Développement Android

J.-F. Couchot

17 janvier 2014

Table des matières

Chapitre 1

Introduction à l'OS embarqué Android

1.1 Composants principaux d'une application

Une application Android est construite à partir de quatre catégories de composants : les activités, les services, les intentions et les fournisseurs de contenu.

1.1.1 Une activité

C'est la brique de base. Par défaut une application est une activité. C'est un morceau de code qui s'exécute à la demande de l'utilisateur, de l'OS,... et qui peut être tué par l'utilisateur ou l'OS pour préserver la mémoire du téléphone. Une activité peut interagir

— avec l'utilisateur en lui demandant des données
— avec d'autres activités ou services en émettant des intentions ou fournissant du contenu (voir ci dessous)

1.1.2 Un service

C'est l'analogue des services ou démons qui s'exécutent sur un OS classique. C'est un morceau de code qui s'exécute habituellement en arrière-plan entre le moment où il est lancé jusqu'à l'arrêt du mobile.

1.1.3 Une Intention

Une intention (intent en anglais) est un message contenant des données émis par Android (l'OS, une autre application, un autre service) pour prévenir les applications s'exécutant de la survenue d'un événement : déplacement du GPS, réception d'un SMS...

1.1.4 Un fournisseur de contenus

Une application Android répond normalement à un besoin et fournit donc certaines fonctionnalités. On appelle cela un fournisseur de contenus. Une application doit normalement se déclarer comme fournisseur de tel ou tel contenu (lorsqu'ils sont identifiés). Une autre application qui nécessiterait à un moment ce contenu émet une requête auprès de l'OS pour l'obtenir. L'OS lance alors directement l'application déclarée comme fournisseur.

1.2 Organisation d'un projet Android

1.2.1 Arborescence

Un projet Android développé via l'ADT possède peu ou prou l'arborescence de fichiers et dossiers suivante :

— Le dossier *assets* (un bien) qui contient les paquetages externes utilisés par l'application développée : les fontes, les fichiers jar, etc...
— Le dossier *bin* qui contient toutes les classes compilées (.class) ainsi qu'une archive exécutable du projet (.apk)
— Le dossier *gen* qui contient les fichiers générés par l'ADT, notamment R.java
— Le dossier *res* qui contient les ressources statiques du projet, notamment :
  — les dossiers *drawable* : on y met les images du projet ;
  — le dossier *layout* : contient les descriptions XML de l'interface graphique ;

— le dossier *menu* : contient les descriptions XML des menus ;
— le dossier *values* : qui spécifie les chaînes de caractères (strings.xml), les styles...
Noter que chaque dossier ressource peut être particularisé en fonction du support d'exécution cible (voir http ://developer.android.com/guide/topics/resources/providing-resources.html).

— Le dossier *src* (raccourci de « source »). C'est là qu'est placé l'ensemble des fichiers source Java de l'application. Pratiquement tout le travail effectué lors de la création d'une application Android est faite dans ce dossier ou bien dans le dossier "res"
— Le dossier *doc*. Raccourci pour documentation. C'est là que seront mises les documentation relatives au projet.
— Le fichier XML *AndroidManifest.xml* Ce fichier est généré par l'ADT lorsqu'on créé un nouveau projet Android. Ce fichier définit les fondamentaux de l'application : quelle est l'activité principale, quelles sont les permissions requises sur l'OS pour que l'application puisse s'exécuter,...

1.2.2 Travaux Pratiques : Hello World

Suivre le TP `Hello World`.

1.2.3 Exercice

1. Faire en sorte que lorsqu'on passe en mode paysage (dans l'émulateur `Ctrl+F11`) l'application `Hello World` affiche en plus *Landscreen*.
2. Comment faire en sorte que l'icône de lancement de l'application `Hello World` soit le même quelle que soit la résolution du mobile ? Modifier en conséquence le dossier `res`.
3. Comment faire en sorte que l'icône de lancement de l'application `Hello World` soit le fichier icon2.jpg lorsque l'affichage est en mode paysage ?
4. Comment faire en sorte que l'application affiche les messages en français si la langue du téléphone est le français et des messages en anglais sinon ?
5. Dans le code suivant extrait du fichier `manifest.xml` expliquer
   (a) la ligne 1
   (b) les ligne 2 à 4

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

Chapitre 2

Les interfaces utilisateur

On commence par créer une interface utilisateur graphique. La gestion des événements liés à celle-ci sera entrevue en TP et approfondie plus tard.

2.1 Rendu visuel

Dans une application Android, l'interface utilisateur est construite à l'aide d'objets de type `View` et `ViewGroup`. Les objets de type `View` sont les unités de base, i.e. des composants gérant les interactions avec l'utilisateur. Les objets de type `ViewGroup` organisent la manière dont sont présents les composants. Un `ViewGroup` est classiquement défini à l'aide d'un `Layout`.

2.2 Interface : programmation Java ou déclaration XML

La construction d'interface peut se faire selon deux approches complémentaires : la première est programmatique (Java) et la seconde est une déclaration à l'aide d'un fichier XML.

2.2.1 Programme Java

Dans la version programmatique, un objet `ViewGroup` est instancié dans le code Java (à l'aide du mot-clé `new`) comme un `Layout` particulier (ligne 4). Puis chaque composant est instancié (lignes 4 et 6) et ajouté à ce `ViewGroup` via la méthode `addView` (ligne 9 et 10) et enfin le `ViewGroup` est attaché à l'activité via `setContentView`.

```
public class Program extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
         LinearLayout llayout =  new LinearLayout(this);
         llayout.setOrientation(LinearLayout.VERTICAL);
         TextView nlbl = new TextView(this);
         nlbl.setText("Nom");
         EditText nedit = new EditText(this);
         llayout.addView(nlbl);
         llayout.addView(nedit);
         setContentView(llayout);
    }
}
```

On se passe complètement du fichier XML de description de l'organisation (dossier `res/layout`) dans une telle approche programmatique.

