Installer l'environnement de développement Android

(http://developer.android.com/sdk/installing.html)

Pour développer une application Android il est préférable de se doter de l'**IDE Eclipse**. Google fournit un **SDK Android** (kit de développement) composé des outils nécessaires au développement mais attention ce n'est pas un IDE. Le SDK Android permet d'émuler un téléphone virtuel grâce au concept **AVD** (Virtuel Device Application).

Google fournit également un plugin (un complément logiciel) nommé **ADT** (Android Development Tools) permettant d'intégrer un environnement complet de développement pour Eclipse.

Le SDK Android nécessite l'installation du kit de développement Java : **JDK**.

Installer l'environnement la 1ère fois :

1. Créer votre répertoire « android »
2. Copier et installer le **kit de développement de java** (JDK) [de préférence Java SE Development Kit -*jdk 6u24_windows*-]
3. Copier et installer l'**atelier de développement Eclipse** *[Eclipse IDE for Java Developers-Helios SR1-]*
    - dézipper Eclipse dans **android/eclipse**
    - démarrer eclipse.exe
        - workspace : **android/appAndroid**
        - fermer Eclipse
4. Copier et installer le **SDK Android** (Kit de développement)
    - Copier le répertoire androidSDK *(version déjà installée : android-sdk_r08-windows)* dans **android/**
    - SINON *(à éviter car beaucoup trop long à faire)*
        - dézipper android-sdk dans androidSDK
        - démarrer **SDK Manager**
        - Installer tous les paquetages
5. **AVD (Virtuel Device Application)** : définit l'environnement d'un téléphone virtuel afin de tester les applications)
    - Démarrer **SDK Manager**
    - sélectionner à droite **Virtual Device**
    - Cliquer sur New
        - Name=formation
        - Target=android 2.2
        - cliquer sur create AVD
        - fermer SDK Manager
    - remarque : le répertoire virtuel du téléphone *(configuration et espace mémoire)* se trouve dans le répertoire **MesDocuments/.android/avd** *(cela change en fonction des environnements et des versions de Windows)*
    - Fermer SDK Manager

6. **ADT (Android Development Tools)** est un plugin pour l'IDE Eclipse. Il permet de créer sous Eclipse un projet Android complet (vue, contrôleur, paquetage). Pour l'installation de ADT suivez les étapes suivantes :
    - Copier le fichier Zip **ADT-x.y.z** *(9.0.0 pour la version actuelle) dans* ***android/***
    - Démarrer **Eclipse**
    - Help + Install New Software...
    - Cliquer sur Add
    - Dans la fenêtre de dialogue : cliquer sur Archive
    - Sélectionner le fichier zip **ADT-x.y.z**
    - Cliquer sur Ok
    - **Attention :** décocher « Contact all update sites ... » en bas de la fenêtre
    - Cliquer sur Select All puis cliquer sur Next, par la suite sélectionner tous les outils proposés si nécessaire
    - Lire et accepter la licence et cliquer sur Finish
    - Quand l'installation est complète : redémarrer Eclipse
7. **Lier Eclipse avec le SDK Android** afin de bénéficier des outils de développement fournit avec le kit SDK
    - Choisir Window + Preferences
    - Sélectionner **Android**
    - dans **SDK Location** : indiquer le chemin du répertoire **androidSDK** déjà installé
    - vous devriez avoir ?\android\androidSDK\android-sdk-windows
    - Cliquer sur **Apply**, s'affiche alors la liste des environnements *(des versions)* d'Android disponibles
    - Cliquer sur Ok

Remarque : les mises à jour du SDK android et du plugin android pour Eclipse se font par internet. Il est vivement conseiller de mettre à jour l'un et l'autre dans le même temps. L'ensemble des mises à jour peut être réalisées sous Eclipse.

- *Pour le plugin: Help + Check for Update*
- *Pour le SDK android : window + Android Sdk et AVD + Installed Package + Update All*

Pour connaître l'état des nouvelles versions pour Android pensez à consulter la page du site Web consacrée au SDK android : http://developer.android.com/sdk/index.html

Hello Android : prise en main de l'atelier

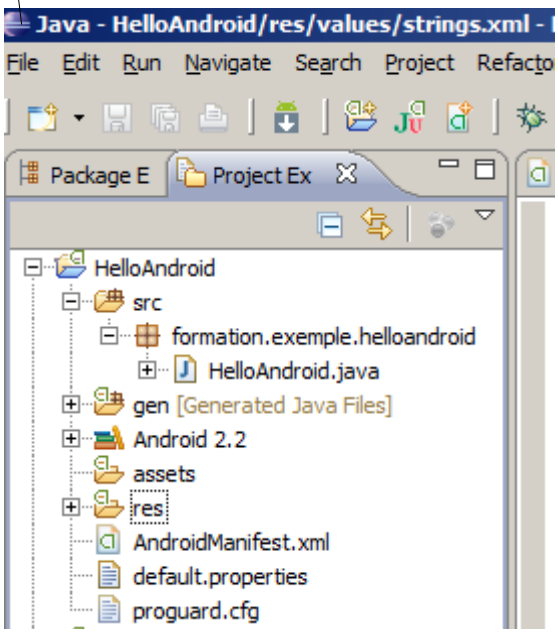
Cette première application va nous permettre de :

- Créer un nouveau projet Android
- Créer un contrôleur : **Activity**
- Démarrer une application Android
- Utiliser une vue au format XML **(layout)**

Créer un nouveau projet Android sous Eclipse

1. **File + New + Project**
2. Choisir « Android Project » et cliquer sur **Next**
3. Remplir les champs du formulaire
    - *Project Name :* HelloAndroid
    - *Build Target :* Choisir **Android 2.2**
    - *Application Name :* Hello, Android
    - *Package Name :* formation.exemple.helloandroid
    - *Create Activity :* HelloAndroid
    - *Min SDK Version :* 8
4. Cliquer sur **Finish**

Vue sur l'explorateur de paquetage : HelloAndroid



Quelques précisions :

Project Name :
c'est le nom du répertoire qui contiendra les fichiers projet.

Build Target :
indique la version Android à utiliser, correspond à un ensemble de paquetages disponibles (API)

Application Name :
c'est le titre de l'application qui sera affichée sur le téléphone

Package Name :
*chaque paquetage sous Android doit disposer d'**un nom unique**. L'espace de nommage d'un paquetage correspond à celui utilisé avec le langage de programmation Java. Les classes « Activity » seront installées sous ce paquetage.*

Create Activity :
c'est la classe « Activity » de démarrage qui sera générée par Eclipse. Au départ d'une application au moins une « Activity » est nécessaire. Elle peut ou non proposer une interface utilisateur.

