Séance 5b : Débuter en programmation Android

Nous considérerons qu'avant d'entamer cette séance pratique, vous avez déjà complété les étapes de la séance pratique 5a. Si tel n'est pas le cas, veuillez d'abord compléter les instructions décrites dans cette séance précédente.

Pour démarrer

Pour commencer nous allons regarder de plus près une partie de notre première application "Hello World", en particulier le code de la seule activité qu'elle comprend pour l'instant. *com.myfirstproject.MyActivity*.

```
1 package com.LSINF1225.myapplication;
2 import ...
3 public class MyActivity extends Activity {
4     @Override
5     protected void onCreate(Bundle savedInstanceState) {
6         super.onCreate(savedInstanceState);
7         setContentView(R.layout.activity_my);
8     }
9 ...
```

Listing 1 – L'activité "Hello World" par défaut

Référer à des ressources avec la classe descriptrice de ressources R

La ligne "`setContentView(R.layout.activity_my)`" du code de l'activité définit le layout (écran) de cette activité comme étant celui décrit dans le fichier */res/layout/activity_my.xml*. Cette ressource est "indexée" via la classe *R*. Bien que le fonctionnement de la classe R puisse vous sembler quelque peu obscur lorsque vous commencez à programmer pour Android elle est très utile pour parvenir à accéder aux ressources (tout ce qu'utilise votre application qui n'est pas du code source) dont Android a besoin. Une chose positive au sujet de l'emploi de cette classe est qu'elle est générée de façon transparente et gérée par le framework. Android y ajoutera automatiquement les ressources que vous créerez. Ainsi, une expression telle que *R.layout.activity_my* est interprétée comme une ressource de type layout nommée *activity_my*. Vu que les ressources de type layout existent sous forme de fichiers XML, le framework recherchera le fichier *activity_my.xml* situé dans le répertoire */res/layout*. Ceci est une bonne illustration du principe de "convention plutôt que configuration" en matière de design logiciel. Le layout de l'activité du "Hello World" est montré dans le listing 2. Notez qu'une vue statique du layout est déjà présente dans un volet preview à coté de votre éditeur XML. Au bas de l'éditeur vous pouvez accéder via l'onglet Design à un fenêtre d'édition assistée.

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:paddingLeft="@dimen/activity_horizontal_margin"
6      android:paddingRight="@dimen/activity_horizontal_margin"
7      android:paddingTop="@dimen/activity_vertical_margin"
8      android:paddingBottom="@dimen/activity_vertical_margin"
9      tools:context=".MyActivity"
10     android:clickable="true">
11     <TextView
12         android:text="@string/hello_world"
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content" />
15 </RelativeLayout>
```

Listing 2 – layout par défaut de la "Hello World" activity

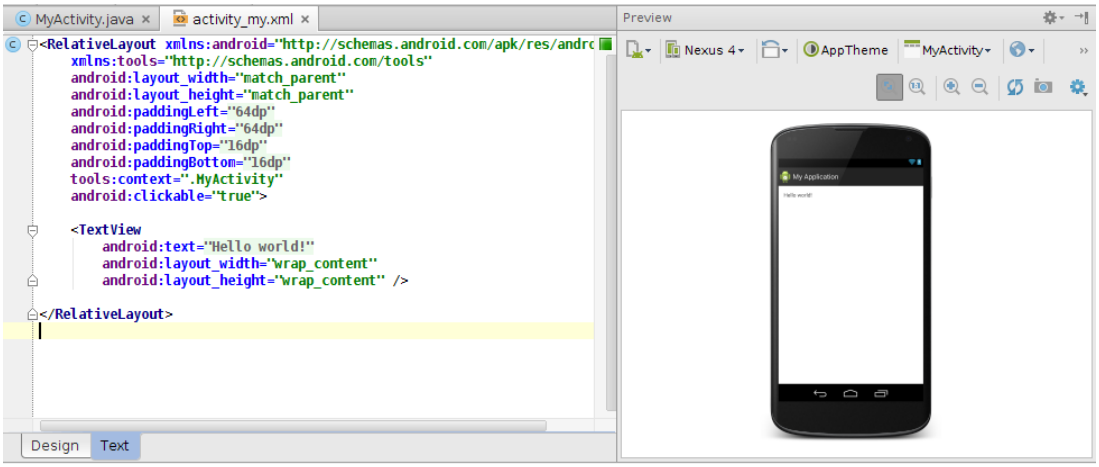

FIGURE 1 – L'éditeur de layouts d'Android studio

Notre layout comprend un layout relatif et une vue de type texte imbriquée. [1] (lignes 12 à 15) Le contenu de la textView est la ressource de type string *hello_world* (ligne 13). En plus des layouts, plusieurs autres types de ressources peuvent être définies. Par exemple, les ressources de type string sont utiles pour conserver à un même endroit des valeurs string importantes telles celles qui sont visibles sur l'interface utilisateur. Les strings sont définis dans le fichier */res/values/strings.xml*. Ce fichier est visible dans le listing 3. A la ligne 4 on peut voir la définition du string *hello_world*.

1. http://developer.android.com/guide/topics/ui/layout/relative.html

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My Application</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
</resources>
```