2.2.2 Déclaration XML

La philosophie est autre ici : l'organisation visuelle de l'activité est statique et définie une fois pour toute dans un fichier XML du dossier `res/layout`. Celui donné ci dessous aura la même interprétation visuelle que le programme donné avant.

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

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
  <TextView
   android:id="@+id/txtvname"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/name" />
  <EditText
   android:id="@+id/edtname"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>
</LinearLayout>
```

Il reste uniquement à attacher cette organisation à l'activité via `setContentView` (ligne 4).

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

2.3 Les principaux composants graphiques

2.3.1 Exercice

On considère une activité dont la vue est liée au layout défini par les fichiers `res/layout/main.xml` et `res/values/arrays.xml` ci-dessous. Dire quels éléments vont être affichés, comment ...

```
<?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" >
  <CheckBox android:id="@+id/cbxYN"
    android:layout_width="20dp"
    android:layout_height="20dp"
    android:checked="false" />
  <RadioGroup android:id="@+id/rgGroup1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/RB1" android:text="Button1" />
    <RadioButton android:id="@+id/RB2" android:text="Button2" />
    <RadioButton android:id="@+id/RB3" android:text="Button3" />
  </RadioGroup>
  <Spinner android:id="@+id/spnSaisons"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:entries="@array/saisons"/>
  <DatePicker android:id="@+id/dPdate"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
  <ImageView android:id="@+id/imgIcon"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity = "center"
    android:src="@drawable/icon" />
</LinearLayout>

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <array name="saisons">
        <item>printemps</item>
        <item>ete</item>
        <item>automne</item>
        <item>hiver</item>
    </array>
</resources>
```

2.4 Les Layouts

Les Layouts organisent le positionnement des composants graphiques dans l'interface utilisateur. On énonce les principales ci-dessous. On pourra se référer à http://developer.android.com/guide/topics/ui/declaring-layout.html.

2.4.1 LinearLayout

Cette organisation aligne les composants dans une seule direction : verticalement ou horizontalement (en fonction de la valeur l'attribut `android:orientation`). Gère l'alignement (`android:gravity`) du composant. Voir http://developer.android.com/guide/topics/ui/layout/linear.html.

2.4.2 RelativeLayout

Permet de déclarer des postions relativement par rapport au parent ou par rapport à d'autres composants. Voir http://developer.android.com/guide/topics/ui/layout/relative.html.

2.4.3 FrameLayout

On empile les composant les uns sur les autres. Chacun est positionné dans le coin en haut à gauche en masquant plus ou moins celui du dessous sauf en cas de composant transparent.

2.4.4 TableLayout

Cette organisation agence les composants selon un quadrillage (comme avec l'élément `<table>` en HTML). Il utilise pour cela l'élément `<tableRow>` qui déclare une nouvelle ligne. Les cellules sont définies par les composants qu'on ajoute aux lignes.

2.4.5 Travaux Pratiques

Réaliser le TP intitulé « Les menus de l'utilisateur : Lanceur, menus et sudoku ».

2.5 Les menus des activités

On se restreint dans une première partie aux menus que l'on peut lancer à partir du bouton menu du téléphone, soit les version d'android 2.3.X. A partir de la version 3.0, ce bouton n'étant plus nécessairement présent, on doit implanter une *barre d'action* à la place.

2.5.1 Les menus sous la version 2.3.X

Le modèle de construction est basé sur le patron MVC :

1. le menu est défini statiquement en XML et gonflé via un inflater,
2. son contrôle se fait en gérant un événement et
3. le modèle est modifié en accédant à l'unique instance de celui-ci de manière statique.

2.5.1.1 Définition statique des menus

Les menus sont enregistrés dans un fichier xml du dossier `/res/menu` du projet. Ils contiennent éventuellement des sous-menus. Ci dessous on donne le fichier `/res/menu/sudoku_menu.xml` qui définit deux menus et un sous menu.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_param"
          android:icon="@android:drawable/ic_menu_preferences"
          android:title="@string/libelle_param">
      <menu>
        <group android:checkableBehavior="single">
          <item android:id="@+id/menu_n1"
```

FIGURE 2.1 – Barre d'action

```
            android:title="@string/libelle_niv_1" />
          <item android:id="@+id/menu_n2"
            android:title="@string/libelle_niv_2" />
          <item android:id="@+id/menu_n3"
            android:title="@string/libelle_niv_3" />
        </group>
      </menu>
    </item>
    <item android:id="@+id/menu_quitter"
          android:icon="@android:drawable/ic_menu_close_clear_cancel"
          android:title="@string/libelle_quitter" />
</menu>
```

2.5.1.2 Exercices

1. Que sont `@android:drawable/ic_menu_close_clear_cancel` et `@android:drawable/ic_menu_p`
2. Que devrait-on faire pour utiliser des icones personnalisés ?
3. A quoi servent les lignes `<menu>...</menu>` ?

2.5.1.3 Gonflage du menus défini en XML

Lors de l'appuie sur le bouton menu du telephone, la méthode `onCreateOptionsMenu(Menu menu)` est invoquée dans l'objectif d'associer un menu xml à l'objet `menu` passé en paramètre.

Comme le `LayoutInflater` gonflait les layouts défini en xml, le `MenuInflater` gonfle les menus définis en xml. On effectue l'association comme suit :

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

2.5.1.4 Gestion des événements liés aux menu

La méthode `onOptionsItemSelected(MenuItem item)` de l'activité où est crée le menu est invoqué lorsqu'un élément est sélectionné dans le menu apparu sur le téléphone. Il suffit de donner un code traitant tous les items du menu (et des éventuels sous-menus) comme suit :

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
  // Handle item selection
  switch (item.getItemId()) {
    case R.id.menu_n1:
    ...
    return true;
    case R.id.menu_quitter:
    this.finish();
    return true;
  }
}
```

2.5.1.5 Exercice

Compléter le code pour qu'on puisse choisir entre trois niveaux et que cela charge trois grilles différentes correspondantes aux niveaux.

2.5.2 Une barre d'actions

Une barre d'action peut être incluse dans toute application à partir de Android 3.0 (API numéro 11).

2.5.2.1 Mise en place

Le manifest doit préciser que l'application visée ou la version minimum du sdk minSdkVersion, `targetSdkVersion` et `minSdkVersion` est fixée à 11 ou plus.