Min SDK Version:
cette valeur spécifie la version Android minimum permettant d'exécuter l'application.
Voir également *http://developer.android.com/guide/appendix/api-levels.html*

La classe Activity

Voici le code généré au moment de la création de votre projet :

```
package formation.exemple.helloandroid;
import android.app.Activity;
import android.os.Bundle;
public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

La classe **HelloAndroid** hérite de **Activity**. *Activity* est la classe contrôleur capable de réaliser des actions sous le système Android. Une *Activity* est nécessaire pour réaliser une interface utilisateur.

Démarrer l'application

1. Run + Run
2. Sélectionner « Android Application »

*Le plugin d'Eclipse crée une version exécutable de votre application et charge **Android Emulator**.*
Une fois l'émulateur démarré cliquer sur « Menu » pour voir votre application.

*L'application utilise une vue par défaut **main.xml**, ce fichier est placé dans le dossier **res/layout**.*

Android XML Layout : construire une interface utilisateur

Google préconise de séparer traitement et interface utilisateur en utilisant le principe de **vue**, appelé **layout** sous Android.
Définir une vue de manière externe au programme permet de séparer la logique applicative et la présentation (interface utilisateur) *[principe du modèle-vue-contrôleur]*.

1. Ouvrir le fichier main.xml : dossier **res/layout** de votre projet.

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

*Chaque composant correspond à un objet issue de la classe **View**. Chaque sous-classe issue de la classe **View** peut donc être décrit au format **xml**. Ici **TextView** et une sous-classe de la classe **View**.*

Quelques précisions :

xmlns:android :
doit être présent, indique que le composant fait référence aux attributs définis dans le projet.

android:id :
permet d'assigner un identifiant unique au composant, cet identifiant servira au niveau du code source de l'application.

android:layout_width :
largeur du composant View sur l'écran. Ici il prendra toute la place disponible *(fill_parent)*.

android:layout_height :
comme pour le précédent mais en hauteur.

android:text :
permet de définir le texte à afficher soit directement soit de manière externe.
*Ici on utilise une ressource externe, ce qui permettra éventuellement de mettre en place une application multi-langues. La chaine **hello** est définie dans **res/value/string.xml**.*

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, HelloAndroid!</string>
    <string name="app_name">Hello, Android</string>
</resources>
```

2. Ouvrir le fichier **res/value/string.xml** et modifiez le comme ceci :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, Dire Bonjour</string>
    <string name="app_name">Hello, Android</string>
</resources>
```

3. Tester votre « nouvelle » application

La classe R : une classe générée par Eclipse

La classe R définit pour le projet les identifiants de tous les composants ou attributs manipulables par programmation ou référencés dans le/les fichiers **xml** du projet.

4. Ouvrir : `gen/[nom du paquetage]/R.java`

```
package formation.exemple.helloandroid;

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

Remarque : ce fichier ne doit **jamais** être modifié par le programmeur car il est généré automatique par Eclipse.

Hello Button : LinearLayout, composant, écouteur

HelloButton va nous permettre de :

- Créer un nouveau projet Android
- Créer un contrôleur : **Activity**
- Utiliser un **Linear Layout**
- Utiliser un bouton de commande **Button**
- Utiliser un **TextView**
- Activer un écouteur sur **Button**
- Visualiser la hiérarchie d'une vue

Créer un nouveau projet Android sous Eclipse

1. **File + New + Project**
2. Choisir « Android Project » et cliquer sur **Next**
3. Remplir les champs du formulaire
    - *Project Name :* HelloButton
    - *Build Target :* Choisir **Android 2.2**
    - *Application Name :* Hello, Button
    - *Package Name :* formation.exemple.hellobutton
    - *Create Activity :* HelloButton
    - *Min SDK Version :* 8
4. Cliquer sur **Finish**

Layout : LinearLayout

<LinearLayout> indique que l'affichage de chaque composant ***(View)*** du conteneur doit se faire de manière linéaire (verticalement et horizontalement). La balise <Linear Layout> est gérée par un objet **LinearLayout** qui est un **ViewGroup.** *Un ViewGroup est une objet View spécial capable de contenir d'autres objets View.*

1. Ouvrir le fichier main.xml (dossier **res/layout** de votre projet), et modifier le comme ceci :

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

<Button android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Dit Bonjour!"/>
<TextView
      android:id="@+id/textview1"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text=""
      />
</LinearLayout>
```

Quelques précisions :

android:layout_height :
"wrap_content" : la hauteur du bouton est fonction du contenu de celui-ci.

Button et TextView

Les composants **Button** et **TextView** sont des widgets (paquetage `android.widget`). Tous les widgets sont des sous-classes de **View**.

Ici chaque widget dispose d'un identifiant car nous allons les utiliser dans notre application.

Ecouteur d'évènement : un clic sur Button

Pour réagir à un événement *(un clic par exemple)* chaque widget doit indiquer explicitement le ou les évènements pris en charge par le composant.

```
/** écouteur */
       monBouton.setOnClickListener(this);
```

Modifier le contenu d'un TextView

Chaque widget peut-être modifié par programme. Pour cela il est nécessaire de créer un objet ayant un lien avec le composant affiché.

```
TextView textview1=(TextView)findViewById(R.id.textview1);
textview1.setText("Bonjour Thierry");
```

Activity HelloButton

2. Ouvrir le fichier **HelloButton.java** et modifier le comme ceci :

```
package formation.exemple.hellobutton;

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

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

        /** écouteur */
        Button monBouton=(Button) findViewById(R.id.button1);
        monBouton.setOnClickListener(this);
    }

