PRÉPARATION ENVIRONNEMENT DE DÉVELOPPEMENT SOUS ECLIPSE INDIGO SR1

Installation du plug-in ADT (Android Development Tools pour Eclipse) et du SDK Android

- Dans le menu général d'Eclipse, sélectionner **Help – Install New Software…**
- Cliquer sur le bouton **Add…**
- Dans la nouvelle fenêtre **Add Repository** qui apparaît, entrer Android dans le champ Name et l'adresse **https://dl-ssl.google.com/android/eclipse/** dans le champ **Location**.

Si un problème de téléchargement apparaît dans la liste des composants disponibles, remplacer **https** par **http**.

- Le groupe **Developer Tools** apparaît dans la liste des composants disponibles. Cocher la case figurant devant ce nom de groupe.
- Une fois le plug-in installé, on est invité à redémarrer Eclipse.
- Après redémarrage, Eclipse propose de télécharger le SDK Android et de l'installer.

Vérification de l'installation du SDK Android :

- Commande [Window – Préférences]. Sélectionner "Android" dans la partie gauche. Constater dans la partie droite, que le champ "SDK Location" est bien renseigné avec le nom du dossier que l'on avait indiqué pour le téléchargement du SDK Android.
- ***Affichage de la liste des composants du SDK installés*** **:** Commande [Window - Android SDK Manager]

DÉMARCHE DE CRÉATION D'UN PROJET ANDROID

Création du projet

Commande [File – New Project]. Dérouler la branche "Android" et choisir "Android Project".

Propriété "Application Name" : Nom de l'application qui s'affichera sur le terminal Android.

Propriété "Package Name" :

Nom interne de l'application sur le terminal Android. Il s'agit d'un nom unique parmi toutes les applications développées pour terminaux Android. Aussi, on utilise le format standard suivant :

extension_de_votre_domaine*.*nom_du_domaine*.android.*nom_du_projet

Exemple : **com.masociete.android.monappli**

Propriété "Create Activity" :

Si ce champ est activé, il indique que l'assistant devra créer une activité dont le nom est indiqué juste à côté. Une activité est la couche de présentation de votre application (concept d'écran)

Compilation

Le projet est automatiquement compilé lorsque l'on demande son application. La compilation engendre la création d'un fichier binaire ***nom-application*.apk** dans le répertoire **bin** du projet.

Exécution

Exécution sur l'émulateur Android

Création de l'AVD

Il faut commencer par créer un AVD (Android Virtual Device) selon la procédure suivante :

- Commande [Window – AVD Manager] et cliquer sur le bouton "New…"
- On arrive sur une boîte de dialogue sur laquelle de nombreux champs sont à renseigner :
  - **Name** : Nom que l'on donne à l'AVD. Il ne doit pas contenir de caractères spéciaux ni d'espaces.
  - **Snapshot** :
    Lorsque cette option est cochée, il est possible de sauvegarder l'AVD, de façon à mémoriser un état de cet AVD, par exemple à un moment où on lui a rentré certaines données en interne.
    On peut aussi faire une sauvegarde dans le but de provoquer un démarrage rapide de l'émulateur à partir de l'état sauvegardé.
  - **SD Card** :
    Permet d'émuler la présence d'une carte SD dans l'appareil virtuel en spécifiant une taille. Il est également possible de réutiliser une carte SD virtuelle précédemment créée en indiquant le fichier image de cette carte SD.
  - **Skin** : Permet de spécifier la résolution de l'écran de l'appareil émulé.
  - **Hardware** :
    Permet d'affiner la configuration matérielle en renseignant des propriétés avancées comme par ex la présence d'un appareil photo ou d'un clavier physique.

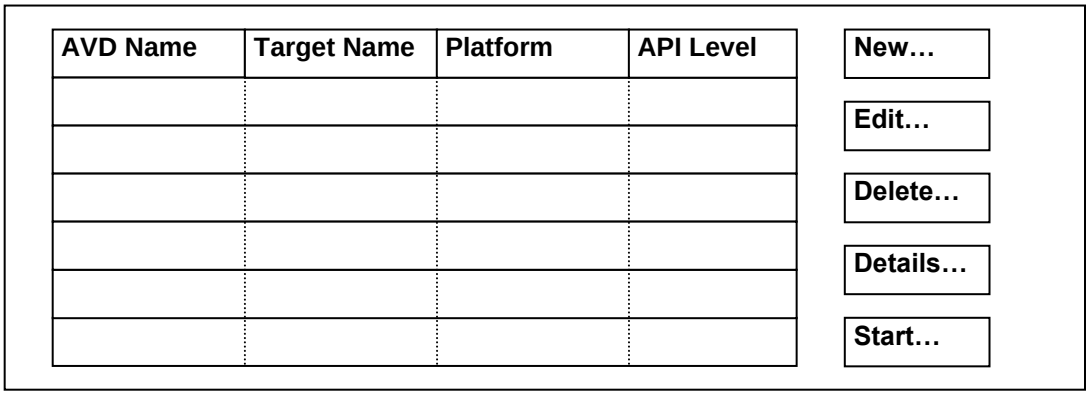
A noter :

► Si on a l'erreur "unable to find a 'userdata.img' file for abi", c'est qu'il faut installer le package "ARM EABI System Image" : Commande [Window – Android SDK Manager]. La ligne "ARM EABI System Image" apparaît cochée. Cliquer alors sur le bouton "Installa Package".

► Il est possible de créer autant d'AVD que l'on souhaite. Cela permet de tester une même application sur de nombreuses configurations matérielles.

→

Fenêtre de l'AVD Manager :



Démarrage de l'AVD

Avant de lancer l'application, il faut démarrer l'AVD. Commande [Window - AVD Manager]. Le bouton **Start** permet de démarrer l'émulateur sélectionné. Patienter pendant le démarrage du système. Cela peut prendre plusieurs minutes selon la puissance du poste de développement. D'où l'intérêt de pouvoir restaurer l'AVD depuis une sauvegarde en cochant la case **Launch from snapshot**. L'application est ensuite chargée et lancée automatiquement.

Déverrouillage :

Si l'écran de verrouillage apparaît (cadenas au bas de l'écran), le déverrouiller pour voir apparaître l'application lorsqu'elle sera exécutée. Pour effectuer ce déverrouillage, glisser à droite le cadenas vers l'extérieur, avec la souris.

Lancement de l'application

- *Réglage préliminaire à effectuer :*
  Le temps de chargement de l'application sur l'AVD pouvant être long, il faut augmenter le délai au-delà duquel un "time out" se produit, qui est fixé initialement à 5 secondes (5000 ms) ⇒ Commande [Window – Preferences – Android – DDMS], puis modifier la zone de texte "ADB connection time out (ms"=)".
- Sur la feuille "Package Explorer" d'Eclipse, faire un clic droit sur le projet et choisir "Run As – Android Application".
- *Remarque :*
  Dans certaines documentations, il est mentionné que cette commande lance l'AVD, puis installe l'application sur cet AVD et l'exécute, et que de ce fait, l'étape précédente de démarrage de l'AVD ne serait pas nécessaire. Après, tests, il y a bien effectivement démarrage de l'AVD, mais soit on a une erreur de chargement de l'application, soit elle ne s'y exécute pas.
- Si un message d'erreur du type "The connection to ad bis down, and a severe error has occured" apparaît, redémarrer Eclipse.