Listing 3 – Ressources par défaut de type string (values/strings.xml)

Programmation Android

Durant la séance précédente, vous avez créé une application Android plutôt simple. Une activité par défaut a été créée par le framework de sorte qu'il ne fallait rien faire d'autre pour pouvoir voir une petite application en action. Cette section explique comment modifier l'application par défaut pour en obtenir une qui soit un peu plus intéressante. Gardez cependant à l'esprit que (du à des contraintes de temps) ceci n'est pas un tutoriel Android détaillé. Son but est de vous accompagner dans vos premiers pas à l'intérieur de l'environnement de développement Android.

Durant cette séance, nous reviendrons sur les notions introduites au cours de la séance magistrale et introduirons de façon pratique certaines notions que vous aurez l'occasion d'approfondir au cours d'une séance suivante. Dans un premier temps, nous allons modifier pas à pas l'écran par défaut.

Ajouter des ressources

Nous allons commencer par ajouter quelques ressources de type string que nous utiliserons dans notre application. Modifiez le fichier ressources *strings.xml* en y ajoutant à la fin les entrées montrées dans le listing 4 (lignes 6 à 9).

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My Application</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="button_choose_item">Choose new item</string>
    <string name="label_current_selection">Current selection: </string>
    <string name="label_default_selection">None</string>
    <string name="title_activity_my_list">My List Activity</string>
</resources>
```

Listing 4 – Fichier ressources de type string modifié (values/strings.xml)

Créer une nouvelle activité