    public void onClick(View v) {
            TextView textview1=(TextView)findViewById(R.id.textview1);
            textview1.setText("Bonjour Thierry");
      }
}
```

Quelques précisions :

setContentView (R.layout.main) :
méthode permettant l'affichage de la vue désignée par son identifiant R.layout.main*) (identifiant généré automatiquement dans la classe R).*

findViewById(R.id.button1) :
méthode permettant de retrouver l'identifiant d'un objet View et d'instancier un objet View. (Button) permet de « caster » l'objet View en Button.

monBouton.setOnClickListener(this) :
méthode permettant d'activer un écouteur ***OnClickListener*** *pour* **monBouton**. ***this*** *doit être de type* ***View.OnClickListener*** *car la méthode* ***setOnClickListener*** *reçoit un paramètre de type* ***View.OnClickListener. onClick(View v)*** *est la méthode qui sera appelée au moment du clic.* ***abstraite qui doit*** *La méthode appelée est une redéfinition de la méthode* ***abstraite*** *onClick(View v).*

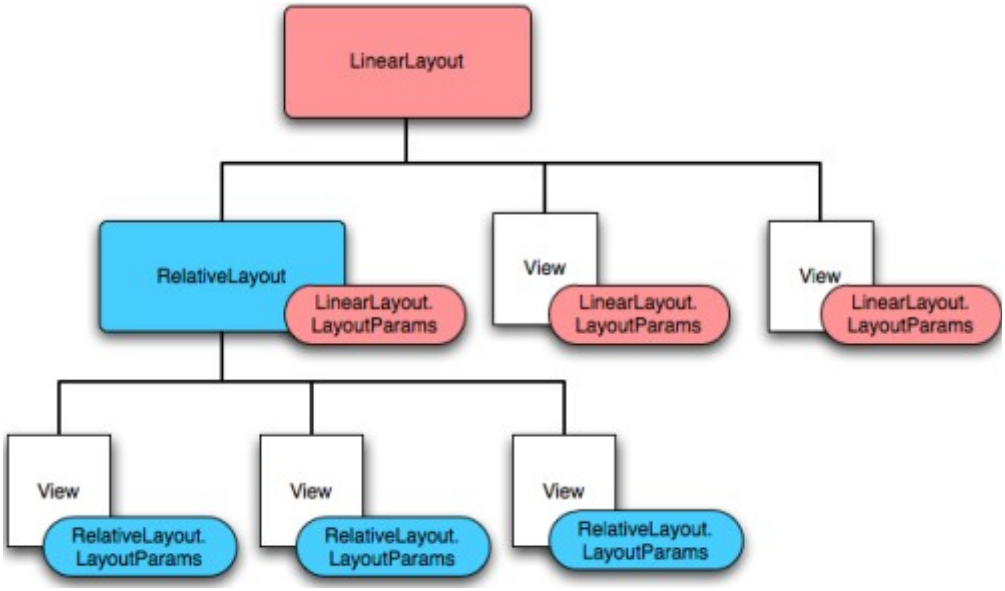
public void onClick(View v) :
méthode abstraite de la classe ***View.OnClickListener,*** *elle doit être redéfinie obligatoirement dans la sous-classe* ***(ici HelloButton).*** *Le paramètre* ***v*** *retourne un objet* ***View*** *correspondant au composant cliqué.*

implements View.OnClickListener :
la clause ***implements*** *permet d'étendre le type de la classe. La classe* ***interface*** **View.OnClickListener** *impose un gabarit sur l'action clic sous la forme d'une méthode abstraite[****onClick(View v)****] qu'il faudra redéfinir.*

3. Démarrer et tester l'application *HelloButton*

L'application HelloButton

Exemple de hiérarchie d'une vue



Hello EditText : RelativeLayout, EditText, écouteur

HelloEditText va nous permettre de :

- Créer un contrôleur : **Activity**
- Utiliser un **RelativeLayout**
- Utiliser deux **Button(s)**
- Utiliser **EditText, TextView**
- Activer un écouteur sur **chaque Button**
- Utiliser le paramètre **v** de **onClick(View v)**

2. Créer le projet HelloEditText *(remplir le formulaire du projet selon les principes des précédents projets)*

Layout : RelativeLayout

<RelativeLayout> indique que l'affichage de chaque composant ***(View)*** du conteneur est positionnable par rapport au conteneur ou par rapport à un autre objet View. La balise est gérée par un objet **RelativeLayout** qui est un **ViewGroup.** *Un ViewGroup est une objet View spécial capable de contenir d'autres objets View.*

3. Ouvrir le fichier main.xml (dossier **res/layout** de votre projet), et modifier le comme ceci :

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:background="#00aa00"
      android:padding="10px" >
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textColor="#000000"
        android:text="Saisir un texte:"/>
    <EditText
        android:id="@+id/mytext"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/label"/>
    <Button
        android:id="@+id/ok"
        android:layout_width="100dip"
        android:layout_height="wrap_content"
        android:layout_below="@id/mytext"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="OK" />
    <Button
      android:id="@+id/cancel"
        android:layout_width="100dip"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="Annuler" />
    <TextView
        android:id="@+id/affiche"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/cancel"
        android:textColor="#aa0000"/>
        />
</RelativeLayout>
```

Quelques précisions :

android:layout_width :
"100dip" : permet de donner une taille précise à un composant.

android:layout_below :
"@id/mytext" le composant doit se positionner sous le champ ***mytext.***

android:toLeftOf :
"@id/ok" le composant doit se positionner à gauche du ***Button ok.***

EditText

Le composant **EditText** permet la saisie d'un texte. [paquetage `android.widget`].
C'est un **TextView** éditable.

Ecouteur d'évènement : un clic sur Button

Pour réagir à un événement *(un clic par exemple)* chaque widget doit indiquer explicitement le ou les évènements pris en charge par le composant.

```
/** écouteur */
        buttonOk.setOnClickListener(this);
        buttonCancel.setOnClickListener(this);
```

L'évènement onClick : `public void onClick(View v)`

Le paramètre **v** donne des informations sur le composant cliqué ; notamment son identifiant.

```
switch (v.getId())
{       case R.id.ok :
                myAffiche.setText(myText.getText());
                break;
        case R.id.cancel :
                myAffiche.setText("");
                myText.setText("");
                break;
}
```

La partie verte correpond à l'application HelloEditText

Activity HelloButton

3. Ouvrir le fichier **HelloEditText.java** et modifier le comme ceci :

```
package formation.exemple.helloedittext;

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

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

        /** écouteur */
        Button buttonOk=(Button) findViewById(R.id.ok);
        buttonOk.setOnClickListener(this);
        Button buttonCancel=(Button) findViewById(R.id.cancel);
        buttonCancel.setOnClickListener(this);
    }

    public void onClick(View v) {
             EditText myText=(EditText)findViewById(R.id.mytext);
             TextView myAffiche=(TextView)findViewById(R.id.affiche);
             switch (v.getId())
             {      case R.id.ok :
                           myAffiche.setText(myText.getText());
                           break;
                    case R.id.cancel :
                           myAffiche.setText("");
                           myText.setText("");
                           break;
             }

      }
}
```

Quelques précisions :

setContentView (R.layout.main) :
méthode permettant l'affichage de la vue désignée par son identifiant `R.layout.main` *(identifiant généré automatiquement dans la classe R).*

findViewById(R.id.ok) :
méthode permettant de retrouver l'identifiant d'un objet View et d'instancier un objet View. (Button) permet de « caster » l'objet View en Button.

buttonOk.setOnClickListener(this) :
méthode permettant d'activer un écouteur ***OnClick*** *pour* ***boutonOk.***
this *doit être de type* ***View.OnClickListener.*** *En effet la méthode qui sera appelée sur le clic doit être implémentée dans une classe de type* ***View.OnClickListener.***
La classe ***interface*** *impose un gabarit sur l'action clic, la méthode appelée est une redéfinition de la méthode* ***abstraite*** *onClick(View v).*

public void onClick(View v) :
méthode abstraite de la classe ***View.OnClickListener,*** *elle doit être redéfinie obligatoirement dans la sous-classe* ***(ici HelloEditText).***
Le paramètre ***v*** ***r****etourne un objet* ***View*** *correspondant au composant cliqué.*
Pour de nombreux objets ***View*** *c'est la méthode* ***onClick()*** *qui est appelée sur l'évènement « clic ».*

implements View.OnClickListener :
la clause ***implements*** *permet d'étendre le type de la classe.*
La classe implémentée est une classe ***interface.***

v.getId():
cette méthode retourne l'identifiant du composant cliqué.

myText.getText():
la méthode ***getText()*** *retourne le contenu d'un champ.*

myAffiche.setText(myText.getText():
la méthode ***setText()*** *permet de mémoriser une chaine de caractères et de l'afficher.*
Le type du paramètre peut-être ***String*** *[qui* ***utilise*** *notamment la classe interface* **CharSequence***]*

R.id.cancel:
Le bouton « cancel » efface le contenu du champ éditable et du champ label.

```
myAffiche.setText("");
myText.setText("");
```

4. Démarrer et tester l'application HelloEDitText

Hello ListView : ListView, Toast, Adapter, écouteur

HelloListView va nous permettre de :

- Créer un contrôleur : **ListActivity**
- Utiliser un **ListView**
- Activer un écouteur sur **ListView**
- Créer un fichier **XML** décrivant chaque item de la liste
- Ajouter une méthode dans une classe
- Manipuler un tableau <**String**>
- Utiliser **Toast**

1. Créer le projet HelloListView *(remplir le formulaire du projet selon les principes des précédents projets)*

Organisation du layout main.xml

Le layout **main.xml** peut-être organisé de façon à présenter soit une liste soit un message indiquant que la liste est vide.