UNE PREMIÈRE APPLICATION

Dans la partie gauche d'Eclipse, dérouler la branche "src", sous le nom donné au projet, de façon à ce qu'apparaisse le nom de fichier ***nom-projet*Activity.java**, fichier généré automatiquement, dont voici le contenu :

```
package edu.gomet.android.appli1;

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

public class Android1Activity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

→ "Android1" = nom donné au projet. En fait, cette classe déclarée ici doit porter le "Android1Activity", qui est le nom de l'activité (écran) de l'application, spécifié dans le fichier **AndroidManifest.xml**, par l'attribut **android.name** de la balise **<activity>**.

→ Cette instruction signifie que l'on va remplir l'écran du téléphone avec les objets graphiques qui sont définis dans le fichier "**main.xml**" (qui sera étudié plus loin), qui contient par défaut une zone de texte remplie avec le texte "Hello"

On va faire en sorte que l'application affiche simplement le texte "Bonjour", en remplaçant la ligne "**setContentView(R.layout.main)**" par les lignes suivantes :

```
TextView tv = new TextView(this);
tv.setText("Bonjour");
setContentView(tv);
```

APPLICATION RÉALISANT LA SOMME DE 2 VALEURS SAISIES

Contenu de l'interface graphique

- Une zone de texte pour saisir un montant de frais de taxi
- Une zone de texte pour saisir un montant de frais de train
- Une zone de texte pour recevoir le total des frais saisis
- Un bouton permettant de lancer le calcul.

Réalisation de l'interface graphique

- Dans la partie gauche de la fenêtre d'Eclipse, dérouler la branche "res", sous le nom donné au projet, puis la branche "layout". Double-cliquer alors sur le fichier "**main.xml**". Un onglet "main.xml" apparaît alors dans la partie droite, comportant un écran de téléphone, et une boîte à outils contenant des contrôles graphiques que l'on peut placer sur l'écran.
- Cet onglet "main.xml" est lui-même constitué de sous-onglets "Graphical Layout" et "main.xml", apparaissant dans sa partie basse. Le sous-onglet "main.xml" permet de visualiser et éditer directement le fichier "main.xml", alors que l'onglet "Graphical Layout" contient l'écran de téléphone et la boîte à outils.
- ***Contenu initial du fichier "main.xml" :***

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

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

</LinearLayout>
```

On voit qu'il y a par défaut une "TextView" contenant le texte "Hello". C'est le texte que l'on aurait vu si, dans la première application, on avait laissé l'instruction **setContentView(R.layout.main)**.

- ***A propos de la modification directe dans le fichier "main.xml" :***

Si l'on vient de faire des positionnements de contrôles graphiques dans le sous-onglet "Graphical Layout", on n'arrive pas, sur le sous-onglet "main.xml", à éditer le fichier. De plus, lorsque l'on déplace le curseur à l'intérieur du fichier, les instructions disparaissent puis réapparaissent. Il faut en fait d'abord fermer l'onglet "main.xml", puis le rouvrir en double-cliquant à nouveau sur "main.xml" dans la partie gauche de la fenêtre d'Eclipse.

● ***Placement des contrôles graphiques sur l'écran de téléphone :***

- Conteneur LinearLayout pour contenir horizontalement le libelle "Frais Taxi" et la zone de saisie associée :

► Dans la boîte à outils, prendre l'outil "LinearLayout" associé à un icône représentant un rectangle contenant des traits verticaux (l'autre outil "LinearLayout" associé lui à un icône représentant un rectangle contenant des traits verticaux est un layout pour contenir des contrôles les uns sous les autres).

► Le layout ne se voit pas sur l'écran de téléphone, mais pour le faire apparaître, cliquer sur son nom dans la partie droite de la fenêtre d'Eclipse. On peut modifier son orientation (verticale ou horizontale) par un clic droit et en choisissant "Orientation".

► Une fois que l'on a glissé le layout sur l'écran de téléphone, il se place tout en haut de l'écran de téléphone, ou juste dessous le layout précédent s'il y en a déjà un. Pour le positionner plus bas, effectuer un clic droit sur le layout, puis choisir "Other Properties", puis "Layout Parameters", puis "LayoutMarginTop", et mettre la valeur de cette propriété à "50dp" par exemple.

► A noter que cette propriété de positionnement "LayoutMarginTop", ainsi que les autres, telles que "LayoutMarginLeft" par ex, existent pour tous les contrôles, autres que des Layout, et que l'on y a accès par le même menu conceptuel.

- Champ de texte (TextView) contenant le libelle "Frais Taxi" :

► Dans la boîte à outils, les contrôles "TextView" sont dans la partie "Form Widgets". Ils sont au nombre de 4, intitulés respectivement "TextView", "Large", "Medium" et "Small", correspondant à des TextView de différentes tailles. On peut prendre l'outil "Large".

► Pour affecter le libellé à ce champ de texte, effectuer un clic droit sur ce dernier, choisir "Edit Text", puis cliquer sur le bouton "New String". Une boîte de dialogue s'ouvre alors, permettant de créer une nouvelle chaîne de caractères, avec un identifiant (champ "New R.String") et un contenu (champ "String").

→ Les chaînes de caractères créées sont enregistrées dans le fichier "**string.xml**", que l'on peut faire apparaître dans la partie gauche de la fenêtre d'Eclipse, dans la branche **res\values**, sous le nom donné au projet. En double-cliquant sur ce fichier "**string.xml**", on peut le visualiser et l'éditer directement. Noter cependant que lorsque l'on modifie le contenu d'une chaîne de caractères dans le fichier "**string.xml**", il faut fermer le fichier "**main.xml**" et le rouvrir pour voir apparaître le nouveau contenu sur l'écran du téléphone.

→ Dans le fichier "**main.xml**", la ligne spécifiant le libellé désiré est la suivante :

android:text=*"@string/Taxi"*

"Taxi" étant l'identifiant de la chaîne créée, contenant le texte "Frais Taxi".

Noter que l'on peut travailler sans créer de chaînes dans le fichier "**string.xml**", en remplaçant la ligne précédente par la ligne :

android:text=*"Frais Taxi"*

En procédant comme cela, on a néanmoins un warning "Hardcoded string, should use @string resource".

- Champ de texte de saisie (EditView) pour saisir les frais de taxi :

Sélection du contrôle dans la boîte à outils

Sélectionner le contrôle "abc", dans la partie "TextFields".

Réglage de la couleur de fond (à blanc par exemple)

Par défaut, la couleur de fond est noire, comme celle du téléphone, ce qui ne convient pas.

Commencer par créer un fichier de spécifications de couleurs, en effectuant, dans la partie gauche de la fenêtre d'Eclipse, un clic droit sur le dossier "res\values", sous le nom donné au projet. Choisir la commande [New – File] et donner le nom "colors.xml" au fichier. S'ouvre alors dans la partie centrale d'Eclipse une fenêtre permettant de créer des ressources d'attributs de couleur. Cliquer alors sur le bouton "Add…", sélectionner l'élément "Color", donner par exemple "txtbackground" comme identifiant à la ressource, et lui donner comme valeur "#FFF" (blanc).

Revenir ensuite sur l'onglet "main.xml". Effectuer un clic droit sur le champ de texte, choisir "Other Properties – All By Name – BackGround…", et sélectionner la ressource précédemment créée c'est-à-dire la ressource "txtbackground".

Réglage de la couleur du texte (à noir par exemple)

Comme on a mis une couleur de fond blanche, il faut régler la couleur du texte à noire, car par défaut, elle est gris très clair et serait donc à peine perceptible. Dans le fichier "colors.xml", on créera donc comme précédemment une ressource ayant comme identifiant "txtcolor" par exemple, et comme valeur "#000" (noir). Et on affectera cet attribut à la propriété "TextColor" du champ de texte.

Faire en sorte que la saisie des chiffres soit autorisée dans le champ de texte

Dans la partie droite de la fenêtre d'Eclipse, effectuer un clic droit sur le champ de texte, choisir "Input Type" et cocher "Number". L'item "Text" est coché par défaut, et ne se décoche pas.

Dans le fichier "main.xml", la ligne spécifiant cette propriété "InputType" est alors :

android:InputType="number|text"

Si on veut autoriser uniquement la saisie de chiffres, il faut intervenir directement dans le fichier "main.xml" et enlever le type "text".

A propos des formats que l'on peut choisir pour un champ de texte

On peut choisir une multitude de formats, comme par exemple **DateTime**, **Phone**, **NumberPassword**, **TextPassword**, **TextMultiLine**, **TextEmailAddress**, etc.

Réglage de la largeur

Lorsque le contrôle EditView se trouve à l'intérieur d'un layout, le réglage de sa largeur n'est apparemment pas possible. Il occupe toute la largeur restante du layout.