```
1  package com.LSINF1225.myapplication;
2  import android.app.ListActivity;
3  import java.util.List;
4  import java.util.Arrays;
5  import android.os.Bundle;
6  import android.widget.ArrayAdapter;
7  import android.widget.AdapterView;
8  import android.widget.Toast;
9  import android.view.View;
10 import android.content.Intent;
11 public class MyListActivity extends ListActivity {
12     public static final String CHOSEN_TEXT = "texteChoisi";
13     List<String> someStrings = Arrays.asList("java", "scala", "prolog", "smalltalk");
14     String chosenString;
15     @Override
16     public void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, //contexte courant
19                 android.R.layout.simple_list_item_1, //id de l'element de layout utilise
20                                                      //par defaut pour representer les
21                                                      //lignes de la liste (cet id est pre
                                                        // -defini dans Android)
22                 someStrings); // la collection d'elements à adapter à la liste
23         setListAdapter(adapter);
24         getListView().setOnItemClickListener(new AdapterView.OnItemClickListener() {
25             @Override
26             public void onItemClick(AdapterView<?> parent, View view, int position, long
               id) {
27                 chosenString = someStrings.get(position);
28                 Toast.makeText(MyListActivity.this, "Vous avez choisi : " + chosenString,
                    Toast.LENGTH_SHORT).show();
29             }
30         });
31     }
32     @Override
33     public void finish() {
34         if(chosenString != null) {
35             Intent data = new Intent();
36             data.putExtra(CHOSEN_TEXT, chosenString);
37             setResult(RESULT_OK, data);
38         }
39         super.finish();
40     }
41 }
```

Listing 5 – Une ListActivity

Maintenant, créez une nouvelle classe dans le même package et nommez la *MyListActivity* modifiez-la comme indiqué dans le listing 5. En étendant la classe *ListActivity*, nous signifions à Android que la nouvelle activité va juste afficher une liste de valeurs à l'écran.

Ajoutez une variable représentant un objet List, nommez la variable *someStrings* et initialisez-la avec une liste de Strings (ex : liste de vos langages préférés) (ligne 5). Dans la méthode *onCreate* de l'activité, instanciez un objet de type *ArrayAdapter* (ligne 10). Un ArrayAdapter sait comment insérer une collection d'objets dans une listView. Il reçoit comme paramètres l'activité qui affiche la liste, l'id de la vue qui affiche chacun des élément de la liste (par défaut cette vue est nommée *simple_list_item1*, cfr. ligne 11) et la collection d'objet à afficher par la ListView (dans notre cas nommée *someStrings*). A la ligne 15, nous configurons notre ListActivity pour qu'elle utilise l'objet ArrayAdapter que nous avons précédemment créé. Nous ajoutons également un *OnItemClickListener* à la ListView de l'activité (lignes 16 à 22). Le Listener permettra à la liste d'être à l'écoute des évènements de type "itemClick" vennant de l'utilisateur. La réaction à l'évènement est définie dans la méthode *onItemClick* qui reçoit la position de l'élément qui a été séléctioné. Nous mettons cet élément dans une variable d'instance et affichons un court message à l'utilisateur (lignes 19 à 20). Il est important de se rappeler de faire un appel à la methode *onCreate* de la super classe (ligne 9) pour éviter que le framework ne lance une exception de type *SuperNotCalledException*. Pour l'heure, ne faites pas attention à la méthode *finish*. Son rôle sera expliqué plus tard. Après avoir créé cette nouvelle activité, il nous faut la déclarer dans le fichier manifeste de l'application *AndroidManifest.xml* (voir listing 6, lignes 17 à 20).

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.LSINF1225.myapplication" >
4      <application
5          android:allowBackup="true"
6          android:icon="@drawable/ic_launcher"
7          android:label="@string/app_name"
8          android:theme="@style/AppTheme" >
9          <activity
10             android:name=".MyActivity"
11             android:label="@string/app_name" >
12             <intent-filter>
13                 <action android:name="android.intent.action.MAIN" />
14                 <category android:name="android.intent.category.LAUNCHER" />
15             </intent-filter>
16         </activity>
17         <activity
18             android:name=".MyListActivity"
19             android:label="@string/title_activity_my_list" >
20         </activity>
21     </application>
22 </manifest>
```

Listing 6 – Le fichier manifeste de l'application

Le fichier manifeste Android contient des informations importantes à propos de l'application et ses activités qui y sont toutes listées. La balise *intent-filter* peut être utilisée pour spécifier entre autres quelle est l'activité qui devrait apparaître en premier à l'écran au lancement de l'application pour la première fois. Dans notre exemple, cette balise est imbiquée dans la déclaration de l'activité *MainActivity*. Faites l'expérience qui suit : déplacez la balise *intent-filter* et insérez la dans la déclaration de *MyListActivity* (pour que la *MyActivity* soit la première à se lancer) puis exécutez l'application. Vous devriez voir s'afficher une liste de langages de programmation sur l'écran de votre emulateur ou de votre appareil Android (figure 2). Essayer de sélectionner un élément au hasard de la liste pour voir ce qui se produit. Ensuite n'oubliez pas de remettre la balise *intent-filter* à sa place dans la déclaration de la *MyActivity*.