2. Ouvrir le fichier main.xml (dossier **res/layout** de votre projet), et modifier le comme ceci :

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

    <ListView android:id="@+id/android:list"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
      <TextView android:id="@+id/android:empty"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/no_notes"/>
</LinearLayout>
```

Quelques précisions :

<ListView android:id="@+id/android:list">:
précise les propriétés du composant ***ListView.***

<TextView android:id="@+id/android:empty">:
indique que la liste sera remplacé par le message « no_notes » si la liste est vide.
Le message est précisé dans ***string.xml***

3. Ouvrir le fichier string.xml (dossier **res/values** de votre projet), et modifier le comme ceci :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, HelloListView!</string>
    <string name="app_name">Hello ListView</string>
     <string name="no_notes">Aucun pays</string>
</resources>
```

Organisation du layout list_item.xml

Le layout **list_item.xml** décrit l'organisation de chaque ligne du **ListView**.
*Un ou plusieurs **TextView** peuvent être présents sur une ligne. Chaque **TextView** affichant une information relative à la ligne.*
[par exemple si on souhaite afficher le nom et le prénom d'une liste de personnes il faudra deux TextView}.

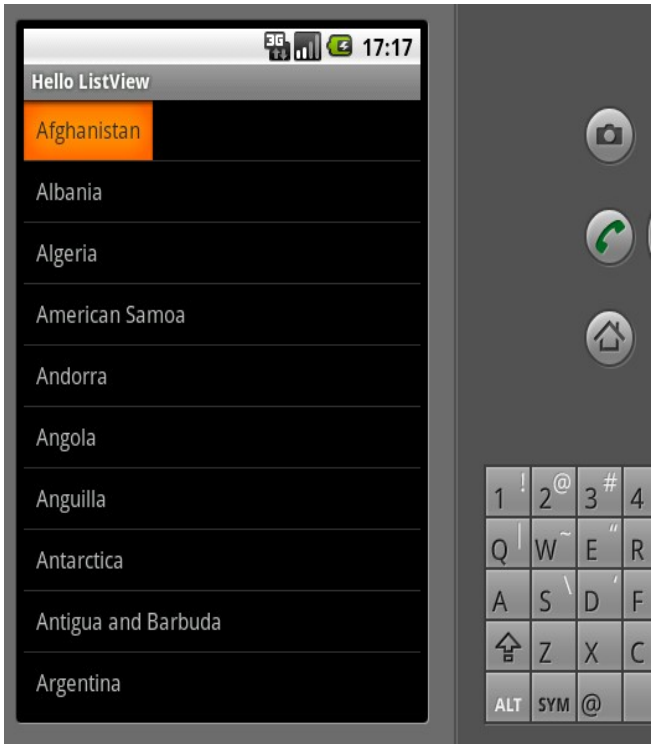
4. Créer le fichier list_item.xml (dossier **res/layout** de votre projet), et modifier le comme ceci :

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="10dp"
    android:textSize="16sp" >
</TextView>
```

Quelques précisions :

android:padding *:*
*"10dp" défini la marge intérieure du **TextView**.*

android:textSize:
"16sp" il est possible de modifier la taille des caractères affichés



ListActivity HelloListView

5. Ouvrir le fichier **HelloListView.java** et modifier le comme ceci :

```
package formation.exemple.hellolistview;

import android.app.ListActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

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

        String[] countries=getCountries();
        ArrayAdapter<String> listCountries=new ArrayAdapter<String>(this,
        R.layout.list_item, countries);

        setListAdapter(listCountries);

        ListView lv = getListView();
        lv.setTextFilterEnabled(true);

        /** écouteur */
      lv.setOnItemClickListener(this);
      }

      public void onItemClick(AdapterView<?> ad, View v, int pos, long id) {
            // When clicked, show a toast with the TextView text
        TextView item=(TextView) v;
        CharSequence texte=item.getText();
        Toast toast = Toast.makeText(getApplicationContext(),texte,Toast.LENGTH_SHORT);
        toast.show();
      }

      private String[] getCountries()
        { return  new String[] {
                "Afghanistan", "Albania", "Algeria", "American Samoa", "Andorra",
                "Angola", "Anguilla", "Antarctica", "Antigua and Barbuda", "Argentina",
                "Armenia", "Aruba", "Australia", "Austria", "Azerbaijan",
                "Bahrain", "Bangladesh", "Barbados", "Belarus", "Belgium",
                "Belize", "Benin", "Bermuda", "Bhutan", "Bolivia",
                "Bosnia and Herzegovina", "Botswana", "Bouvet Island", "Brazil", "British Indian Ocean Territory",
                "British Virgin Islands", "Brunei", "Bulgaria", "Burkina Faso", "Burundi",
                "Cote d'Ivoire", "Cambodia", "Cameroon", "Canada", "Cape Verde",
                "Cayman Islands", "Central African Republic", "Chad", "Chile", "China",
                "Christmas Island", "Cocos (Keeling) Islands", "Colombia", "Comoros", "Congo",
                "Cook Islands", "Costa Rica", "Croatia", "Cuba", "Cyprus", "Czech Republic",
                "Democratic Republic of the Congo", "Denmark", "Djibouti", "Dominica", "Dominican Republic",
                "East Timor", "Ecuador", "Egypt", "El Salvador", "Equatorial Guinea", "Eritrea",
                "Estonia", "Ethiopia", "Faeroe Islands", "Falkland Islands", "Fiji", "Finland",
                "Former Yugoslav Republic of Macedonia", "France"};

      }
}
```

Quelques précisions :

String[] countries=getCountries() :
String[] permet de déclarer un tableau de type ***String.***

private String[] getCountries() :
getCountries() est une méthode (une fonction ici) qui retourne un tableau de type ***String.*** *Il est possible de créer un tableau en utilisant la syntaxe suivante :*