- Bouton permettant de lancer le calcul :

Pour centrer le bouton, il faut également le mettre dans un layout. Puis, effectuer un clic droit dessus et choisir la commande [*NomLayout* – Gravity – Center]

Programmation de l'application : Contenu du fichier *nom-projet*Activity.java

```
package edu.gomet.android.appli1;

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

public class Android1Activity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button calculer;
        calculer = (Button) findViewById(R.id.button1);
        calculer.setOnClickListener(new View.OnClickListener() {
          public void onClick(View v) {
                EditText edtaxi;
                EditText edtrain;
                EditText edtotal;
                double total;
                edtaxi = (EditText) findViewById(R.id.editTaxi);
                edtrain = (EditText) findViewById(R.id.editTrain);
                total= Double.valueOf(edtaxi.getText().toString()).doubleValue();
                total= total + Double.valueOf(edtrain.getText().toString()).doubleValue();
                edtotal = (EditText) findViewById(R.id.editTotal);
                edtotal.setText(String.valueOf(total));
          }
        });
```

→ Signifie que l'on affiche l'interface définie dans le fichier **main**(.xml), se trouvant dans la branche **layout** du projet.

A propos des 4 dernières instructions "import" :

On les fait générer par Eclipse, lorsque ce dernier signale une erreur après saisie d'une instruction :

Exemple 1 :

Après avoir tapé l'instruction :

Button calculer;

on a l'erreur "Button cannot be resolved to a type", et le premier choix de correction proposé est "Import Button (android widget)"

Exemple 2 :

Après avoir tapé l'instruction :

calculer.setOnClickListener(new View.OnClickListener() {

on a l'erreur "View cannot be resolved to a type", et le premier choix de correction proposé est "Import View (android view)"

Test de l'application : Pour la saisie dans les EditText, penser à utiliser le clavier affiché sur le téléphone.

AFFICHER UN DEUXIÈME ECRAN ET LUI PASSER DES DONNÉES

Exemple : Ecran de saisie d'un login et d'un mot de passe, ouvrant un deuxième écran affichant le login saisi.

Création d'une deuxième classe d'activité pour le deuxième écran :

Pour créer une classe d'activité, effectuer un clic droit sur le nom du package situé sous la branche "src", puis choisir la commande "New – Other – Android Activity". Ainsi, le fichier **manifest.xml** sera mis à jour (il faut qu'il y ait dans ce fichier une balise **<activity>** pour chaque activité). Si on crée la classe comme une classe ordinaire, avec la commande "New – Class", le fichier **manifest.xml** ne serait pas à jour.

Code ouvrant le deuxième écran :

```
EditText login = (EditText) findViewById(R.id.email);
String loginStr = login.getText().toString();
EditText pass = (EditText) findViewById(R.id.password);
String passStr = pass.getText().toString();

Intent intent = new Intent(MainViewActivity.this,DisplayLoginActivity.class);
intent.putExtra("login", loginStr);
intent.putExtra("password", passStr);
startActivity(intent);
```

Code de la deuxième activité :

```
setContentView(R.layout.login_display);
Intent thisIntent = getIntent();
String login = thisIntent.getExtras().getString("login");
String pass = thisIntent.getExtras().getString("password");
TextView loginText = (TextView) findViewById(R.id.loginText);
String message = "Bonjour " + login;
loginText.setText(message);
```

UTILISER DES ONGLETS (TABLAYOUT) – FAIRE PLUSIEURS ÉCRANS DIFFÉRENTS (ACTIVITÉS) DANS UNE APPLICATION – GESTION D'IMAGES AVEC DES RESSOURCES "DRAWABLE"

On va réaliser 3 onglets "Artists", "Albums", "Songs". Chaque onglet utilisera une activité propre.

Création d'une classe d'activité par onglet :

Il faut commencer à créer, dans le projet, en plus de l'activité principale générée automatiquement lors de la création du projet Android, 3 classes d'activité : **ArtistActivity**, **AlbumActivity** et **SongActivity**.

Pour créer une classe d'activité, effectuer un clic droit sur le nom du package situé sous la branche "src", puis choisir la commande "New – Other – Android Activity". Ainsi, le fichier **manifest.xml** sera mis à jour (il faut qu'il y ait dans ce fichier une balise **<activity>** pour chaque activité). Si on crée la classe comme une classe ordinaire, avec la commande "New – Class", le fichier **manifest.xml** ne serait pas à jour.

Remplacer ensuite le contenu de la classe par (pour la classe **ArtistActivity**) :

```
public class ArtistsActivity extends Activity {
  public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  TextView textview = new TextView(this);     ) On peut aussi faire le contenu de l'onglet avec l'assistant graphique.
  textview.setText("This is the Artists tab"); ) Il faut à ce moment-là remplacer ces 3 lignes par :
  setContentView(textview);                    ) setContentView (R.layout.main_ArtistActivity)
                                               ) (main_ArtistActivity = Nom du fichier XML contenant la définition
                                               ) des contrôles graphiques constituant l'interface de l'onglet "Activity"
  }
}
```

(on procédera par recopie du contenu du fichier ***nom-projet*Activity.java**)

Noter que si, pour ces 3 activités créées, on crée par code le contenu de leur interface (juste une TextView), elles n'utilisent pas de fichier layout propre.

Si on voulait définir le contenu de chaque écran (activité) supplémentaire à l'aide d'un fichier XML :

- Dans la partie gauche de la fenêtre d'Eclipse, dérouler la branche "res", sous le nom donné au projet. Effectuer un clic droit sur la branche "layout", et choisir [New – Other – XML – XML File], et donner un nom au fichier XML.
- Le fichier créé s'ouvre alors dans un nouvel onglet, contenant juste la ligne :

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

  et avec l'erreur "Premature End of File".
- Fermer alors l'onglet, et le rouvrir en double-cliquant sur le nom du fichier dans la partie gauche de la fenêtre d'Eclipse.
- On se retrouve sur le sous-onglet "Graphical Layout" de l'onglet de nom égal au nom du fichier XML. La boîte à outils apparaît bien mais pas d'écran de téléphone vierge. Ceci est dû au fait que dans le fichier XML, il n'y a pas de layout principal de défini. D'ailleurs, le message "No XML content. Please add a root view or layout to your document." est affiché.
  Il suffit alors de recopier, à partir du fichier **main.xml**, les lignes suivantes :

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

  </LinearLayout>
  ```

- Pour que ce fichier XML soit associé à la classe d'activité correspondante, il faudrait, dans l'instruction
  setContentView(R.layout.*main*);
  de sa méthode **OnCreate**( ), remplacer "**main**" par le nom du fichier XML venant d'être créé.

Réalisation de l'icône de chaque onglet :

Chaque icône sera réalisé en 2 versions, une correspondant au cas où l'onglet est sélectionné, et une correspondant au cas où l'onglet n'est pas sélectionné. Une recommandation est de faire l'icône de l'état "sélectionné" de couleur sombre (gris), et celui de l'état "non sélectionné" de couleur lumineuse (blanche). Par ex :

Ces images seront sauvegardées dans le répertoire **res/drawable** du projet. Pour les faire apparaître dans la partie gauche d'Eclipse, dans la branche **res/drawable** du projet, effectuer un clic droit sur cette branche et choisir "Refresh".

On aura 6 images en tout (2 images x 3 onglets), que l'on pourra appeler par ex : **ic_tab_artits_grey.png**, **ic_tab_artits_white.png**, **ic_tab_albums_grey.png**, **ic_tab_albums_white.png**, **ic_tab_songs_grey.png**, **ic_tab_songs_white.png**

Création d'une "Statelistdrawable", qui est un bloc de lignes XML spécifiant une image à afficher selon un état ("sélectionné" ou "non sélectionné") :

Pour l'onglet "Artists", on va créer dans le répertoire **res/drawable** du projet, un fichier de nom **ic_tab_artits.xml** par ex contenant les lignes suivantes :

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- When selected, use grey -->
    <item android:drawable="@drawable/ic_tab_artists_grey"
          android:state_selected="true" />
    <!-- When not selected, use white-->
    <item android:drawable="@drawable/ic_tab_artists_white" />
</selector>
```

Explications à propos des clauses "android:drawable="@drawable/- - -" :

On voit que pour ces clauses, on ne met pas directement le chemin et le nom des fichiers **.png**, mais le nom d'une ressource de type "drawable". En effet, dès que l'on met un fichier image **.png** ou **.jpg** ou **.gif** (ce dernier format d'image étant néanmoins déconseillé) dans le répertoire **res/drawable** du projet, Android crée une ressource "drawable" portant le nom du fichier image.