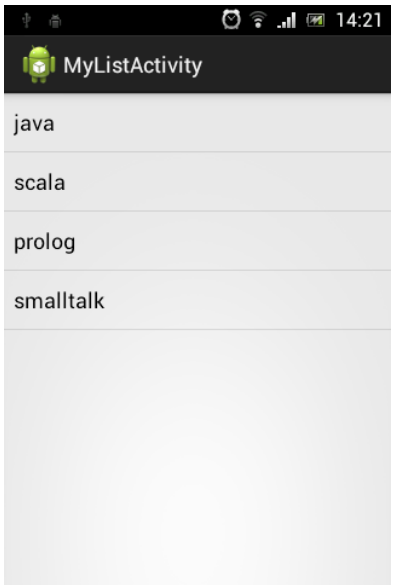

FIGURE 2 – Une activité liste

Vous vous demandez probablement comment Android procède pour retrouver les classes qui implémentent les activités déclarées dans le fichier manifeste. Vous remarquerez que les activités dans le fichier manifeste ne sont référencées que par leurs noms, leur nom de package n'est pas mentionné dans leurs déclarations. C'est parce qu'à la création de l'application, nous avions spécifié le package de l'application et que cette information a été enregistrée au début du fichier manifeste (voir listing 6 ligne 3). Cela suffit pour que la plate-forme arrive à retrouver les classes déclarées en tant qu'activités bien qu'elles ne soient référencées que par leurs noms (le nom du package sera simplement rajouté au début).

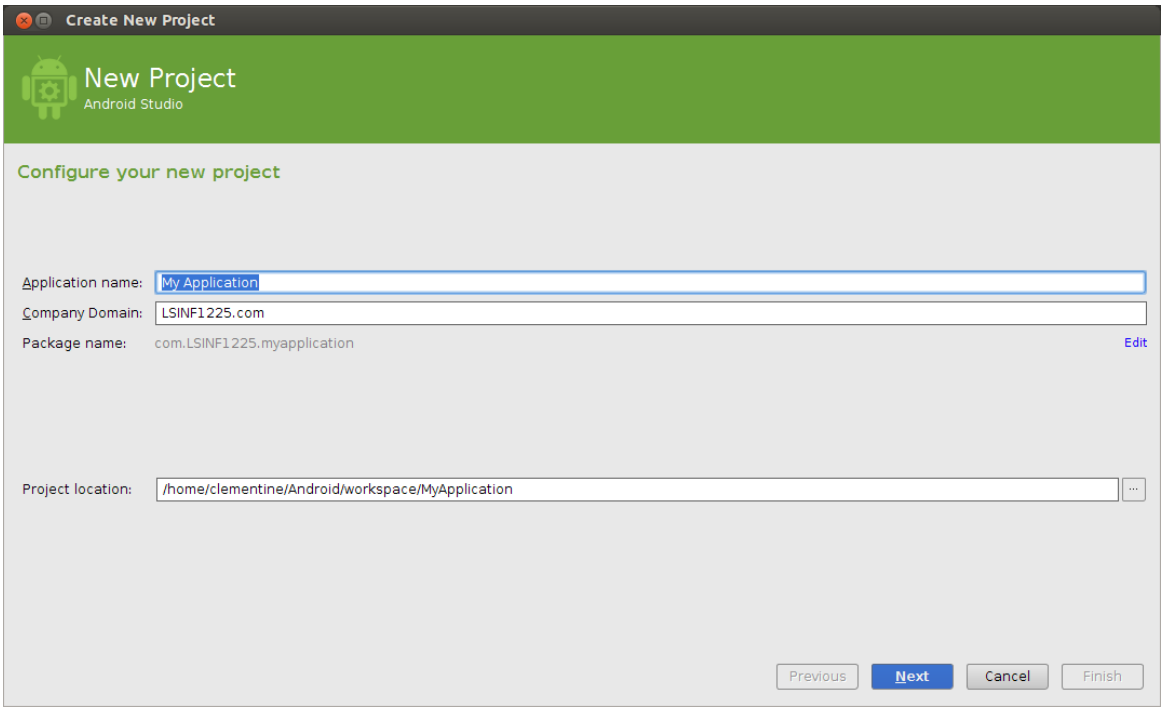

FIGURE 3 – Assistant de création de nouveau projet

Aller d'une activité à l'autre

Cette section est une introduction pratique aux concepts qui seront approfondis durant la prochaine séance magistrale. Modifiez le layout de votre activité principale `/res/layout/activity\_my.xml` afin d'y inclure deux labels et un bouton comme illustré dans le listing de code 7. Remarquez que les labels des TextViews et du bouton ne sont pas codés en dur dans leurs vues respectives. Ces labels réfèrent plutôt au ressources de type string définies auparavant (lignes 21, 26 et 34).