```
new String[]{"Afghanistan", "Albania", "Algeria",...};
```

ArrayAdapter (Context context, int resource, T[] objects) :
un objet générique de la classe interface ***ListAdapter.*** *Il permet de créer un objet* ***ArrayAdapter*** *à partir d'un tableau et d'un fichier XML (*ici **list_item.xml***) qui décrit chaque ligne d'un* ***ListView.***
paramètres
context : le contexte de l'application [this]
ressource : l'identifiant du layout décrivant l'affichage [R.layout.list_item]
T[]objects : le tableau à afficher [countries]

setListAdapter(listCountries) :
la méthode de la classe ***ListActivity*** *fournie un objet* ***ListAdapter [ArrayAdapter*** *listCountries] à l'objet* ***ListView.***

ListView lv = getListView() :
la méthode retourne le ***ListView*** *courant du contrôleur [ListActivity].*

lv.setTextFilterEnabled(true) :
la méthode active ou désactive un filtre pour ***ListView.***
lv : ListView courant

lv.setOnItemClickListener(this) :
la méthode active un écouteur sur le ***ListView.*** *Chaque item devient cliquable.*
lv : ListView courant
this : *doit être de type* ***AdapterView.OnItemClickListener.*** *La classe* ***interface*** *impose un gabarit sur l'action clic, la méthode appelée est une redéfinition de la méthode* ***abstraite*** onItemClick(AdapterView<?> ad, View v, **int** pos, **long** id).

public void onItemClick(AdapterView<?> ad, View v, int pos, long id):
méthode abstraite de la classe ***AdapterView.OnItemClickListener,*** *elle doit être redéfinie obligatoirement dans la sous-classe* ***(ici HelloListView).***
Les paramètres reçus sont en rapport avec la ligne cliqué et le ListView.

Implements AdapterViewOnItemClickListener :
la clause ***implements*** *permet d'étendre le type de la classe.*

AdapterView :
une classe utilisée pour créer un objet **View** contenant des *widgets* nécessitants un **Adapter** [typiquement des listes, des tableaux, des galeries d'images, etc.]

Toast :
Toast est une classe permettant d'afficher plus ou moins longtemps un message simple.

Hello WebServiceDataBase : ListView, web service

HelloWebServiceDataBase va nous permettre de :

- Utiliser un composant du paquetage plum.webservice.database **: PlumDataBase**
- Ajouter une autorisation dans **AndroidManifest.xml**
- Manipuler le type **Cursor et** le lier à un **layout**
- Créer un contrôleur : **Activity**
- Utiliser un **layout** disposant d'un **ListView** et d'un **bouton.**
- Créer un fichier **XML** décrivant chaque item de la liste
- Ajouter une méthode dans une classe

1. Créer le projet HelloWebServiceDataBase *(remplir le formulaire du projet selon les principes des précédents projets)*

Organisation du layout main.xml

Le layout **main.xml** peut-être organisé de façon à présenter à la fois une liste et un bouton.

2. Ouvrir le fichier main.xml (dossier **res/layout** de votre projet), et modifier le comme ceci :

```
<?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="wrap_content"
              android:orientation="vertical" >
       <LinearLayout
             android:orientation="vertical"
             android:layout_width="fill_parent"
             android:layout_height="fill_parent"
             android:layout_weight="1">
            <ListView
            android:id="@android:id/list"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
             />
       </LinearLayout>
       <LinearLayout
                  android:orientation="vertical"
                  android:layout_width="fill_parent"
                  android:layout_height="fill_parent"
                  android:layout_weight="0">
         <Button android:id="@+id/ajouter"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Ajouter" />
      </LinearLayout>
 </LinearLayout>
```

Quelques précisions :

<LinearLayout … >
pour permettre l'affichage d'une liste et d'un bouton de commande permettant un défilement de la liste et un emplacement fixe pour le bouton il faut mettre en place un ***layout*** *contenant deux layouts : un pour* ***ListView,*** *un pour* ***Button.***

android:layout_weight="0":
la propriété ***weight*** *permet de préciser le « poids » de chaque layout de façon à ce que chacun d'un « négocie » plus ou moins de place à l'affichage.*

android:id="@android:id/list" :
il faut donner un nom à la liste afin de pouvoir l'utiliser de façon explicite dans le contrôleur (Activity).

Organisation du layout list_row.xml

Le layout **list_row.xml** décrit l'organisation de chaque ligne du **ListView**.
*Un ou plusieurs **TextView** peuvent être présents sur une ligne. Chaque **TextView** affichant une information relative à la ligne.*
[par exemple si on souhaite afficher le nom et le prénom d'une liste de personnes il faudra deux TextView}.

4. Créer le fichier list_row.xml (dossier **res/layout** de votre projet), et modifier le comme ceci :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
<TextView android:id="@+id/id"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:padding="10dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
   <TextView android:id="@+id/nom"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
 </LinearLayout>
```

Quelques précisions :

android:padding :
*"10dp" défini la marge intérieure du **TextView**.*

android:textSize:
"16sp" il est possible de modifier la taille des caractères affichés

<TextView android:id="@+id/id" et <TextView android:id="@+id/nom"
*deux TextView font présents car on souhaite afficher l'identifiant et le nom de chaque ligne de la table **table1**.*

Ajouter le paquetage plum.webservice.database

*Pour permettre à votre application d'accéder au **webservice** il faut utiliser un paquetage déjà développer (**plum.webservice.database**), dans lequel se trouve la classe **PlumDataBase.***

5. Procéder comme suit :

- *Copier le répertoire **plum** dans votre environnement de travail*
- *cliquer droit sur votre projet + New + Source Folder, afin d'ajouter un répertoire à paquetages. Nommez le **plum**.*
- *Sélectionner le répertoire*
- *cliquer File + New Package, nommez le **plum.webservice.database***
- *ajouter le fichier PlumDataBase.java,*
  - *sélectionner le paquetage **plum.webservice.database***
  - *cliquer droit + Import + File System + Browse **plum***
  - *cocher **PlumDataBase.java,** puis cliquer sur **Finish***

HelloWebServiceDataBase : PlumDataBase, Cursor

6. Ouvrir le fichier **HelloWebServiceDataBase.java** et modifier le comme ceci :

```
package formation.exemple.helloplumservicedatabase;

import plum.webservice.database.PlumDataBase;
import android.app.Activity;
import android.database.Cursor;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;

public class HelloPlumWebServiceDataBase extends Activity
{
      PlumDataBase webdata;

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

      webdata=new PlumDataBase
      ("http://10.0.2.2/plumwebservice/plumWebServiceDataBase.php");

      listTable1();
    }

    public void listTable1(){
      ListView lvTable1 = (ListView) findViewById(android.R.id.list);
      Cursor c=webdata.query("select * from table1 order by nom");

        String[] from = new String[] { c.getColumnName(0),c.getColumnName(1) };
        int[] to = new int[] {R.id.id , R.id.nom,};

        // Now create an array adapter and set it to display using our row
        SimpleCursorAdapter liste =
        new SimpleCursorAdapter(this, R.layout.list_row, c, from, to);

        lvTable1.setAdapter(liste);
    }
}
```