```
<manifest ... >
    <uses-sdk android:minSdkVersion="4"
              android:targetSdkVersion="11" />
    ...
</manifest>
```

Dans l'exemple précédent, on a une application qui est telle que chacune des ses activités embarque la barre par défaut donnée à la figure ci-dessus.

lorsqu'elle est exécutée sur un Android 3.0 au moins. Dans une exécution sur un Android 2.3.X ou moins, la barre d'action est fixée en haut de l'écran.

2.5.2.2 Masquer la barre

Pour supprimer statiquement la barre d'action dans une activité particulière, on modifie le manifest comme suit :

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

Pourle faire dynamiquement, on fait comme suit :

```
ActionBar actionBar = getActionBar();
// actionBar.hide();
// actionBar.show();
```

2.5.3 Particulariser la barre d'action (ajouter un menu)

Pour intégrer un menu personnel dans une barre d'action, on déclare que le menu est un item d'action de la barre. Pour cela il suffit de définir un menu en xml et de le gonfler comme à la section « Gonflage du menus défini en XML ».

2.5.3.1 Exercice

Compléter le code pour qu'on puisse choisir entre trois niveaux et que cela charge trois grilles différentes correspondantes aux niveaux.

Chapitre 3

Les Fragments

3.1 Introduction

Les fragment ont été introduits à la version 3.0 (level 11) pour faciliter la modularité dans le développement d'applications pour des supports très différents (petits téléphones, grandes tablettes). Intuitivement les gabarits (layout) sont découpés en blocs nommés fragments. L'apparence de l'activité peut être modifiée à la volée, adaptée au support...

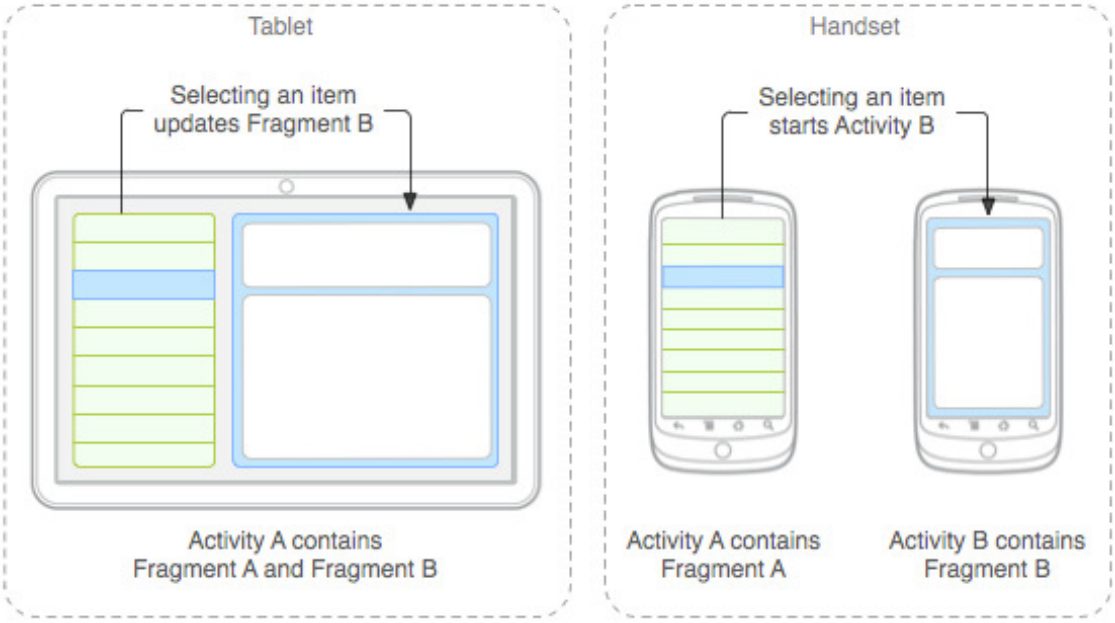

FIGURE 3.1 – Bénéfices de l'utilisation de fragments

A la figure 3.1,
— le fragment A définit la présentation d'une liste d'articles (dans un gabarit particulier) ;
— le fragment B définit la présentation d'un article (dans un gabarit particulier aussi).

Ces deux fragments
— sont combinés dans une seule activité pour l'affichage sur une tablette, mais
— sont embarqués dans deux activités différentes pour un écran de petite taille.

Dans une application basée sur ce principe, on a
— des objets qui héritent de `FragmentActivity` : ce sont des activités qui contiendront des fragments ;
— des objets qui héritent de `Fragment` : ils seront utilisés dans les FragmentActivity ;
— des gabarits (layout) pour dire comment sont représentés les fragments et les fragmentActivity

3.2 Détails d'une application Master/Detail

En créant un nouveau projet de type Master/Detail dans l'assistant, où :

— l'objet représenté à droite est nommé JeuDetail et
— la liste à gauche est nommée JeuList,
on a le diagramme de classes donné à la figure 3.2.

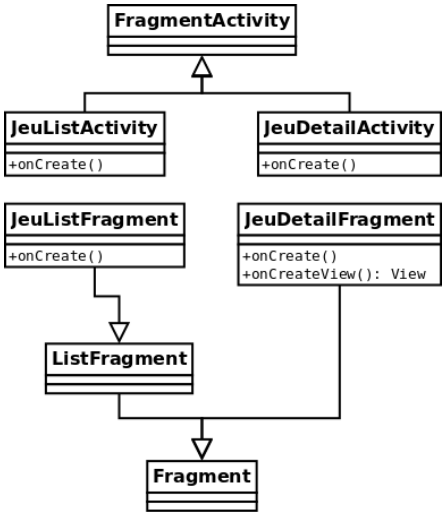

FIGURE 3.2 – Diagramme des classes particulières d'un projet Master/Detail.

L'activité exécutable est `JeuListActivity`. Le gabarit qui permet son affichage est :
— `activity_jeu_list.xml` sur un mobile, ou bien
— `activity_jeu_twopane.xml` si l'écran est large. En effet, un lien vers le gabarit `activity_jeu_twopane.xml` est défini dans la ressource `values-large` qui est chargée pour les écrans larges :