Maintenant modifiez le code de la classe *MyActivity* comme illustré dans le listing 8. Créez deux variables d'instances de type TextView et Button respectivement (lignes 6 et 7). Dans la méthode *onCreate*, nous initialisons la première variable d'insance créée avec l'une des TextViews qui se trouve dans le layout de l'activité (ligne 13). Nous utilisons l'id de la vue dans le layout pour récupérer la vue grâce à la méthode *findViewById*. Cette vue de type textView que nous récupérons servira à afficher la sélection courrante de l'utilisateur. De la même manière, nous nous servons ensuite de l'id du bouton pour le retrouver à l'intérieur du layout et l'assigner à la variable d'instance *selectItemButton*. Nous ajoutons au bouton un *onClickListener* comme fait plus haut. Le comportement que nous souhaitons à l'arrivée d'un évènement de type "click" est qu'une vue contenant notre liste de strings s'ouvre et une fois un élément sélectionné dans la liste, que notre premier écran réapparaisse

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MyActivity"
    android:clickable="false">
    <LinearLayout
        android:id="@+id/linear_layout_selection"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true" >
        <TextView
            android:id="@+id/label_current_selection"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/label_current_selection" />
        <TextView
            android:id="@+id/current_selection"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/label_default_selection" />
    </LinearLayout>
    <Button
        android:id="@+id/button_choose_item"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/linear_layout_selection"
        android:layout_centerHorizontal="true"
        android:text="@string/button_choose_item" />
</RelativeLayout>
```

Listing 7 – Le nouveau layout de l'activité MyActivity avec des vues de type textView et un bouton

avec avec cette information affichée. Ceci s'implémente en utilisant des *intents*.[2]. Nous instancions un objet Intent en fournissant comme premier argument l'activité courante et comme second argument l'activité que nous voudrions démarrer à l'activation du bouton (ligne 18). Nous enclenchons le lancement de la nouvelle activité en appelant la méthode *startActivityForResult* qui prend comme premier argument l'objet Intent créé précédemment et comme second argument un code contenu dans une constante qui servira à récupérer s'il y en a les résultats issus de l'invocation de la seconde activité. La méthode *onActivityResult* (ligne 32) sera appelée par le framework Android lorsque celui ci aura reçu le résultat d'exécution venant de l'activité invoquée. Nous

2. `http://developer.android.com/guide/components/intents-filters.html`

vérifions alors d'abord que le résultat correspond au code envoyé au moment de l'invocation de l'activité et ensuite si l'invocation s'est déroulée avec succès (ligne 33). Après quoi nous cherchons à savoir s'il existe une information portant le code *MyListActivity.CHOSEN_TEXT* dans l'intent renvoyé. Si tel est le cas nous extrayons cette information (un string) et nous l'assignons à la textView *currentSelsction*.