7. Modifier **AndroidManifest.xml** en ajouter dans <manifest la balise suivante :

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

ce qui donne ceci :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="formation.exemple.helloplumservicedatabase"
      android:versionCode="1"
      android:versionName="1.0">
        <uses-permission android:name="android.permission.INTERNET" />
      <application android:icon="@drawable/icon"    android:label="@string/app_name">
…….[le reste est identique]
```

Cette indication donne l'autorisation à l'application d'accéder à internet, si la balise n'est pas présente la connexion échoue.

Quelques précisions :

PlumDataBase (paquetage plum.webservice.database) :
Cette classe permet d'accéder à une base de données (MySql) grâce à un web service.
Il permet :

- *d'indiquer l'emplacement du webservice chargé d'interroger (Select) ou d'exécuter une requête de mise à jour (Insert, Update).*

```
webdata=new
PlumDataBase("http://10.0.2.2/plumwebservice/plumWebServiceDataBase.php");
```

- *de retourner un* ***Cursor*** *grâce à la méthode* ***query.***

```
Cursor c=webdata.query("select * from table1 order by nom");
```

- *d'exécuter des requêtes SQL de mises à jour.*

```
String sql=new String("delete from table1 where _id=12");
webdata.execute(sql);
```

Remarque : l'émulateur fournit avec le SDK Android est en fait un système Linux émulé par Qemu. L'émulateur fait donc tourné un second système qui dispose de sa propre adresse localhost [10.0.2.2]. Donc on ne peut pas accéder à la boucle locale 127.0.1.1 à partir de l 'émulateur. Pour l'accès à internet aucun problème car l'émulateur crée un pont entre notre connexion internet et lui-même.

private void listeTable1() :
une méthode que l'on ajouter à la classe. Elle est chargée d'interroger la table ***table1,*** *et de préparer un ListView affichant l'identifiant et le nom des personnes présentes dans* ***table1.***

SimpleCursorAdapter (Context context, int layout, Cursor c, String[] from, int[] to) :
un objet générique de la classe interface ***ListAdapter.*** *Il permet de créer un objet* ***ListAdapter*** *à partir d'un curseur et d'un fichier XML (*ici **list_row.xml***) qui décrit chaque ligne d'un* ***ListView.***
paramètres
context : le contexte de l'application [this]
layout : l'identifiant du layout décrivant l'affichage [R.layout.row_item]
Cursor : le curseur généré par ***webdata.query.*** *Ce curseur doit* ***absolument*** *posséder un champ* ***_id*** *pour pouvoir être couplé avec un* ***ListView.***
from : un tableau **String** indiquant les colonnes du curseur à utiliser
to : un tableau **int** indiquant l'identifiant des TextView(s) à utiliser dans le layout list_row.xml. La View n prend sa valeur dans la colonne 1 : to[n] ← *from[n].*

String[] from = new String[] { c.getColumnName(0),c.getColumnName(1) } :
le vecteur ***from*** *aura la forme suivante :* ***from[0]-> '_id'*** *et* ***from[1]->'nom'***

Liste des noms présents dans table1

HelloWebServiceDataBase : web service, Cursor, startActivity

On veut pouvoir ajouter, modifier et supprimer des noms dans la table **table1**. Pour cela nous allons devoir créer plusieurs **Activity**. Cette division des tâches est conforme au indication de bonne pratique dans le cadre d'un développement sous Android.

HelloWebServiceDataBase va nous permettre de :

- Démarrer une **Activity** avec **startActivity**
- Utiliser un composant du paquetage plum.webservice.database **: PlumDataBase**
- Manipuler le type **Cursor**
- Terminer une **Activity**
- Créer le contrôleur [Activity] newTable1.java
- Créer le contôleur ModOrSupTable1.java

1. Reprendre et compléter le projet HelloWebServiceDataBase

L'Activity AddNewTable1.java : Ajouter un nom dans table1

Organisation du layout newtable1.xml

Le layout **newtable1.xml** permet la saisie d'un nouveau nom.

2. Ouvrir le fichier newTable1.xml (dossier **res/layout** de votre projet), et modifier le comme ceci :

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
     android:background="#80FF00"
     android:padding="10px" >
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textColor="#000000"
        android:text="Saisir un nom:"/>
    <EditText
        android:id="@+id/mytext"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/label"/>
    <Button
      android:id="@+id/close"
        android:layout_width="100dip"
        android:layout_height="wrap_content"
         android:layout_alignParentRight="true"
         android:layout_below="@id/mytext"
        android:text="Fermer" />
         <Button
        android:id="@+id/ok"
        android:layout_width="100dip"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/close"
        android:layout_alignTop="@id/close"
        android:layout_marginLeft="10dip"
        android:text="Ajouter" />
</RelativeLayout>
```

La classe AddNewTable1.java : web service

2. Ouvrir le fichier AddNewTable1.java
(dossier src/**formation.exemple.helloplumwebservice** de votre projet), et modifier le comme ceci :

```
package formation.exemple.helloplumservicedatabase;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import plum.webservice.database.PlumDataBase;
public class AddNewTable1 extends Activity implements View.OnClickListener {
      PlumDataBase webdata;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.newtable1);

        /** écouteur */
        Button buttonOk=(Button) findViewById(R.id.ok);
        buttonOk.setOnClickListener(this);
        Button buttonclose=(Button) findViewById(R.id.close);
        buttonclose.setOnClickListener(this);

        webdata=new PlumDataBase
      ("http://10.0.2.2/plumwebservice/plumWebServiceDataBase.php");
    }

    public void onClick(View v) {
      EditText myText=(EditText)findViewById(R.id.mytext);

      switch (v.getId())
      {case R.id.ok :
            StringBuilder sqlb=new StringBuilder("insert into table1 values('','");
            sqlb.append(myText.getText());sqlb.append("')");
            String sql=sqlb.toString();
            webdata.execute(sql);

            Toast.makeText(getApplicationContext(), "Le nom a été ajouté",
            Toast.LENGTH_SHORT).show();
            myText.setText("");
            break;
      case R.id.close :
            finish();
            break;
            }
      }
}
```

Quelques précisions :

StringBuilder : cette classe permet de mettre en oeuvre la concaténation de String de façon efficace.

```
//par exemple : "insert into table1 values('','Thierry')
StringBuilder sqlb=new StringBuilder("insert into table1 values('','");
sqlb.append('Thierry');sqlb.append("')");
```

finish() :
*est une méthode la classe **Activity** , elle provoque l'arrêt du contrôleur en cours. Ici permet de fermer le contrôleur **addNewTable1.java**. Le contrôleur **HelloPlumWebService** est réactivé car c'est lui qui a démarré **addNewTable1**.*

webdata.execute(sql) :
*execute est une méthode la classe **PlumDataBase** , elle permet de demander l'exécution d'une requête de mise à jour (update, delete) auprès du web service.*

StartActivity : démarrer l'activité AddNenewTable1, Intent

3. Ouvrir le fichier HelloPlumWebServiceDataBase.java (dossier src/**formation.exemple.helloplumwebservice** de votre projet).