Réalisation de l'interface graphique :

Ouvrir le fichier **res/layout/main.xml** et mettre le contenu suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="5dp">
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:padding="5dp" />
    </LinearLayout>
</TabHost>
```

Explications :

- Un TabHost est un conteneur pour une vue composée d'onglets.
- Il requiert la présence d'un TabWidget et d'un FrameLayout. Pour positionner le TabWidget et le FrameLayout verticalement, un LinearLayout est utilisé. Le FrameLayout est là où va le contenu de chaque onglet.
- Noter que le TabWidget et le FrameLayout ont respectivement les id "tabs" et "tabcontent". Ces noms doivent être utilisés pour que le TabHost puisse retrouver la référence de chacun d'eux. Il attend exactement ces noms.

Programmation de l'application :

- Ouvrir le fichier ***nom-projet*Activity.java**, et faire hériter la classe de l'activité de la classe **Tabactivity** :

```
public class HelloTabWidget extends TabActivity {
```

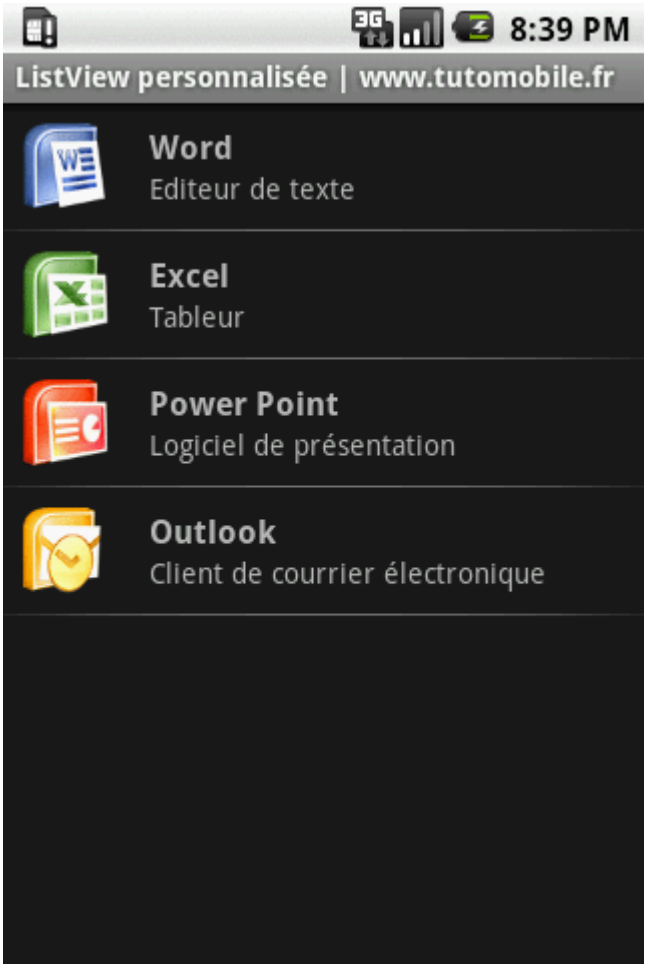
- Puis utiliser le code suivant pour la méthode **OnCreate** :

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

    Resources res = getResources(); // Resource object to get Drawables
    TabHost tabHost = getTabHost();  // The activity TabHost
    TabHost.TabSpec spec;  // Resusable TabSpec for each tab
    Intent intent;  // Reusable Intent for each tab

    // Create an Intent to launch an Activity for the tab (to be reused)
    intent = new Intent().setClass(this, ArtistsActivity.class);

    // Initialize a TabSpec for each tab and add it to the TabHost
    spec = tabHost.newTabSpec("artists").setIndicator("Artists",
                      res.getDrawable(R.drawable.ic_tab_artists))
                  .setContent(intent);
    try
    {
        tabHost.addTab(spec);
    }
    catch (Exception e)
    { }

    // Do the same for the other tabs
    intent = new Intent().setClass(this, AlbumsActivity.class);
    spec = tabHost.newTabSpec("albums").setIndicator("Albums",
                      res.getDrawable(R.drawable.ic_tab_albums))
                  .setContent(intent);

    try
    {
       tabHost.addTab(spec);
    }
    catch (Exception e)
    { }

    intent = new Intent().setClass(this, SongsActivity.class);
    spec = tabHost.newTabSpec("songs").setIndicator("Songs",
                      res.getDrawable(R.drawable.ic_tab_songs))
                  .setContent(intent);
    try
    {
       tabHost.addTab(spec);
    }
    catch (Exception e)
    { }

    try
    {
       tabHost.setCurrentTab(2);
    }
    catch (Exception e)
    { }
}
```

Noter que dans ce code, chaque instruction **addTab** et l'instruction **setCurrentTab** ont été placées dans un bloc **try-catch**. Si on ne le fait pas, on a apparemment une erreur d'exécution.

UTILISER UNE LISTEVIEW (LISTE D'ÉLÉMENTS) – AFFICHER UNE BOITE DE DIALOGUE – DÉFINITION DE COULEURS

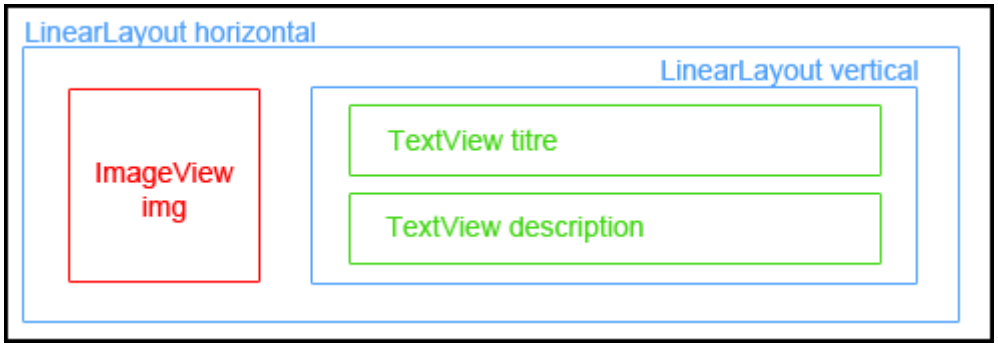
Exemple :



Exemple d'application : Affichage d'une liste d'appartements pour une application immobilière.

Procédure :

- Placer un contrôle ListView sur l'écran du téléphone
- Il faut ensuite créer un autre fichier XML, que l'on va appeler **affichageitem.xml**, qui va définir une vue qui correspondra au contenu des items de la ListView.
  Cette vue aura la structure suivante :



