 janvier 2009.
JNI	Tutorial: Android JNI, Edwards Research Group, CC-BY-SA, Avril 2012.
Version	Historique des versions d'Android, from Wikipedia, the free encyclopedia.
BAESystems	The Butterfly Effect of a Boundary Check, Sergei Shevchenko, 22 janvier 2012.
SELinux	Security Enhanced (SE) Android: Bringing Flexible MAC to Android, S. Smalley and R. Craig, in 20th Annual Network & Distributed System Security Symposium, San Diego, California, USA, 2013.
SECustom	Android* Security Customization with SEAndroid, Liang Zhang, Intel, March 2014.
II	Les IntentService, Florian FournierProfil Pro, Developpez.com, Décembre 2014.
LLB	Life Before Loaders, Alex Lockwood, Juillet 2012.
ON	Tab Layout, Android developers.
PT	Processes and Threads, Android developers.
DBSQL	Data storage: Using Databases, Android developers, Novembre 2010.
SS	Sensor Simulator.
Emulator	Android Emulator.
ATK	AsyncTask.
JQM	Getting Started with jQuery Mobile.
UF	<uses-feature>.
CAM	Controlling the Camera.
SIGN	Signing Your Applications.
Storage	Storage Options.
Signing	Signing your applications.
LSAS	[android-developers] Launching a Service at the startup, Archives googlegroups, R Ravichandran and Dianne Hackborn, Juillet 2009.
KEK	Comment on fait un jeu pour smartphone ?, Kek, 2011.

ZZJN12 Detecting repackaged smartphone applications in third-party android market- places, W. Zhou, Y. Zhou, X. Jiang, and P. Ning, in Second ACM conference on Data and Application Security and Privacy, E. Bertino and R. S. Sandhu, Eds. San Antonio, TX, USA: ACM Press, Feb. 2012, pp. 317–326.

QC13 Mobile Security: A Look Ahead, Q. Li and G. Clark, Security & Privacy, IEEE, vol. 11, no. 1, pp. 78–81, 2013.