```
<item name activity_jeu_list" type="layout">@layout/activity_jeu_twopane+</item>
```

3.2.1 Cas d'un écran large

Un extrait du gabarit `activity_jeu_twopane.xml` est donné à la figure 3.3. C'est un `LinearLayout` horizontal qui contient
— le fragment `com.example.android_awale_f.JeuListFragment` ;
— le framelayout `jeu_detail_container`.

```
<LinearLayout ...
    android:orientation="horizontal"...>
    <fragment ...
        android:id="@+id/jeu_list" ...
        android:name="com.example.android_awale_f.JeuListFragment"
        tools:layout="@android:layout/list_content" .../>
    <FrameLayout
        android:id="@+id/jeu_detail_container".../>
</LinearLayout>
```

FIGURE 3.3 – Extrait du gabarit `activity_jeu_twopane`.

Le fragment est défini par la classe `JeuListFragment`. Elle hérite de `ListFragment` qui est prévue pour afficher des données sous la forme d'une liste.

L'activité principale `JeuListActivity` demande le remplacement du framelayout lorsqu'un item est sélectionné dans le fragment de gauche comme montré à la figure 3.4. Plus précisement, dans cette méthode on instancie le fragment, on lui donne comme argument le numéro de l'item cliqué puis on remplace le layout `jeu_detail_container` par ledit fragment.

Un fragment n'est dessiné que lors de l'invocation de la méthode `oncreateView()` détaillé à la figure 3.5. Celle ci retourne une vue gonflée dans laquelle un texte a été fixé. Le gabarit `fragment_jeu_detail.xml` ne contient qu'un TextView identifié par `jeu_detail`.

```
public void onItemSelected(String id) {
  if (mTwoPane) {
    JeuDetailFragment fragment = new JeuDetailFragment();

    Bundle arguments = new Bundle();
    arguments.putString(JeuDetailFragment.ARG_ITEM_ID, id);
    fragment.setArguments(arguments);

    getSupportFragmentManager().beginTransaction().
                                  replace(R.id.jeu_detail_container,
                                          fragment).commit();
  }else{...}
}
```

FIGURE 3.4 – Extrait de la méthode de gestion de clic sur item de `JeuListActivity`.

```
public View onCreateView(LayoutInflater inflater,
                         ViewGroup container, Bundle savedInstanceState) {
  View rootView = inflater.inflate(R.layout.fragment_jeu_detail,
                                   container, false);
  ((TextView) rootView.findViewById(R.id.jeu_detail)).setText(mItem.content);
  return rootView;
```

FIGURE 3.5 – Création de la vue du fragment `JeuDetailFragment`.

3.2.2 Cas d'un mobile

L'application exécute dans ce cas les deux activités `JeuListActivity` et `JeuDetailActivity`.

```
<fragment ...
    android:id="@+id/jeu_list"
    android:name="com.example.android_awale_f.JeuListFragment".../>
```

FIGURE 3.6 – extrait du gabarit `activity_jeu_list`.

La première (`JeuListActivity`) charge le gabarit `activity_jeu_list.xml`, qui est un élément fragment dont un extrait est donné à la figure 3.6. Comme dans le cas d'un écran large, l'objet `JeuListFragment` est chargé par le gabarit. Ici, lorsqu'un item est sélectionné dans le fragment, l'activité demande le lancement de la seconde activité `JeuDetailActivity` comme montré à la figure 3.7

Lorsqu'elle est lancée, l'activité `JeuDetailActivity` charge le gabarit `activity_jeu_detail.xml` comme détaillé à la figure 3.8. Ce gabarit est un FrameLayout vide identifié par `jeu_detail_container`. On lui ajoute un `JeuDetailFragment`.

3.3 Travaux Pratiques

1. Faire générer les classe ci-dessus par l'assistant. Exécuter le code sur un émulateur de mobile et un émulateur de tablette.
2. Reprendre le projet Awale et le convertir en une application basée sur les fragments et le modèle Master/Detail.

```
  public void onItemSelected(String id) {
    if (mTwoPane) {...}
    }else{
      Intent detailIntent = new Intent(this, JeuDetailActivity.class); ...
      startActivity(detailIntent);
    }
  }
```

FIGURE 3.7 – Extrait de la méthode de gestion de clic sur item de `JeuListActivity`.

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_jeu_detail);
  ...
  JeuDetailFragment fragment = new JeuDetailFragment();
  Bundle arguments = new Bundle();
  arguments.putString(JeuDetailFragment.ARG_ITEM_ID, getIntent()
                      .getStringExtra(JeuDetailFragment.ARG_ITEM_ID));

  fragment.setArguments(arguments);
  getSupportFragmentManager().beginTransaction()
                                        .add(R.id.jeu_detail_container,
                                             fragment).commit();...
  }
```

FIGURE 3.8 – Extrait de la méthode `onCreate` de `JeuDetailActivity`.

Chapitre 4

Cycle de vie d'une activité

4.1 Méthodes du cylce de vie

La figure 3.1 résume le cycle de vie d'une activité en montrant quelles méthodes sont appelées lors des événement qu'elle peut rencontrer.