Code Java :

```
package edu.gomet.android.listView;

import java.util.ArrayList;
import java.util.HashMap;

import android.app.Activity;
import android.app.AlertDialog;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.AdapterView.OnItemClickListener;

public class ListViewActivity extends Activity {

  private ListView maListViewPerso;

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

      //Récupération de la listview créée dans le fichier main.xml
      maListViewPerso = (ListView) findViewById(R.id.listviewperso);

      //Création de la ArrayList qui nous permettra de remplir la listView
      ArrayList<HashMap<String, String>> listItem = new ArrayList<HashMap<String, String>>();

      //On déclare la HashMap qui contiendra les informations pour un item
      HashMap<String, String> map;

      //Création d'une HashMap pour insérer les informations du premier item de notre listView
      map = new HashMap<String, String>();
      //on insère un élément titre que l'on récupérera dans le textView titre créé dans le fichier affichageitem.xml
      map.put("titre", "Word");
      //on insère un élément description que l'on récupérera dans le textView description créé dans le fichier affichageitem.xml
      map.put("description", "Editeur de texte");
      //on insère la référence à l'image (converti en String car normalement c'est un int) que l'on récupérera dans l'imageView
créé dans le fichier affichageitem.xml
      map.put("img", String.valueOf(R.drawable.word));
      //enfin on ajoute cette hashMap dans la arrayList
      listItem.add(map);

      //On refait la manip plusieurs fois avec des données différentes pour former les items de notre ListView

      map = new HashMap<String, String>();
      map.put("titre", "Excel");
      map.put("description", "Tableur");
      map.put("img", String.valueOf(R.drawable.excel));
      listItem.add(map);

      map = new HashMap<String, String>();
      map.put("titre", "Power Point");
      map.put("description", "Logiciel de présentation");
      map.put("img", String.valueOf(R.drawable.powerpoint));
      listItem.add(map);
```

```
        map = new HashMap<String, String>();
        map.put("titre", "Outlook");
        map.put("description", "Client de courrier électronique");
        map.put("img", String.valueOf(R.drawable.outlook));
        listItem.add(map);

        //Création d'un SimpleAdapter qui se chargera de mettre les items présents dans notre list (listItem) dans la vue affichageitem
        SimpleAdapter mSchedule = new SimpleAdapter (this.getBaseContext(), listItem, R.layout.affichageitem,
            new String[] {"img", "titre", "description"}, new int[] {R.id.img, R.id.titre, R.id.description});

        //On attribue à notre listView l'adapter que l'on vient de créer
        maListViewPerso.setAdapter(mSchedule);

        //Enfin on met un écouteur d'évènement sur notre listView
        maListViewPerso.setOnItemClickListener(new OnItemClickListener() {
            @SuppressWarnings("unchecked")
            public void onItemClick(AdapterView<?> a, View v, int position, long id) {
                    //on récupère la HashMap contenant les infos de notre item (titre, description, img)
                    HashMap<String, String> map = (HashMap<String, String>) maListViewPerso.getItemAtPosition(position);
                    //on créé une boite de dialogue
                    AlertDialog.Builder adb = new AlertDialog.Builder(ListViewActivity.this);
                    //on attribue un titre à notre boite de dialogue
                    adb.setTitle("Sélection Item");
                    //on insère un message à notre boite de dialogue, et ici on affiche le titre de l'item cliqué
                    adb.setMessage("Votre choix : "+map.get("titre"));
                    //on indique que l'on veut le bouton ok à notre boite de dialogue
                    adb.setPositiveButton("Ok", null);
                    //on affiche la boite de dialogue
                    adb.show();
            }
        });

    }
}
```

Application supplémentaire : Ajout d'une case à cocher dans chaque item, et changement de la couleur de fond de l'item lorsque l'on coche la case à cocher

Dans le fichier "affichageitem.xml", définition d'une méthode qui permettra de récupérer l'événement du clic sur la checkbox :

Ceci sera fait par la directive **android:onClick="MyHandler"**, comme indiqué ci-dessous :

```
<CheckBox android:layout_height="wrap_content"
                        android:id="@+id/check"
                        android:layout_width="wrap_content"
                        android:layout_gravity="right"
                        android:layout_marginRight="10sp"
                        android:onClick="MyHandler"/>
```

Cette méthode "**MyHandler**" sera à écrire.

Dans le code Java, utilisation d'un adaptateur personnel au lieu de l'adaptateur "SimpleAdapter" pour mettre les items présents dans notre list (listItem) dans la vue "affichageitem" :

- Nouveau code :

  ~~SimpleAdapter~~ MyListAdapter mSchedule = new ~~SimpleAdapter~~ MyListAdapter (this.getBaseContext(), listItem, R.layout.affichageitem, new String[] {"img", "titre", "description"}, new int[] {R.id.img, R.id.titre, R.id.description});

- A propos de la classe "MyAdapter" : Elle sera à écrire.
- Pourquoi utiliser un adaptateur personnel :

  Parce que l'on va "attacher" à chaque checkbox un "tag" (une étiquette en quelque sorte) contenant la position (l'index) de la checkbox dans la liste View. Cette position nous servira ensuite à changer la couleur de fond du bon item dans la ListView. La création de ce tag se fera donc dans un adaptateur personnel

Ecriture de la méthode "MyHandler", dans le code Java, après la fermeture de la méthode onCreate(Bundle savedInstanceState) :

```
public void MyHandler(View v) {
  CheckBox cb = (CheckBox) v;
  //on récupère la position à l'aide du tag défini dans la classe MyListAdapter
  int position = Integer.parseInt(cb.getTag().toString());

  // On récupère l'élément sur lequel on va changer la couleur
  View o = maListViewPerso.getChildAt(position).findViewById(R.id.blocCheck);
```

→ "blockCheck" = nom du layout horizontal encadrant le tout.

```
  //On change la couleur
  if (cb.isChecked()) {
    o.setBackgroundResource(R.color.green);
  } else {
    o.setBackgroundResource(R.color.blue);
    }
  }
```

Définition des couleurs utilisées dans un fichier color.xml :

- Effectuer un clic droit sur la branche ***nom-projet*/res/values** et choisir [New – Android XML File]
- Taper "**color**" dans le champ "File:", puis cliquer sur le bouton "Finish".
- On se trouve alors sur un premier onglet "Ressources" d'un groupe de 2 onglets, le deuxième portant le nom "Color.xml" et affichant le contenu de ce fichier.
- Cliquer sur le bouton "Add…"
- Une boîte de dialogue "Create a new element at the top level" s'affiche alors. Sélectionner alors "Color"
- Saisir ensuite des noms de couleurs avec leur valeur, de façon à obtenir le fichier **color.xml** suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="blue">#3050A0</color>
    <color name="green">#006600</color>
</resources>
```

Ecriture de la classe MyListAdapter :

Fichier **MyListAdapter.java** :

```
package edu.gomet.android.listView;

import java.util.List;
import java.util.Map;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.CheckBox;
import android.widget.SimpleAdapter;

public class MyListAdapter extends SimpleAdapter
{
   private LayoutInflater          mInflater;

   public MyListAdapter (Context context, List<? extends Map<String, ?>> data,
                         int resource, String[] from, int[] to)
   {
            super (context, data, resource, from, to);
            mInflater = LayoutInflater.from (context);

   }

   @Override
   public Object getItem (int position)
   {
            return super.getItem (position);
   }
```

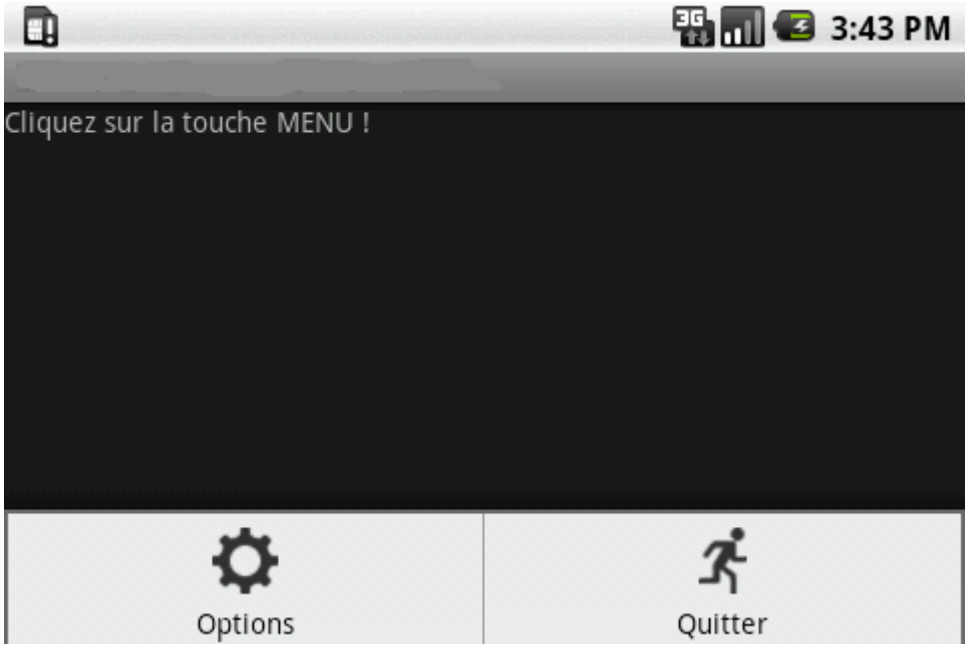
```
    @Override
    public View getView (int position, View convertView, ViewGroup parent)
    {
            //Ce test permet de ne pas reconstruire la vue si elle est déjà créée
            if (convertView == null)
            {
                    // On récupère les éléments de notre vue
                    convertView = mInflater.inflate (R.layout.list_detail, null);
                    // On récupère notre checkBox
                    CheckBox cb = (CheckBox) convertView.findViewById (R.id.check);
                    // On lui affecte un tag comportant la position de l'item afin de
                    // pouvoir le récupérer au clic de la checkbox
                    cb.setTag (position);
            }
            return super.getView (position, convertView, parent);
    }