- Modifier `public class HelloPlumWebServiceDataBase` comme ceci :

```
public class HelloPlumWebServiceDataBase extends Activity implements
View.OnClickListener{
```

- Ajouter la méthode un écouteur sur le bouton **Ajouter** :

```
Button monBouton=(Button) findViewById(R.id.ajouter);
monBouton.setOnClickListener(this);
```

- Ajouter les 2 méthodes suivantes :

```
 @Override
//onClick permet de démarrer l'activité AddNewTable1
public void onClick(View v) {
      Intent intent = new Intent(this,AddNewTable1.class);
      startActivity(intent);
}
//onRestart est un événement qui se déclenche au moment de la ré-activation de
l'activité
protected void onRestart(){
    super.onRestart();
    listTable1();
}
```

4. Modifier le fichier AndroidManifest.xml pour indiquer la présence d'une nouvelle Activity dans l'application , ajouter ceci [**flèche en gras**] :

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=
             "formation.exemple.helloplumservicedatabase.HelloPlumWebServiceDataBase"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

➡

```
      <activity
     android:name="formation.exemple.helloplumservicedatabase.AddNewTable1"
      android:label="@string/app_ajout">
      </activity>
</application>
```

5. Modifier le fichier **res/values.strings.xml** afin de préciser le libellé à afficher en entête de l'Activity :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, HelloPlumWebServiceDataBase!</string>
    <string name="app_name">HelloPlumWebServiceDataBase</string>
    <string name="app_ajout">HelloPlumWebServiceDataBase : Ajouter</string>
</resources>
```

6. Démarrer HelloPlumWebServiceDataBase.java et tester votre nouveau contrôleur.

Quelques précisions :

StartActivity() :
est une méthode la classe ***Activity*** *, elle permet de démarrer un contrôleur (Activity) depuis une Activity active.*

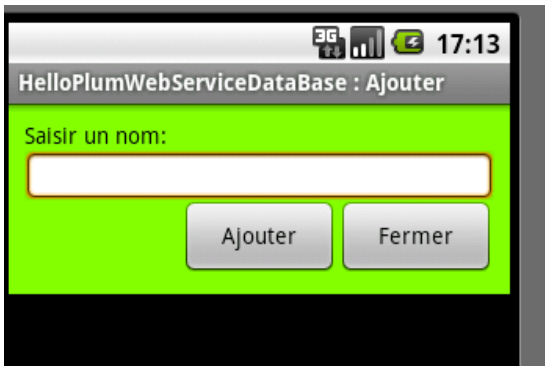
intent = new Intent(this ,AddNewTable1.class) :
Intent *est une classe qui contient la description abstraite d'une activité à réaliser. Elle permet également de mémoriser des données. Cette objet permet de facilité le démarrage d'une activité dans une application. Elle permet de faire le lien entre différents contrôleurs [Activity].*
paramètres
context : le contexte de l'application [this]
Class<?> cls : la classe Activity à utiliser [AddNewTable1]

protected void onRestart() :
est une méthode évènementielle qui se produit lorsque l'Activity est réactivée , ici lorsque ***AddNewTable1*** *se termine.*

AndroidManifest.xml :
est une fichier qui contient les informations essentielles sur une application Android. Ce fichier est examiné par le système dès le lancement d'une application, il est obligatoire et se situe toujours à la racine du projet.

Chaque nouvelle Activity doit être signalée dans ce fichier.

```
<activity
    android:name="formation.exemple.helloplumservicedatabase.AddNewTable1"
    android:label="@string/app_ajout">
</activity>
```



L'Activity ModOrSupTable1.java : Modifier/Supprimer un nom dans table1

Organisation du layout mod_or_del_table1.xml

Le layout **mod_or_del.xml** permet la modification ou la suppression d'un nom.

1. Ouvrir le fichier mod_or_sup_table1.xml (dossier **res/layout** de votre projet), et modifier le comme ceci :

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
     android:background="#80FF00"
     android:padding="10px" >
    <TextView
        android:id="@+id/mlabel"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textColor="#000000"
        android:text="Nom:"/>
    <EditText
        android:id="@+id/mmytext"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/mlabel"/>
    <Button
      android:id="@+id/mclose"
        android:layout_width="100dip"
        android:layout_height="wrap_content"
         android:layout_alignParentRight="true"
         android:layout_below="@id/mmytext"
        android:text="Fermer" />
        <Button
        android:id="@+id/msup"
        android:layout_width="100dip"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/mclose"
        android:layout_alignTop="@id/mclose"
        android:layout_marginLeft="10dip"
        android:text="Supprimer" />
         <Button
        android:id="@+id/mmod"
        android:layout_width="100dip"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/msup"
        android:layout_alignTop="@id/msup"
        android:layout_marginLeft="10dip"
        android:text="Modifier" />
</RelativeLayout>
```

StartActivity : démarrer l'activité ModOrSupTable1

2. Ouvrir le fichier HelloPlumWebServiceDataBase.java (dossier src/**formation.exemple.helloplumwebservice** de votre projet).

- Modifier `public class HelloPlumWebServiceDataBase` comme ceci :

```
public class HelloPlumWebServiceDataBase extends Activity implements
View.OnClickListener
,AdapterView.OnItemClickListener{
```

- Ajouter un écouteur sur la liste à la fin de la méthode **listeTable1()**

```
lvTable1.setOnItemClickListener(this);
```

- Ajouter la méthode suivante :

```
//agit si on choisit un nom dans la liste : démarre l'activité ModOrSupTable1
 public void onItemClick(AdapterView<?> ad, View v, int pos, long id) {
            Intent intent = new Intent(this,ModOrSupTable1.class);

            intent.putExtra("formation.exemple.helloplumservicedatabase.id", id);
            startActivity(intent);
      }
```

3. Modifier le fichier AndroidManifest.xml pour indiquer la présence d'une nouvelle Activity dans l'application , ajouter ceci :

```
        <activity android:name=".ModOrSupTable1"
        android:label="@string/app_modsup">
         </activity>
```

4. Modifier le fichier **res/values.strings.xml** afin de préciser le libellé à afficher en entête de l'Activity :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, HelloPlumWebServiceDataBase!</string>
    <string name="app_name">HelloPlumWebServiceDataBase</string>
    <string name="app_modsup">HelloPlumWebServiceDataBase : Ajouter</string>
</resources>
```

Quelques précisions :

StartActivity() :
*est une méthode la classe **Activity** , elle permet de démarrer un contrôleur (Activity) depuis une Activity active.*

intent = new Intent(this ,ModOrSupTable1.class) :
***Intent** est une classe qui contient la description abstraite d'une activité à réaliser. Elle permet également de mémoriser des données. Cette objet permet de facilité le démarrage d'une activité dans une application. Elle permet de faire le lien entre différents contrôleurs [Activity].*
paramètres
context : le contexte de l'application [this]
*Class<?> cls : la classe Activity à utiliser [ModOrSup*Table1*]*

intent.putExtra("formation.exemple.helloplumservicedatabase.id", id) :
*La méthode **putExtra** permet mémoriser des données qui pourront être utilisées par l'activité après le démarrage de celle-ci (**StartActivity**).*
paramètres
String name : le nom de la donnée, elle sera toujours préfixée avec le nom du paquetage,

ici "formation.exemple.helloplumservicedatabase", le nom de la variable est donc **id**.
value : la valeur fournie (long, int, String, etc.)

La classe ModOrSupTable1.java : web service, Intent, Cursor

5. Ouvrir le fichier ModOrSupTable1.java
(dossier src/**formation.exemple.helloplumwebservice** de votre projet), et modifier le comme ceci :