```
1  package com.LSINF1225.myapplication;
2  import ...
3
4  public class MyActivity extends Activity {
5      private static final int REQUEST_CODE = 1;
6      private TextView currentSelection;
7      private Button selectItemButton;
8
9      @Override
10     protected void onCreate(Bundle savedInstanceState) {
11         super.onCreate(savedInstanceState);
12         setContentView(R.layout.activity_my);
13         currentSelection = (TextView) findViewById(R.id.current_selection);
14         selectItemButton = (Button) findViewById(R.id.button_choose_item);
15         selectItemButton.setOnClickListener(new View.OnClickListener() {
16             @Override
17             public void onClick(View arg0) {
18                 Intent i = new Intent(MyActivity.this, MyListActivity.class);
19                 startActivityForResult(i, REQUEST_CODE);
20             }
21         });
22     }
23
24     @Override
25     public boolean onCreateOptionsMenu(Menu menu) {
26         //"Gonfle" le menu (ajout des elements de la barre d'action)
27         getMenuInflater().inflate(R.menu.my, menu);
28         return true;
29     }
30
31     @Override
32     public void onActivityResult(int requestCode, int resultCode, Intent data) {
33         if (requestCode == REQUEST_CODE && resultCode == RESULT_OK) {
34             if(data.hasExtra(MyListActivity.CHOSEN_TEXT))
35                 currentSelection.setText(data.getExtras().getString(MyListActivity.
                    CHOSEN_TEXT));
36         }
37     }
38 }
```

Listing 8 – La nouvelle version de l'activité MyActivity

Pour comprendre comment ces données ont été transmises de l'activité MyListActivity, il nous faut revenir au

listing 5. Dans la méthode *finish* nous vérifions si un string a été sélectionné par l'utilisateur (ligne 26). Si tel est le cas, nous créons un intent sans lui fournir d'arguments (ligne 27), mettons dans cet intent la sélection de l'utilisateur (ligne 28) et définissons cet intent comme la réponse de l'activité *MyListActivity* (ligne 29). Ainsi la sélection de l'utilisateur sera récupérée par l'activité *MyActivity* comme expliqué plus haut. Pour tester tout ceci, exécutez l'application dans votre emulateur ou sur votre appareil Android et appuyer sur le bouton pour avoir accès à l'écran de sélection. Après avoir sélectionné un élément de votre liste, utilisez le bouton "Back" d'Android pour revenir au premier écran. L'élément sélectionné devrait apparaître à l'écran comme indiqué sur la figure 4.

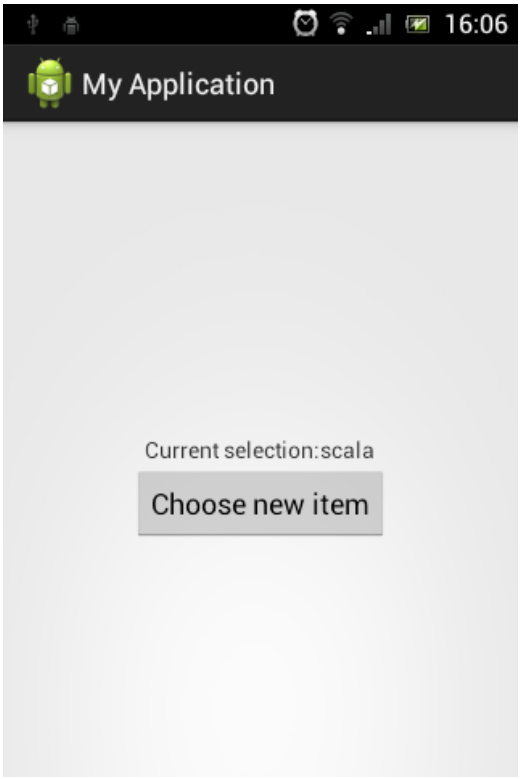

FIGURE 4 – Selecting an item

Introduction à l'utilisation de bases de données

Jusqu'à présent, nous avons affiché une liste de strings codée en dur dans l'activité *MyListActivity*. Dans cette dernière partie du TP, nous allons récupérer ces strings à l'intérieur d'une petite base de données SQLite que nous aurons créé au préalfable.

Dans le code de l'activité *MyListActivity*, mettez en commentaire l'initialisation de la liste de strings (ligne 1) puis insérer les instructions d'interaction avec la base de données comme suit :

```
1     List<String> someStrings; //= Arrays.asList("java", "scala", "prolog", "smalltalk");
2     String chosenString;
3
4     @Override
5     public void onCreate(Bundle savedInstanceState) {
6         super.onCreate(savedInstanceState);
7         MySQLiteOpenHelper db = new MySQLiteOpenHelper(this);
8         //Recuperer toutes les entrees de la base de donnees
9         if ( db.open() ) {
10            someStrings = db.getLanguages();
11        } else {
12            // erreur ouverture bd
13            throw new Error("Impossible d'ouvrir la base de donnees");
14        }
15 ...
```

Listing 9 – Modification de l'activité MyListActivity

La classe qui représente la base de données se nomme *MySQLiteOpenHelper* ; elle hérite de la classe *SQLiteOpenHelper* chargée de la gestion des bases de données SQLite.
Nous l'instancions (ligne 7) puis nous l'ouvrons avec la methode *open* que nous définirons à l'intérieur de cette classe. Nous définirons également une méthode *getLanguages* qui récupèrera la liste de strings à afficher (ligne 9 et 10). Une erreur est lancée au cas où un problème survient à l'ouverture de la base de données (ligne 13).

Dans le package *com.LSINF1225*, créez un nouveau dossier *database* contenant une nouvelle classe java *MySQLiteOpenHelper* qui étend la classe *SQLiteOPenHelper*.
Modifiez cette classe afin d'obtenir le résultat final suivant :