1. `onCreate` : Cette méthode est tout d'abord invoquée lorsque l'activité est créee : C'est là qu'on associe la vue, initialise ou récupère les données persistantes... La méthode `onCreate` reçoit toujours en paramètreun objet `Bundle` qui contient l'état dans lequel était l'activité avant l'invocation.
2. `onStart` : Cette méthode est invoquée avant que l'activité soit visible et Une fois que l'exécution de cette méthode est finie,
   — si l'activité apparaît en premier plan, c'est `onResume` qui est invoquée
   — si l'activité ne peut apparaître en premier plan, c'est `onStop` qui est invoquée
3. `onResume` : Cette méthode est appelée immédiatement avant que l'activité ne passe en Elle est appelée soit parce qu'elle vient d'être (re)lancée, par exemple lorsqu'une autre activité a pris le devant puis a été fermée, remettant votre activité en premier plan.

   C'est souvent l'endroit où l'on reconstruit les interfaces en fonction de ce qui s'est passé depuis que l'utilisateur l'a vue pour la dernière fois.
4. `onPause` : Lorsque l'utilisateur est détourné de votre acivité, en la passant en second plan, c'est cette méthode qui est invoquée avant de pouvoir afficher la nouvelle activité en premier plan. Une fois cette méthode exécutée, Android peut tuer à tout moment l'activité sans vous redonner le contrôle. C'est donc là qu'on arrête proprement les services, threads,... et que l'on sauvegarde les données utiles à l'activité lorsqu'elle redémarera.
5. `onStop` : Cette méthode est invoquée lorsque l'activité n'est plus visible soit par ce qu'une autre est passée en premier plan, soit parce qu'elle est en cours de destruction.

4.2 Exercice

Réaliser le TP intitulé « Cycle de vie d'une activité Android : Vie, pause et mort ».

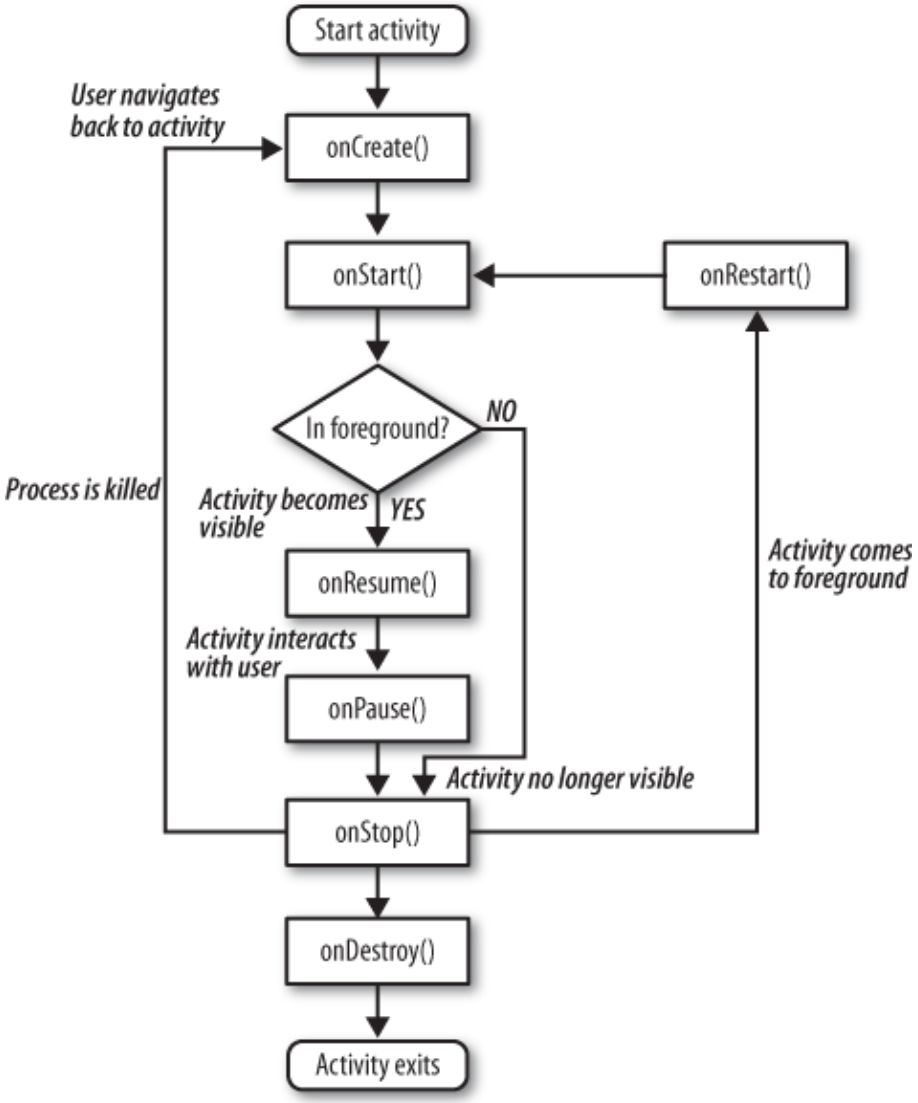

FIGURE 4.1 – Cycle de vie d'une activité android

Chapitre 5

Sauvegarder des données

Dans ce chapitre on voit comment sauvegarder des données d'une application android : une préférence (donnée courte), un fichier (donnée plus volumineuse), un fichier JSON, une base de données, le cloud.

5.1 Les préférences

Android permet de mémoriser une information sous la forme d'une paire *(clef,valeur)* et d'y accéder lors de l'exécution de l'activité ou du service. La valeur est de type primaire (booléen,flottant, entier, long ou chaîne de caractères). Destinée essentiellement pour sauvegarder les préférences de l'application, la méthode permet en fait de stocker toute information pouvant se mettre sous la forme (clef,valeur).

5.1.1 Méthodes

On accède aux préférences partagées avec la méthode
`getSharedPreferences(String nomFichier, int typeAcces)`
où

- `typeAcces` est le type d'accès au fichier :
  - `MODE_PRIVATE` pour un accès interne à l'application,
  - `MODE_WORLD_READABLE` et
  - `MODE_WORLD_WRITEABLE` pour des accès universels en lecture et en écriture
- `nomFichier` est le nom du fichier à lire. S'il n'existe pas, il sera crée lorsqu'on écrira des données dedans

Notez que pour un nom de fichier donné, il n'y a qu'une seule instance d'objet correspondant à ce fichier. Cette instance est partagée par toutes les applications éventuellement.