```
package formation.exemple.helloplumservicedatabase;

import plum.webservice.database.PlumDataBase;
import android.app.Activity;
import android.database.Cursor;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class ModOrSupTable1 extends Activity implements View.OnClickListener {
      private PlumDataBase webdata;
      private long _id;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.mod_or_sup_table1);

        Button buttonMod=(Button) findViewById(R.id.mmod);
        buttonMod.setOnClickListener(this);
        Button buttonSup=(Button) findViewById(R.id.msup);
        buttonSup.setOnClickListener(this);
        Button buttonclose=(Button) findViewById(R.id.mclose);
        buttonclose.setOnClickListener(this);

        _id=getIntent().getLongExtra("formation.exemple.helloplumservicedatabase.id",-
1);

       webdata=new PlumDataBase
      ("http://10.0.2.2/plumwebservice/plumWebServiceDataBase.php");

        StringBuilder sqlb=new StringBuilder("Select nom from table1 where _id=");
        sqlb.append(_id);
        Cursor c =webdata.query(sqlb.toString());

        c.moveToNext();
        String nom=c.getString(0);
        EditText myText=(EditText)findViewById(R.id.mmytext);
        myText.setText(nom);
    }
```

```
public void onClick(View v) {
            EditText myText=(EditText)findViewById(R.id.mmytext);
            StringBuilder sqlb;
            String sql;
            switch (v.getId())
            {      case R.id.mmod :
                   sqlb=new StringBuilder("update table1 Set nom='");
                   sqlb.append(myText.getText());
                   sqlb.append("' where _id=");sqlb.append(_id);
                   sql=sqlb.toString();
                   webdata.execute(sql);
                   Toast.makeText(getApplicationContext(), "Le nom a été modifié",
                   Toast.LENGTH_SHORT).show();
                   break;

                   case R.id.msup :
                   sqlb=new StringBuilder("delete from table1 where _id=");
                   sqlb.append(_id);
                   sql=sqlb.toString();
                   webdata.execute(sql);
                   Toast.makeText(getApplicationContext(), "Le nom a été supprimé",
                   Toast.LENGTH_SHORT).show();
                   finish();
                          break;
                   case R.id.mclose :
                          finish();
                          break;
            }
      }
}
```

Quelques précisions :

_id=getIntent().getLongExtra("formation.exemple.helloplumservicedatabase.id")
La méthode **getLongExtra** est une méthode la classe Intent qui permet de retrouver une donnée mémorisée grâce à la méthode **putExtra.**
paramètres
String name : le nom de la donnée, elle sera toujours préfixée avec le nom du paquetage, ici "formation.exemple.helloplumservicedatabase", le nom de la variable est donc **id.**
long valdefaut : valeur par défaut si la donnée recherchée n'est pas trouvée

c.moveToNext() :
*la méthode **moveToNext** permet de lire une ligne du curseur. Retourne **False** si on est à la fin du curseur.*

String nom=c.getString(0) :
*la méthode **getString** permet de retourner la valeur de la colonne 0 [ici le nom].*

6. Démarrer HelloPlumWebServiceDataBase.java et tester votre nouveau contrôleur.

Outils et astuces

Déboguage

Avec le plugin d'Android pour Eclipse il est possible d'utiliser le debuggueur d'Eclipse.
Pour cela il faut :

1. positionner un ou plusieurs point d'arrêt *(double-clic dans la barre à gauche de la ligne de code)*
2. Sélectionner le projet
3. Run + Debug

Ajouter un paquetage

Pour ajouter automatiquement un paquetage Appuyer sur : **Ctrl-Shift-O**

Android

La machine virtuelle : Dalvik

Pour comprendre certains problèmes il faut avoir en tête que :

- java nécessite une machine virtuelle pour s'exécuter
- cette machine sous Android s'appelle Dalvik VM
- Dalvik VM n'exécute pas le java code byte mais le Dalvik Execution Format(Dex)
- le format Dex prend en général moins de place

Contraintes Android

- mémoire lente
- la machine utilise un système de registre (les autre VM sont basés sur un système de pile [stack])
- Le processeur peut exécuter plusieurs instances de machine virtuelle afin d'isoler chaque application

Activity

Chaque activité qui démarre est mis sur le haut d'une pile. La fin de l'activité provoque la ré-activation de l'activité mise en sommeil au moment du démarrage de celle-ci.

```
Exemple :
A1 démarre A2, quand A2 est terminée A1 se réactive.
```

Cycle de vie d'un contrôleur [Activity]

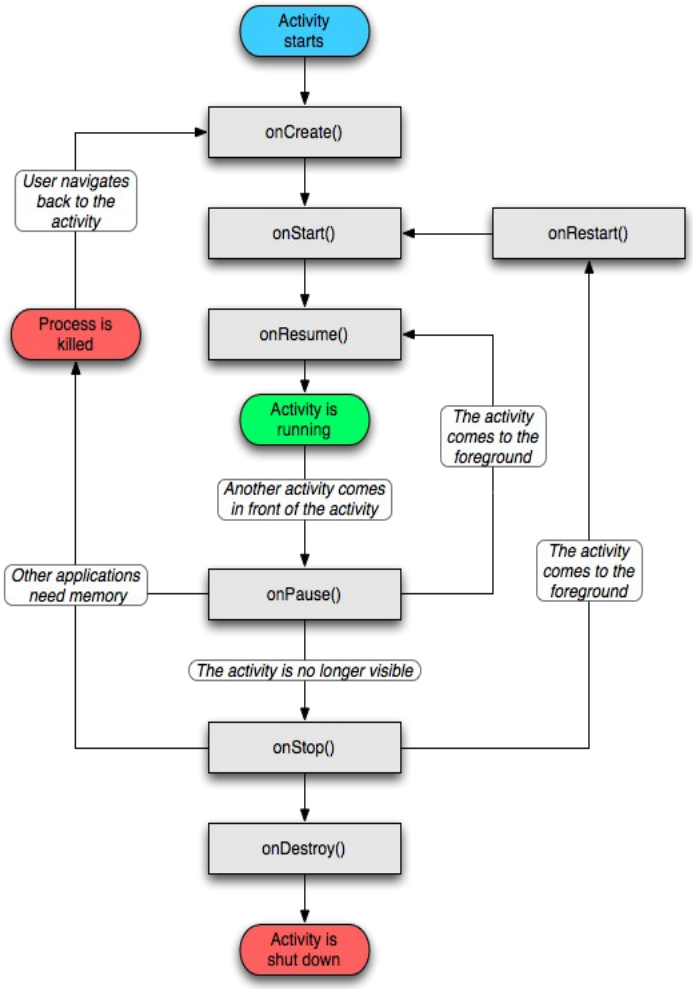


Table des matières