```java
package com.LSINF1225.database;

import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import java.util.ArrayList;
import java.util.List;

/**
 * @author Damien Mercier
 * @version 1
 * @see <a href="http://d.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html">SQLiteOpenHelper</a>
 */
public class MySQLiteOpenHelper extends SQLiteOpenHelper {

    /**
     * Nom du fichier de la base de donnees.
     */
    private static final String DATABASE_NAME = "database.sqlite";

    /**
     * Version de la base de donnees (à incrementer en cas de modification de celle-ci afin que la
     * methode onUpgrade soit appelee).
     *
     * @note Le numero de version doit changer de maniere monotone.
     */
    private static final int DATABASE_VERSION = 1;

    /**
     * Le contexte de l'application
     */
    private static Context context;

    /**
     * Constructeur. Instancie l'utilitaire de gestion de la base de donnees.
     *
     * @param context Contexte de l'application.
     */
    public MySQLiteOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
        this.context = context;
    }
```

Listing 10 – La classe qui représente la base de donnés

```
44      /**
45       * Methode d'initialisation appelee lors de la creation de la base de donnees.
46       * Ici on y cree les tables de la base de donnees et les remplit.
47       *
48       * @param db Base de donnees à initialiser.
49       * @param db Base de donnees à initialiser
50       */
51      @Override
52      public void onCreate(SQLiteDatabase db) {
53          db.execSQL("DROP TABLE IF EXISTS \"languages\";");
54          db.execSQL("CREATE TABLE \"languages\" (\"l_id\" INTEGER PRIMARY KEY
            AUTOINCREMENT NOT NULL , \"l_name\" TEXT);");
55      }
56
57      /**
58       * Ouvre la base de donnees en ecriture
59       */
60      public boolean open() {
61          try {
62              getWritableDatabase();
63              return true;
64          } catch(Throwable t) {
65              return false;
66          }
67      }
68
69      /**
70       * Supprime la table dans la base de donnees.
71       *
72       * @param db Base de donnees.
73       *
74       * @post La tables de la base de donnees passee en argument est effacee.
75       */
76      private void deleteDatabase(SQLiteDatabase db) {
77          db.execSQL("DROP TABLE IF EXISTS \"languages\";");
78      }
79
80      /**
81       * Methode de mise à jour lors du changement de version de la base de donnees.
82       *
83       * @param db         Base de donnees à mettre à jour.
84       * @param oldVersion Numero de l'ancienne version.
85       * @param newVersion Numero de la nouvelle version.
86       * @pre La base de donnees est dans la version oldVersion.
87       * @post La base de donnees a ete mise à jour vers la version newVersion.
88       */
89      @Override
90      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
91
92          deleteDatabase(db);
93          onCreate(db);
94      }
```

Listing 11 – La classe qui représente la base de donnés (suite)