}
```

FAIRE UN MENU ET SOUS-MENU – AFFICHER UN MESSAGE TRANSITOIRE DIT "TOAST"

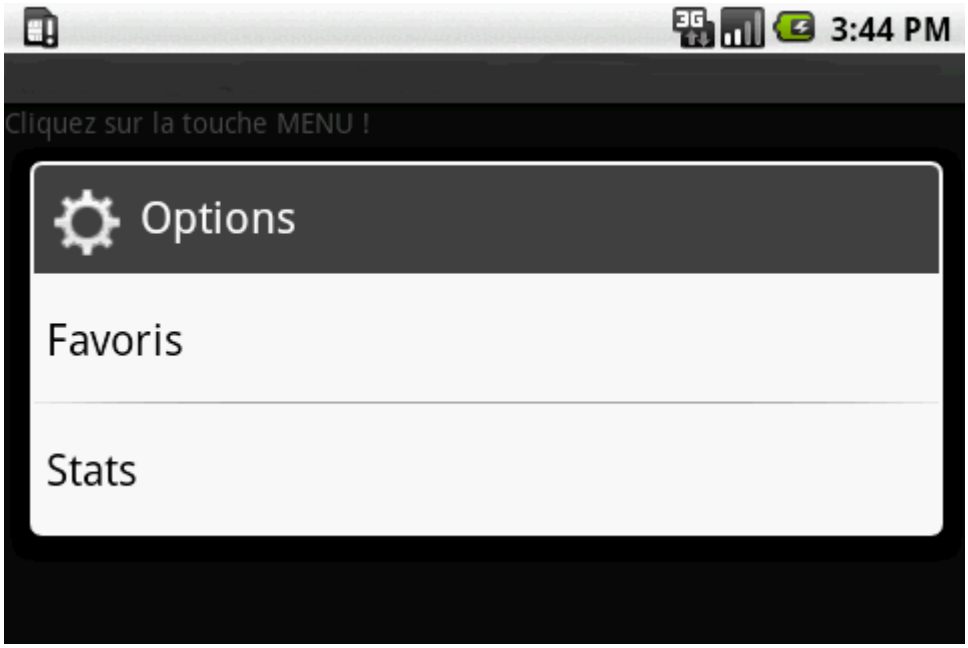
A savoir :

Lorsqu'une application comporte un menu, celui-ci ne s'affiche pas automatiquement au lancement de l'application. Il faut que l'utilisateur appuie sur la touche "Menu" du téléphone. Aussi, souvent, sur le layout de telles applications, on place une TextView contenant le texte "Cliquez sur la touche Menu".

Le menu allant être réalisé :



L'aspect du sous-menu "Options" :



Codage du menu en XML :

Nous venons de voir que le menu ne s'affichait pas au lancement de l'application, mais que cette dernière avait dans son layout, une invitation pour l'utilisateur à appuyer sur la touche "Menu" du téléphone. Par conséquent, le menu sera codé dans un fichier XML autre que le fichier **main.xml**.

On va donc créer un fichier **menu.xml** dans le dossier **layout**. Le premier menu contient les items **Options** et **Quitter**. Lorsqu'on cliquera sur **Quitter** , on sortira de l'application et lorsqu'on cliquera sur **Options**, on aura un sous-menu qui s'affichera avec les items **Favoris** et **Stats**.

Le code XML sera le suivant (le menu XML doit être codé en XML directement, car dans la boîte à outils des contrôles graphiques, il n'y a pas d'outil pour construire un menu) :

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
	<item android:id="@+id/option"
		android:title="Options"
		android:icon="@drawable/option">
		<menu android:id="@+id/sousmenu">
			<item android:id="@+id/favoris"
				android:title="Favoris"   />
			<item android:id="@+id/stats"
				android:title="Stats"  />
		</menu>
	</item>
	<item android:id="@+id/quitter"
		android:title="Quitter"
		android:icon="@drawable/quit" />
</menu>
```

On remarquera que l'on affecte une petite icône pour chaque item du menu mais pas du sous-menu. Tout simplement parce que dans le sous-menu, les icônes ne sont pas supportées par Android.

Code Java :

```
package com.tutomobile.android.menu;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.Toast;

public class Tutoriel10_Android extends Activity {

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

   //Méthode qui se déclenchera lorsque l'on appuie sur le bouton menu du téléphone
   public boolean onCreateOptionsMenu(Menu menu) {
      //Création d'un MenuInflater qui va permettre d'instancier un Menu XML en un objet Menu
      MenuInflater inflater = getMenuInflater();
      //Instanciation du menu XML spécifier en un objet Menu
      inflater.inflate(R.layout.menu, menu);

      // *** Si on regarde l'aspect du sous-menu "Options" sur la copie d'écran précédente, on observe que l'entête
      // *** de ce sous-menu comporte une icône blanche.
      // *** L'instruction suivante modifie donc cette icône d'en-tête du sous menu, car il n'est pas possible de le faire
      // *** via le fichier XML
       menu.getItem(0).getSubMenu().setHeaderIcon(R.drawable.option_white);

      return true;
    }
```

```
    //Méthode qui se déclenchera au clic sur un item
    public boolean onOptionsItemSelected(MenuItem item) {
      //On regarde quel item a été cliqué grâce à son id et on déclenche une action
      switch (item.getItemId()) {
        case R.id.option:
          // *** On affiche un message dans un "Toast"
          // *** Un "Toast" est un message dit transitoire car il ne nécessite aucune intervention
          // *** de la part de l'utilisateur et ne prend même pas le focus: dans le cas où l'utilisateur
          // *** serait en train d'effectuer une saisie, la saisie continuera dans ce même champ durant
          // *** tout le temps de l'affichage du message. Enfin, le message disparait de lui-même.
          // *** Le toast de base a besoin de 3 paramètres: Généralement le contexte de l'application,
          // *** le texte à afficher et enfin la durée d'affichage (La durée pourra prendre deux valeurs:
          // *** Toast.LENGTH_SHORT ou Toast.LENGTH_LONG):
          Toast.makeText(Tutoriel10_Android.this, "Option", Toast.LENGTH_SHORT).show();
          return true;
        case R.id.favoris:
          Toast.makeText(Tutoriel10_Android.this, "Favoris", Toast.LENGTH_SHORT).show();
          return true;
        case R.id.stats:
          Toast.makeText(Tutoriel10_Android.this, "Stats", Toast.LENGTH_SHORT).show();
          return true;
       case R.id.quitter:
          //Pour fermer l'application il suffit de faire finish()
          finish();
          return true;
      }
      return false;}
}
```

UTILISER UNE BASE DE DONNÉES SQLite

Comme exemple, nous allons créer une mini base de données pour enregistrer des livres.

Android fournit la classe abstraite **SQLiteOpenHelper** permettant de gérer la création et la mise à jour de bases de données.

Une **classe abstraite** est une classe dont l'implémentation n'est pas complète et qui n'est pas instanciable. Elle sert de base à d'autres classes dérivées (héritées).