Sur un objet de type `SharedPreferences`, on invoque la méthode `edit()` pour obtenir un flux en écriture. La méthode `putBoolean(String clef, boolean valeur)`, permet d'associer à une clef identifiée par une chaîne une valeur sous la forme d'un booléen. On a aussi `putString`, `putFloat`, `putInt` et `putLong` de même signature et au comportement évident. Les mises à jour ne sont réellement effectuées qu'après invocation de la méthode `commit()` sur le flux.

Pour récupérer les données enregistrées, sur un objet de type `SharedPreferences`, on invoque la méthode `getBoolean(String clef, boolean vd)` qui retourne la valeur booléenne associée à la clef si elle existe ou la valeur par défaut `vd` si elle n'existe pas. On a aussi, `getFloat(String clef, float vd)`, `getInt(String cl` `getLong(String clef, boolean vd)` pour les valeurs numériques et `getString(String clef, String vd` pour récupérer une variable sous la forme d'une chaîne de caractères.

5.1.2 Exercices

1. Dans quelle méthode de l'activité principale devrait se faire la sauvegarde des préférences ?
2. Dans quelle méthode de l'activité principale devrait se faire une récupération des paramètres utilisateurs ?

5.1.3 Travaux Pratiques

1. Creer une application ne comportant qu'un champ de texte éditable initialement vide et un bouton "enregistrer".

2. Lorsque l'utilisateur modifie le champ textuel, appuie sur le bouton "enregistrer" ou ferme l'application, le contenu de ce champs est mémorisé. Au chargement suivant de l'application, le contenu du champ est restauré.

5.2 Les fichiers

En plus de supporter les classes usuelles Java d'entrée/sortie la bibliothèque Android fournit les méthodes simplifiées `openFileInput` et `openFileOuput` permettant de lire et d'écrire aisément dans des fichiers. L'exemple suivant résume la situation :

```
try{
  FileOutputStream fos = openFileOutput("tempfile.tmp", MODE_PRIVATE);
  fos.write(new String("Hello World").getBytes());
  fos.close();

  String s = "";
  FileInputStream fis = openFileInput("tempfile.tmp");
  int b = fis.read();
  while (b != -1){
    s += new Character((char)b);
    b = fis.read();
  }
  fis.close();
  Toast.makeText (this,s,Toast.LENGTH_SHORT).show();
}catch(Exception e){
//
}
```

On note les contraintes suivantes :

— On ne peut créer un fichier qu'à l'intérieur du dossier de l'application. Préciser un séparateur de dossier ("\") dans la méthode `openFileOutput` engendre en effet une exception.

— Par contre d'autres applications peuvent éventuellement le lire et le modifier. Si l'on souhaite ceci, la permission à déclarer à la création sera `MODE_WORLD_READABLE` et `MODE_WORLD_WRITEABLE` respectivement.

— Si le fichier `tempfile.tmp` n'existe pas, l'appel à la méthode `openFileOutput` le crée.

5.3 Le cloud

Cette partie est largement inspirée des pages 214 à 224 du livre "Android, guide de développements d'applications pour Smartphones et Tablettes" de Sébastien PÉROCHON aux éditions ENI, juillet 2011.

Depuis la version 2.2 (API 8) Android permet de sauvegarder les données de l'application sur un serveur de Google. Pour un utilisateur ayant plusieurs appareils, la même application a ses données cohérentes sur tous ces appareils grâce à un processus de synchronisation en arrière plan avec les serveurs de Google.

Pour une utilisation en dehors de l'émulateur les données sont associées au compte Google renseigné dans l'appareil. L'utilisateur doit avoir un compte Google identique sur tous ses appareils pour bénéficier de ce service.

Le principe général est le suivant : l'application communique avec le gestionnaire de sauvegarde des données persistantes sur le serveur. C'est ce dernier qui se charge d'envoyer les données dans le nuage et de les récupérer au lancement de l'application, par exemple.

5.3.1 Hériter de la classe `BackupAgentHelper`

La méthode la plus simple consiste à exploiter la classe `BackupAgentHelper` qui fournit une interface minimale pour synchroniser les données avec un serveur du cloud.

```
public class MonAgentDeSauvegarde extends BackupAgentHelper{
  public void onCreate(){
    SharedPreferencesBackupHelper assistantFichierPrefs = new SharedPreferencesBackupHelper(this,Sudoku.PREFS_NAME);
    addHelper("clefPourAgent",assistantFichierPrefs);
  }
}
```

Dans le code précédent, on a construit un agent de sauvegarde, qui hérite de la classe `BackupAgentHelper`. À la création, on comence par définir quel(s) fichier(s) doit(doivent) être synchronisés dans le cloud. Ici c'est un fichier de préférénce. La démarche est semblable pour un fichier quelconque. Ensuite, on renseigne que c'est cet assistant qui va être utilisé pour la synchronisation.

5.3.2 Modifier le manifest pour toute l'application

Pour bénéficier de ce service Google, le développeur doit tout d'abord réserver un espace dans le cloud. Cela se fait via la page suivante où doit être renseigné le nom du package.

```
http://code.google.com/intl/fr/android/backup/signup.html
```

Une fois enregistré, le développeur reçoit une clef, à insérer dans l'élément application du manifest comme suit :

```
<application>
  ...
  <meta-data android:name="com.google.android.backup.api_key"
          android:value="your_backup_service_key" />
</application>
```

Enfin, l'élément application doit définir l'attribut `android:backupAgent` pour préciser quelle est la classe qui assure la gestion de la persistance des données avec le serveur de Google.

```
<application android:icon="@drawable/icon"
             android:label="@string/app_name"
             android:backupAgent=".MonAgentDeSauvegarde">
```