```
95
96      /**
97       * Renvoie la liste les langages contenus dans la base de donnees
98       * @return null si requete sans resultat, une liste de strings contenant
99       * les langages presents dans la base de donnees.
100      */
101     public List<String> getLanguages() {
102         List<String> languages = new ArrayList<String>();
103         SQLiteDatabase db = getReadableDatabase();
104         // Les resultats de la requête sont mis dans un "curseur"
105         Cursor c = db.query("\"languages\"", // La table
106                 new String[]{"\"l_id\"", "\"l_name\""},
107                 null, // Colonnes pour la clause WHERE
108                 null, // Valeurs pour la clause WHERE
109                 null, // ne pas grouper les lignes
110                 null, // ne pas filtrer par goupe de ligne
111                 null  // pas d'ordre
112         );
113         if (c.moveToFirst()) {
114             for (int i = 0; i < c.getCount(); i++) {
115                 String s = c.getString(c.getColumnIndexOrThrow("l_name"));
116                 languages.add(s);
117                 c.moveToNext();
118             }
119         }
120         c.close();
121         return languages;
122     }
123 }
```

Listing 12 – La classe qui représente la base de donnés (suite)

Le code de cette classe permet de gérer la connexion entre l'application et la base de données.
Le champ constant *DATABASE_NAME* (ligne 20) contient le nom de la futur base de donnée SQLite qui sera stockée à un emplacement bien déterminé dans le système de fichier d'Android à l'exécution.
Le constructeur de la classe *MySQLiteOpenHelper* fait appel au constructeur de la super classe (ligne 41) en lui donnant en arguments le contexte, c'est à dire dans notre cas l'instance de l'activité *MyListActivity* qui communique avec la base de données, le nom de la base de données, null et la version de la base de données. A la place du paramètre *null* nous aurions pu si besoin passer une instance de *Cursorfactory* afin de permettre à l'appel de la méthode *query* plus bas de retourner des sous-classes de la classe *Cursor*.
A l'intérieur de la méthode redéfinie *onCreate* on peut exécuter directement les instructions SQL qui créeront et rempliront les tables de notre base de données (lignes 53 et 54). Si une table nommée *"languages"* existe, nous la supprimons et créons une nouvelle table en lui donnant pour noms de colonnes *"l_id"* et *"l_name"*, avec la colonne *"l_id"* comme clé primaire.
Ajoutez à l'intérieur de cette méthode les instructions qui vont nous permettre de populer la table *"languages"*

avec les strings que nous voulons voir apparaître dans notre liste à l'écran.

```
@Override
public void onCreate(SQLiteDatabase db) {

    db.execSQL("DROP TABLE IF EXISTS \"languages\";");
    db.execSQL("CREATE TABLE \"languages\" (\"l_id\" INTEGER PRIMARY KEY
    AUTOINCREMENT NOT NULL , \"l_name\" TEXT);");

    db.execSQL("INSERT INTO \"languages\" VALUES(1,'java');");
    db.execSQL("INSERT INTO \"languages\" VALUES(2,'scala');");
    db.execSQL("INSERT INTO \"languages\" VALUES(3,'prolog');");
    db.execSQL("INSERT INTO \"languages\" VALUES(4,'smaltalk');");
}
```

Listing 13 – Instructions SQL à ajouter

A l'intérieur d'une méthode *open* (ligne 60), nous testons l'ouverture de la base de données en demandant à la classe *SQLiteOpenHelper* une base de données ouverte en écriture c'est à dire dont on peut modifier le contenu. Nous n'avons pas besoin de conserver la référence de la base de donnée renvoyée. Si une exception est levée lors de l'ouverture de la base de donnée, nous renvoyons false à la méthode appelante.
La méthode *deleteDatabase* reprend simplement l'instruction de suppression de la table *"languages"*.
La méthode *onUpgrade* fait appel aux méthodes *deleteDatabase* et *onCreate* lorsque la version de la base de données est changée.
Enfin, la méthode *getLanguage* (ligne 101) va nous permettre d'ouvrir la base de données et de faire une requête sur celle-ci à l'aide de la méthode *query* (ligne 105). Vu que nous ne ferons qu'afficher les noms de langages, la base de données peut être ouverte en lecture. La méthode *query* vas composer notre requête SQL en fonction des arguments que nous lui donnons : le nom de la table, un tableau de Strings contenant les noms des colonnes visées, la colonne pour la clause WHERE, un tableau de strings contenant les valeurs pour la clause WHERE, une valeur pour la clause GROUPE BY , une valeur pour la clause HAVING et une valeur pour la clause ORDER BY.
Dans notre cas nous n'avons besoin de fournir que les deux premiers arguments.
La méthode *query* renvoie un curseur (Cursor). Un curseur est un Objet qui encapsule l'ensemble des lignes qui résultent de la requête effectuée. Aux lignes 113 à 119 à nous parcourons le curseur *c* pour en extraire l'information qui nous intéresse et la mettre dans la liste de strings *languages* à l'aide des méthodes *moveToFirst*, *getStringIndexOrThrow*, *getString* et *moveToNext*.
Il est important, dès qu'on a récupéré les résultats de la requête, de fermer le curseur par l'appel de la méthode *close* (ligne 120) afin que toutes les ressources qu'il mobilise puissent être libérées.