La classe **SQLiteOpenHelper** étant abstraite, il faut donc créer une classe fille qui en hérite :

```
package com.masociete.android.sqlite;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase.CursorFactory;

public class MaBaseSQLite extends SQLiteOpenHelper {

	private static final String CREATE_BDD = "CREATE TABLE table_livres (id "
	+ "INTEGER PRIMARY KEY AUTOINCREMENT, isbn TEXT NOT NULL, "
	+ "titre TEXT NOT NULL);";

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

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

	/* *** Mise à jour de la base ***
	@Override
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
	  // Cette méthode onUpgrade est appelée lorsque l'on ouvre la base de données et que son numéro de version
	  // a changé.
	  // Ce numéro de version est passé au constructeur de cette classe MaBaseSQLite (voir code de l'application)
	  // Ici, on décide qu'en cas de changement de version, on supprime la table puis on la recrée
	  db.execSQL("DROP TABLE table_livres;");
	  onCreate(db);
	}
}
```

Création / Suppression de la base de données

Nous allons voir que la base de données est créée par l'application. Cette création se fait au moment de la tentative d'ouverture de la base de données par l'application. En effet, si la base n'existe pas au moment de son ouverture par l'application, la méthode **onCreate** ci-dessus est appelée.

D'autre part, en cas de désinstallation d'une application, la base de données créée par cette dernière est normalement détruite. D'ailleurs, une base de données créée par une application n'est accessible que par cette dernière.

Code Java de l'application :

```
package com.masociete.android.sqlite;

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

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

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

     /*** Création d'une instance de la classe MaBaseSQLite
      int VERSION_BDD = 1;
      MaBaseSQLite maBaseSQLite = new MaBaseSQLite(context, "eleves.db", null, VERSION_BDD);

     /*** Ouverture de la BDD en écriture, avec création si elle n'existe pas
     /** On appelle la méthode getWritableDatabase( ) de la classe MaBaseSQLite, qui en fait :
     /**     - appelle sa méthode onCreate (dans laquelle a été mise l'instruction de la création de la base,
     /**       dans le cas où la base n'existe pas.
     /**     - appelle sa méthode onUpgrade, dans le cas où la base a changé de version (le numéro de version a
     /**       été passé au constructeur de la classe MaBaseSQLite, avec la variable VERSION_BDD)
     SQLiteDatabase bdd=maBaseSQLite.getWritableDatabase( )

     /*** Exemple d'insertion de données dans une table
     ContentValues values = new ContentValues();
     values.put("isbn", 123456789);
     values.put("titre", "Programmation Android");
     bdd.insert("table_livres", null, values);

     /*** Deuxième façon d'insérer des données dans une table, en utilisant le langage SQL
    String req;
    req = "insert into table_livres("isbn", "titre") values (123456789, "Programmation Android";
    bdd.execSQL(req);

     /*** Exemple de modification de données dans une table
     ContentValues values = new ContentValues();
     values.put("isbn", 11114444);
     values.put("titre", "Programmation objet");
     bdd.update("table_livres", values, "id=124", null);

    /*** Recherche de plusieurs enregistrements, à l'aide d'un curseur
   String[ ] colonnes {"id", "isbn", "titre"};     → Spécification dans un tableau, des colonnes à rechercher
   String where = "titre LIKE \"Programmation\"";
   String groupBy = null;
   String having = null;
   String order = null;
    Cursor c = bdd.query("table_livres", colonnes, where, groupBy, having, order, null);
    while (c.moveToNext( ))
    {
      // ** Traitement de l'enregistrement lu. Par ex, le numéro isbn lu se trouve dans : c.getText("isbn")
    }
}
```

EXÉCUTER UN SCRIPT PHP SITUÉ SUR UN SERVEUR – LANCER 2 ACTIVITÉS – UTILISER UNE BARRE DE PROGRESSION DE TYPE "DURÉE INDETERMINÉE" ET AYANT LA FORME D'UN DISQUE TOURNANT – PARSER DU XML AVEC LA TECHNOLOGIE SAX

- **Exemple présenté :** Page de login, envoyant au serveur le login et le mot de passe saisis pour vérification.
- **Informations retournées par le script PHP :**

  Le script va renvoyer l'user id correspondant au login et au mot de passe saisis, ou un code erreur pouvant prendre les valeurs 1 (signifiant une erreur lors de la connexion à la base de données), ou 2 (signifiant une erreur sur l'instruction **mysql_select_db**), ou 3 (signifiant qu'il n'existe pas dans la base de données d'enregistrement correspondant au login et au mot de passe saisis).

  Ces informations seront renvoyées sous forme de flux XML, généré à l'aide des instructions suivantes :

```
echo '<?xml version="1.0"?>\n";
echo "<login>\n";
- - - - - - - -
//*** Instruction exécutée lorsque le login et le mot de passe saisis existent dans la base de données
printf('    <user id="%d"/>'."\n",$id);     → Fonctionnement de la function printf :
                                              Elle affiche la chaîne figurant dans son premier paramètre, après avoir
                                              remplacé %d par la variable figurant dans son 2ème paramètre, c'est-à-
                                              dire la variable $id, contenant en fait l'user id recherché dans la base de
                                              données.
- - - - - - - -
//*** Instruction exécutée si erreur
printf('    <error value="%d"/>'."\n".'</login>', $numErreur);
- - - - - - - -
echo "</login>";
```

- **Utilisation d'une deuxième activité :**

  Dans cet exemple, l'écran de login ne sera pas affiché dans l'activité principale, mais dans une deuxième. Quand le login sera effectué, cet écran de login se fermera et on affichera le message "User id *id-utilisateur* logged in" sur l'écran de l'activité principale.

  Il s'agit d'une technique que l'on peut utiliser lorsque l'application comporte plusieurs écrans.

- **L'écran de la page de login :**

  Bien qu'il soit affiché dans une deuxième activité, il est néanmoins défini dans le fichier **main.xml**.

- **Le code de l'activité principale :**

```
package de.demo.main;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;
import de.demo.login.Login;     → Comme la deuxième activité nécessite plusieurs classes, on a choisi de mettre ces classes
                                  de la deuxième activité dans un autre package, de nom "de.demo.login", et le nom de la
                                  classe principale de cette deuxième activité est "Login". L'avantage est que sous Eclipse,
                                  ces classes de la deuxième activité seront dans une branche différente de celle de la classe
                                  de l'activité principale. En effet, dans la partie gauche, sous la branche "src", on a une
                                  sous-branche par package. D'ailleurs, la création d'un nouveau package se fait par un clic
                                  droit sur la branche "src".

public class Main extends Activity
{
   private TextView tv;
  public static final int RESULT_Main = 1;

  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);

    //Appel de la page de Login
    startActivityForResult(new Intent(Main.this, Login.class), RESULT_Main);

    tv = new TextView(this);
    setContentView(tv);
}

  private void startup(Intent i) {
    // Récupère l'identifiant
    int user = i.getIntExtra("userid",-1);
```

```
    //Affiche les identifiants de l'utilisateur
    tv.setText("UserID: "+String.valueOf(user)+" logged in");
  }

  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if(requestCode == RESULT_Main && resultCode == RESULT_CANCELED)
      finish();
    else
      startup(data);
  }
}
```

Explications :

- Fonctionnement général :

On appelle la page de Login et on traite la réponse de cette dernière dans la fonction **onActivityResult**. Si l'authentification est correcte, la fonction **startup()** est appelée.

*A propos du 2ème paramètre passé à la fonction **startActivityForResult** :*

Ce paramètre, chargé avec la variable **RESULT_Main**, initialisée à 1, est en fait un numéro de code retour que l'on choisit.

Il sert surtout dans le cas où depuis notre activité principale, on soit amené à lancer 2 activités différentes.

On lancera la première avec par exemple un code retour à 1, et l'autre avec un code retour à 2.

Lorsque l'une ou l'autre de ces 2 activités va se terminer, c'est la même fonction **onActivityResult** qui va se déclencher.

Dans cette dernière, le traitement à effectuer sera la plupart du temps différent selon que ce soit la première activité ou la deuxième qui se soit terminée.