5.3.3 La demande de synchronisation

Pour faire une demande de sauvegarde, l'application doit invoquer un objet de type `BackupManager` dans toute activité necéssitant la sauvegarde des préférences.

```
BackupManager backupManager = new BackupManager(this);
```

Pour lui dire que les données ont changé, on insère le code suivant chaque fois que les données doivent être modifiées.

```
this.backupManager.dataChanged();
```

5.3.4 Évaluer le développement

On peut simuler une sénario d'enregistrement comme suit :

1. Installer l'application.
   — Si on installe l'application sur un support physique, vérifier que la sauvegarde/restauration est activée.
   — Si on l'installe dans l'émulateur, on active le gestionnaire de backup dans l'émulateur :
   ```
   adb shell bmgr enable true
   ```
2. Modifier la valeur de la variable sauvegardée dans la préférence partagée et dans le cloud.
3. Si on utilise l'émulateur, pour forcer ce dernier à efffectuer les backup dans sa queue
   ```
   adb shell bmgr run
   ```
4. Désinstaller et réinstaller l'application. Les données sauvegardées à l'étape 2 dans le nuage sont restaurées dans l'application.

5.4 Une base de données SQLite

Suivre le cours page http://www.androidhive.info/2011/11/android-sqlite-database-tutorial

Chapitre 6

Le Google Cloud Messaging

Le Google Cloud Messaging (GCM) est un service qui permet aux utilisateurs d'une application Android de ne recevoir des informations la concernant que si celles-ci sont nouvelles. Cette bibliothèque permet donc d'implanter à moindre coût un "push" d'un serveur vers l'application. Cette information peut être n'importe quelles données jusqu'à 4kb.

6.1 Une architecture en trois parties

L'architecture d'un tel développement est basée sur trois entités illustrée à la figure ci-dessous.

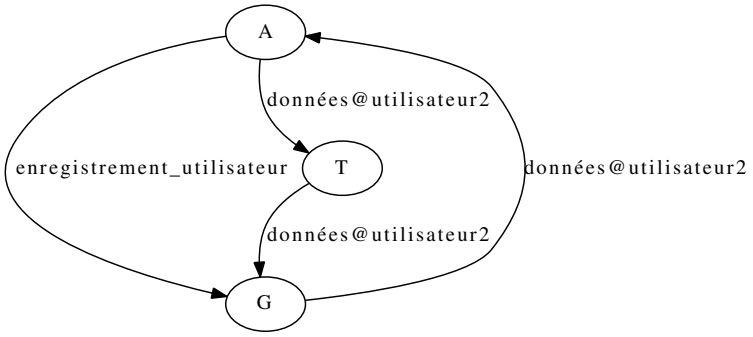

FIGURE 6.1 – Architecture simplifiée GCM

— Le serveur tiers (T). A minima, il envoie au serveur GCM les données pour un utilisateur identifié d'une application.
— Le serveur GCM (G). Il relaie les données envoyées par un serveur tiers aux mobiles possédés par le destinataire identifié.
— L'application (A). A minima elle
  — s'enregistre une fois auprès de G puis,
  — par la suite, traite les informations reçues par le serveur GCM lorsque celles-ci arrivent ;
  — éventuellement, l'application peut aussi envoyer aussi au serveur tiers des données à destination d'un utilisateur identifié.

6.2 Les contraintes technologiques des trois parties

— Le serveur tiers (T). On doit connaître son adresse IP fixe ou son URL.
— Le serveur GCM (G). Pour pouvoir utiliser ce service, il faut
  — disposer d'un identifiant Google du responsable du développement ;
  — disposer d'un nom du paquetage pour l'application ;
  — disposer de l'IP ou de l'URL de (T).
— L'application (A). Elle s'exécute sur une version d'Android supérieur ou égal à 8 (2.2), le Play Store doit être installé et requiert un compte Google configuré. Pour l'évaluation, elle doit s'exécuter sur un émulateur Google APIs supérieur ou égal à 9 (2.2) et requiert un compte Google configuré dans l'émulateur.

6.3 Flux d'information entre les éléments

Les trois étapes clés du projet sont :

1. Activation de la fonctionnalité GCM dans l'application A : l'application enregistre le mobile identifié par le compte Google. Ceci étant fait, le mobile enregistré peut recevoir tous les messages destinés à son propriétaire.
2. L'envoi de message depuis le serveur T : Le serveur exécute un POST au serveur G (`https://android.googleapis.com/gcm/send`). Ce message est composé d'une entête et d'un corps. La première contient notamment l'`API_KEY` qui est la clef du serveur T l'autorisant à communiquer avec G. Le corps doit contenir la liste des destinataires (`registration_id`) et le message en lui même.
3. A réception d'un message sur le mobile : Si le type du message reçu est "GCM" et si sa clef est celle déclarée par l'application à l'enregistrement, alors l'application capture le message et effectue les traitements adaptés.

6.4 Configuration du serveur GCM G.

6.4.1 `SENDER_ID` et `PROJECT_ID`

Sur le cite `https://code.google.com/apis/console`, créer un nouveau projet et récupérer l'identifiant dans l'URL. Dans l'URL `https://code.google.com/apis/console/#project:59876767155` l'identifiant du projet est `59876767155`. Cet identifiant est ensuite référencé comme `SENDER_ID`. Repérer aussi sur la page d'accueil la chaîne stockée en `PROJECT_ID`.

Pour initialiser ensuite le service GCM, dans l'onglet *Services*, activer *Google Cloud Messaging for Android* et accepter les termes de celui-ci.

6.4.2 Obtenir une `API_Key` pour le serveur T.

Toujours sur le même site, choisir l'onglet *API Access* et générer une clef soit pour une adresse IP publique, soit pour un serveur identifié par un nom de domaine. Ceci se fait respectivement en choisissant une clef pour un serveur (*Create Server Key*) ou pour un navigateur (*Create Browser Key*).