Comment saura-t-on quelle est l'activité qui vient de se terminer ? Et bien on le saura en testant la valeur du code retour, qui est renvoyée par l'activité qui vient de se terminer. D'où le test **if (requestCode == RESULT_Main)**

- Deuxième paramètre de la fonction **getIntExtra** :

Il s'agit de la valeur par défaut c'est-à-dire la valeur qui sera retournée dans le cas où la donnée "userid" ne serait pas retournée par la deuxième activité. Ce paramètre est ici renseigné arbitrairement avec la valeur -1, valeur qui ne sera pas en fait utilisée, car dans le cas où la deuxième activité ne renverrait rien, son code retour serait à **RESULT_CANCELED**, et la fonction **startup** ne serait pas appelée.

- **Le code de l'activité secondaire :**

```
package de.demo.login;

import java.io.IOException;
import java.io.InputStream;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.List;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.params.HttpConnectionParams;
import org.apache.http.protocol.HTTP;
import org.xml.sax.Attributes;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.ProgressDialog;
import android.content.Intent;
```

```
import android.os.Bundle;
import android.os.Looper;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import de.demo.main.R;

public class Login extends Activity
{
 public ProgressDialog progressDialog;
 private EditText UserEditText;
 private EditText PassEditText;
 public void onCreate(Bundle savedInstanceState)
 {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  //*** Initialisation d'une progress bar, qui a en fait l'allure d'un disque qui tourne
  //*** En passant la valeur "true" à la méthode setIndeterminate, on met la barre de progression dans un état
  //*** "indéterminé" et elle s'anime alors.
  //*** Si on lui passe la valeur "false", elle se retrouve à l'état "déterminé" et le disque s'arrête de tourner,
  //*** et reste à une certaine progression que l'on aura fixé avec la méthode setProgress
  //*** La valeur que l'on passe à la méthode setProgress est un entier compris entre 0 et 100.
  //*** Méthode setCancelable :
  //*** En lui passant la valeur "false", on fait en sorte que cette barre de progression ne puisse pas être fermée
  //*** par l'utilisateur. Dans la plupart des cas, on procédera ainsi car il n'est guère concevable qu'un utilisateur
  //*** puisse fermer une barre de progression.
  progressDialog = new ProgressDialog(this);
  progressDialog.setMessage("Please wait...");
  progressDialog.setIndeterminate(true);
  progressDialog.setCancelable(false);
  // Récupération des éléments de la vue définis dans le xml
  UserEditText = (EditText) findViewById(R.id.username);
  PassEditText = (EditText) findViewById(R.id.password);
  Button button = (Button) findViewById(R.id.okbutton);
  // Définition du listener du bouton
  button.setOnClickListener(new View.OnClickListener()
  {
     public void onClick(View v)
     {
      int usersize = UserEditText.getText().length();
      int passsize = PassEditText.getText().length();
      // si les deux champs sont remplis
      if (usersize > 0 && passsize > 0)
      {
       progressDialog.show();
       String user = UserEditText.getText().toString();
       String pass = PassEditText.getText().toString();
       // On appelle la fonction doLogin qui va communiquer avec le PHP
       doLogin(user, pass);
      }
      else
      {
```

```
        AlertDialog ad = new AlertDialog.Builder(this);
        ad.setPositiveButton("Ok",null);
        ad.setTitle("Erreur");
        ad.setMessage("Entrer SVP le nom et le mot de passe");
        ad.create( );
        ad.show( );
       }
     }
  });

 button = (Button) findViewById(R.id.cancelbutton);
  // Création du listener du bouton cancel (on sort de l'appli)
  button.setOnClickListener(new View.OnClickListener()
  {
    public void onClick(View v)
    {
      setResult(Activity.RESULT_CANCELED, null);
      finish();
    }
  });
}

private void doLogin(final String login, final String pass)
  final String pw = md5(pass);
  DefaultHttpClient client = new DefaultHttpClient();
  HttpConnectionParams.setConnectionTimeout(client.getParams(), 15000);
  HttpResponse response;
  HttpEntity entity;
  HttpPost post = new HttpPost("http://scripts.masociete.fr/login.php";)
  List<NameValuePair> nvps = new ArrayList<NameValuePair>();
  nvps.add(new BasicNameValuePair("username", login));
  nvps.add(new BasicNameValuePair("password", pw));
  post.setHeader("Content-Type", "application/x-www-form-urlencoded");
  // On passe les paramètres login et password qui vont être récupérés par le script PHP en post
  post.setEntity(new UrlEncodedFormEntity(nvps, HTTP.UTF_8));
  // On récupère le résultat du script
  response = client.execute(post);
  entity = response.getEntity();
  InputStream is = entity.getContent();

  SAXParserFactory spf = SAXParserFactory.newInstance();
  SAXParser sp;
  sp = spf.newSAXParser();
  XMLReader xr = sp.getXMLReader();

  // On declare un "ContentHandler" c'est à dire un gestionnaire de contenu XML
  // La classe LoginContentHandler est définie plus bas. C'est une classe créée ici pour notre application,
  // qui hérite de la classe "ContentHandler". Dans cette classe, on redéfinit la méthode "startElement", qui
  // s'exécutera en fait chaque fois que l'on tombe sur un nouvel élément XML, et la méthode "endElement"
  LoginContentHandler uch = new LoginContentHandler();

  // On affecte au XMLReader le "ContentHandler" précédemment créé
  xr.setContentHandler(uch);

  // Parsage du flux XML, maintenant que le XMLReader a le bon "ContentHandler"
  xr.parse(new InputSource(is));
```

Dans une autre application ("UTILISER UNE LISTVIEW", une boîte de message a été créée sans faire appel à cette méthode **create**. La raison en est que dans cette autre application, il avait été créé un objet **AlertDialog.Builder** et non un objet **AlertDialog** comme ici.

```
  // Fermeture de l'objet InputStream, et vidage du contenu de l'objet HttpEntity, si l'activité ne s'est pas
  // terminée lors du parsage, avec l'instruction finish figurant dans la méthode endElement du ContentHandler
  // C'est que l'on est dans le cas d'une erreur rencontrée. L'activité se terminera alors quand l'utilisateur cliquera
  // sur le bouton "Cancel"
  is.close();
  if (entity != null)
    entity.consumeContent();
}

private class LoginContentHandler extends DefaultHandler
{
  private Boolean in_loginTag= false;
  private int userID;
  private Boolean error_occured     = false;

  public void startElement(String n, String l, String q, Attributes a)     → Paramètres de la methode startElement :
                                                                              - Url de l'espace de nommage
                                                                              - Nom local de la balise
                                                                              - Nom de la balise en version 1.0 :
                                                                                  <code>nameSpaceURI+":"+Localname</code>
                                                                              - Attributs de l'élément, que l'on peut lire de la façon
                                                                                suivante :
                                                          for (int index = 0; index < a.getLength(); index++) {
                                                            System.out.println(a.getLocalName(index) + " = " + a.getValue(index));

    throws SAXException
  {

    if (l == "login")
      in_loginTag = true;
    if (l == "error")
    {
      progressDialog.dismiss();
      switch (Integer.parseInt(a.getValue("value"))
      {
        case 1:
          createDialog("Error", "Couldn't connect to Database");
          break;
        case 2:
          createDialog("Error", "Error in Database: Table missing");
          break;
        case 3:
          createDialog("Error", "Invalid username and/or password");
          break;
        }
        error_occured = true;
      }

      if (l == "user" && in_loginTag && a.getValue("id") != "")
        userID = Integer.parseInt(a.getValue("id"));
    }
```

```
  public void endElement(String n, String l, String q) throws SAXException
  {
    // on renvoie l'id si tout est ok
    if (l == "login")
    {
      in_loginTag = false;

      if (!error_occured)
      {
        progressDialog.dismiss();
        Intent i = new Intent();
        i.putExtra("userid", userID);
        setResult(Activity.RESULT_OK, i);
        finish( ):     → Fermeture de l'application
        }
      }
    }

  public void characters(char ch[], int start, int length)
  {
  }

  public void startDocument() throws SAXException
  {
  }
  public void endDocument() throws SAXException
  {
  }

 }

}
```

- **<u>Ajout à faire dans le fichier manifest pour que la communication avec le serveur web fonctionne</u> :**

  Entre la ligne **</application>** et la ligne **</manifest>**, mettre les lignes :

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