6.5 Développer le code du serveur T.

On donne ci-dessous un exemple de code s'exécutant sur T et demandant au serveur G de transmettre un message à un destinataire.

```
<?php

$apiKey = "La_clef_associee_au_serveur_qui_execute_ce_scripte";

$messageData =  array('message' => "le message a envoyer",
                      'unAutreMessage' => "un autre message a envoyer");

$destinataires =  array('APA9TbHtVrGBZ1xDElPeyD5-rpbSH_
kWgZSprV5sgWYzS2aGfdT3nAMUcSgTyGWg7DC-zpG5C3fvVprcK9iTIl1ILLGo
oEAojgp2c1NXn1336IOiMLhwwRtRb2eF275Daz7isQ4uzcvz-scNaVkcpkkYq3NqdooA');


$ch = curl_init();
curl_setopt( $ch, CURLOPT_URL, "https://android.googleapis.com/gcm/send");
curl_setopt( $ch, CURLOPT_HTTPHEADER,
   array("Content-Type:application/json", "Authorization:key=" . $apiKey));
$gcm_data = array('data' => $messageData,
                  'registration_ids' => $destinataires);

curl_setopt( $ch, CURLOPT_POSTFIELDS, json_encode($gcm_data));


$response = curl_exec($ch);
curl_close($ch);

echo $response;
?>
```

Dans ce qui précède

— `$apiKey` est la clef associée au serveur exécutant ce script. Si cette clef est connue par G, ce dernier lit le message que lui transmet T.
— `$messageData` contient un tableau associatif stockant les éléments des données du message.
— `$destinataires` est un tableau contenant un clef par destinataire. Cette clef est générée par l'application A souhaitant recevoir les messages de G (voir section suivante).
— `$ch` est une session `curl`, initialisée, complêtée par un entête, du contenu via `curl_setopt` puis exécutée et fermée.

6.6 L'application A.

6.6.1 Enregistrer un nom d'utilisateur.

Dans l'application "parametres" de l'émulateur, renseigner un identifiant Google. Ceci est nécessaire pour l'utilisation de GCM.

6.6.2 Le code de l'application A.

Une application se décompose classiquement en trois classes :
— `MsgRecepteur` qui étend `BroadcastReceiver` : son rôle est de capture certaines intentions locales et de transmettre cela à l'activité principale.
— `GCMIntentService` qui étend `GCMBaseIntentService` : c'est cette classe qui reçoit toutes les intentions de type GCM, particulièrement lorsqu'un message GCM arrive. Dans ce cas, il émet une intention locale qui est capturée par le `MsgRecepteur`
— `MainActivity` : l'activité principale responsable de l'affichage notamment. C'est elle qui initie la demande d'enregistrement du mobile dans sa méthode `on_Resume`. Ceci effectué, elle active dans un thread séparé le `MsgRecepteur`.

Le code ci-dessous est celui de `MsgRecepteur`. Il active la méthode `msg_affichage_maj` lorsque la méthode `onRecieve` est invoquée.

```
public class MsgRecepteur extends BroadcastReceiver {
        MainActivity mg = null;

        public MsgRecepteur(MainActivity mg){
                this.mg = mg;
        }

        @Override
        public void onReceive(Context arg0, Intent arg1) {
                this.mg.msg_affichage_maj(
                    arg1.getCharSequenceExtra(
                      GCMIntentService.CONTENUDESMESSAGE).toString());
        }

}
```

Le code ci-dessous est celui de `GCMIntentService`. Il doit être complété avec les méthodes générées automatiquement.

```
public class GCMIntentService extends GCMBaseIntentService {
        public static final String NOUVEAUMESSAGEINTERNE = "fr.iut.couchot.MSG_RECU";
        public static final String CONTENUDESMESSAGE = "contenuDuMessageInterne";

        public GCMIntentService() {
                    super("iutbm1213android");
        }

        @Override
        protected void onMessage(Context arg0, Intent arg1) {
                Intent intent = new Intent(GCMIntentService.NOUVEAUMESSAGEINTERNE);
                intent.putExtra(GCMIntentService.CONTENUDESMESSAGE,
                                arg1.getStringExtra("unAutreMessage"));
                this.sendBroadcast(intent);
        }
```

et enfin

```
public class MainActivity extends Activity {

    private static final String SENDER_ID = "no_du_projet";
    static TextView t = null;
```

```
    MsgRecepteur msgrcpt = null;

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

        t = (TextView) findViewById(R.id.textcentral);

    }

    public void msg_affichage_maj(String messages) {
      t.setText(messages);
    }

    public void onResume() {
      super.onResume();
      GCMRegistrar.checkDevice(this);
      GCMRegistrar.checkManifest(this);
      final String regId = GCMRegistrar.getRegistrationId(this);
      if (regId.equals("")) {
        GCMRegistrar.register(this, SENDER_ID);
      } else {

      }

      this.msgrcpt = new MsgRecepteur(this);
      IntentFilter filter = new IntentFilter(GCMIntentService.NOUVEAUMESSAGEINTERNE);
      this.registerReceiver(this.msgrcpt, filter);
    }
```

et le manifest

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="fr.iut.couchot.gcm_chat"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
      android:minSdkVersion="10"
      android:targetSdkVersion="10" />
    <permission android:name="fr.iut.couchot.gcm_chat.permission.C2D_MESSAGE"
      android:protectionLevel="signature" />
    <uses-permission android:name="fr.iut.couchot.gcm_chat.permission.C2D_MESSAGE" />

    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />

    <!-- Keeps the processor from sleeping when a message is received. -->
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
        android:name=".MainActivity"
        android:label="@string/title_activity_main" >
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
      <receiver android:name="com.google.android.gcm.GCMBroadcastReceiver"
        android:permission="com.google.android.c2dm.permission.SEND" >
        <intent-filter>
          <action android:name="com.google.android.c2dm.intent.RECEIVE" />
          <action android:name="com.google.android.c2dm.intent.REGISTRATION" />
          <category android:name="fr.iut.couchot.gcm_chat" />
        </intent-filter>
      </receiver>
      <service android:name=".GCMIntentService" />
    </application>
</manifest>
```