Création d’interfaces simples

Partie 2

Plan

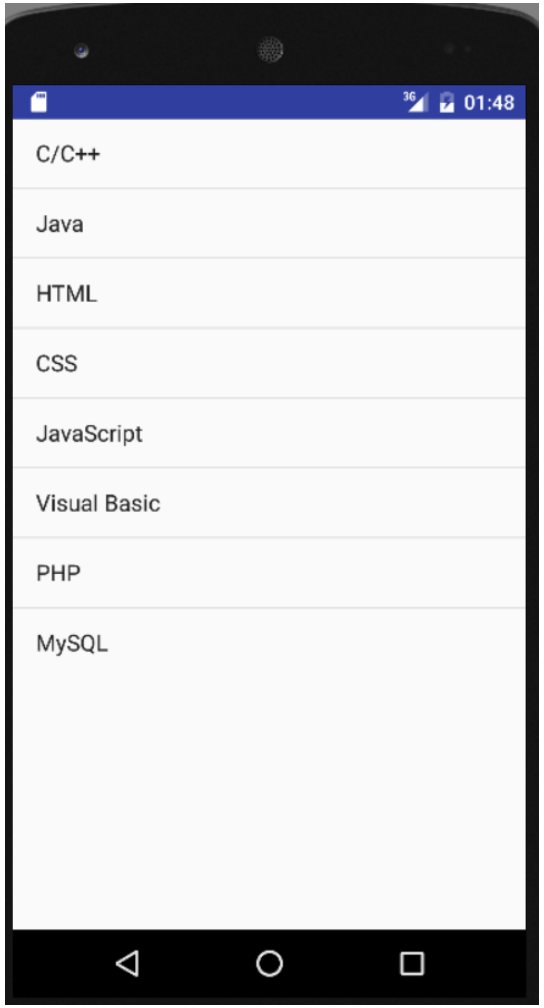
- Les listes (ListView)
  - ListActivity
  - Activity
- Personnalisation d’une liste
- ActionBar

Une liste

- Sous Android, une liste représente un ensemble d'éléments s'affichant les uns à la suite des autres
- Chaque élément d'une liste peut posséder de une à trois lignes maximum
  - Chaque ligne d'une liste peut être personnalisé avec des composants comme TextView, Button, ImageView, …
- Une liste est crée soit à partir de
  - ListActivity : Si la vue contient uniquement une liste
  - Activity : Si la vue contient une liste avec d'autres composants de UI
- Pour insérer des données dans une liste, Android propose plusieurs types d'adapteurs dont :
  - ArrayAdapter : Permet de remplir une liste à partir d'un tableau ou d'une collection
  - SimpleCursorAdapter : Permet de remplir une liste à partir d'une base de données
  - BaseAdapter : Permet de créer son propre adapteur

Exercice 1

- Créer une application Android qui utilise la classe ListActivity pour afficher la liste des cours en Informatique : C/C++, Java, HTML, CSS, JavaScript, Visual Basic, PHP, MySQL



Solution

- Créer un nouveau projet Android (Empty Activity)
- Modifier le fichier res/values/strings.xml pour ajouter la liste d'éléments (voir diapo 6)
- Créer le fichier xml (layout : LinearLayout) nommé main
  - Clic droit sur le dossier res/layout
  - New/Layout resource File
  - File name : main
  - Root Element : LinearLayout
- Écrire le code de la diapo 7 pour éditer le fichier main.xml
- Modifier le code de MainActivity.java en éditant le code de la diapo 8

Solution – Fichier : res/values/strings.xml

```
<resources>
    <string name="app_name">ListeView_Version1</string>

    <string-array name="id_liste_cours">
        <item>C/C++</item>
        <item>Java</item>
        <item>HTML</item>
        <item>CSS</item>
        <item>JavaScript</item>
        <item>Visual Basic</item>
        <item>PHP</item>
        <item>MySQL</item>
    </string-array>

</resources>
```

On crée un tableau de chaine qui compose les éléments qu'on va utiliser pour remplir la liste

Solution – Fichier : main.xml

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

Pour pouvoir manipuler une ListView grâce à une ListActivity, l'identifiant du composant ListView doit être obligatoirement @android:id/list qui est un identifiant déjà défini par Android.

Solution – Fichier : MainActivity.java

On utilise l'adapter ArrayAdapter pour pouvoir insérer des données dans le composant ListView.
Le constructeur ArrayAdapter prend 3 paramètres :

- Le premier paramètre correspond au contexte d'affichage de la liste qui est l'activité dans laquelle la liste est affichée (d'où l'utilisation de this)
- Le 2e paramètre indique le layout qui va être utilisé pour chaque ligne de la liste. On prend un layout prédéfini d'Android. Vous pouvez toujours choisir un layout personnalisé
- Le 3e paramètre est le tableau qui contient les données qui vont afficher dans la liste.

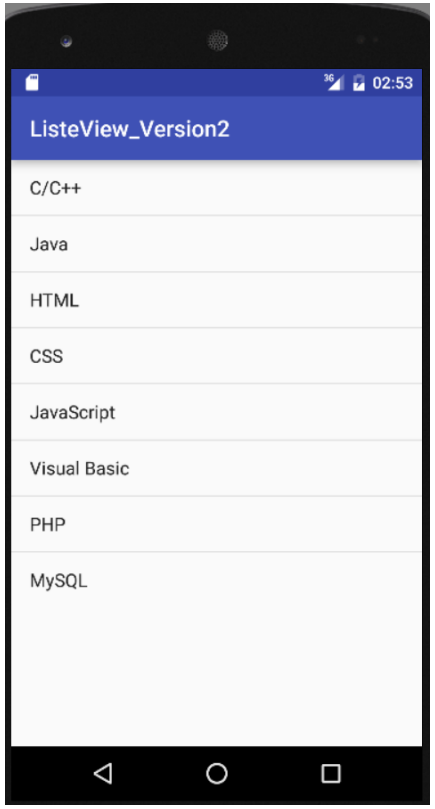
```
public class MainActivity extends ListActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Resources res = getResources();
        String[] listeCours = res.getStringArray(R.array.id_liste_cours);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, listeCours);
        setListAdapter(adapter);
    }
}
```

On récupère le tableau de String de nom id_liste_cours défini dans le fichier res/values/strings.xml

La méthode setListAdapter lie l'adapter contenant les données à la liste.

Exercice 2

- Créer une application Android qui utilise la classe Activity pour afficher la liste des cours en Informatique : C/C++, Java, HTML, CSS, JavaScript, Visual Basic, PHP, MySQL



Solution

- Créer un nouveau projet Android (Empty Activity)
- Modifier le fichier res/values/strings.xml pour ajouter la liste d'éléments (voir diapo 11)
- Créer le fichier xml (layout : LinearLayout) nommé main
  - Clic droit sur le dossier res/layout
  - New/Layout resource File
  - File name : main
  - Root Element : LinearLayout
- Écrire le code de la diapo 12 pour éditer le fichier main.xml
- Modifier le code de MainActivity.java en éditant le code de la diapo 13

Solution – Fichier : res/values/strings.xml

```
<resources>
    <string name="app_name">ListeView_Version1</string>

    <string-array name="id_liste_cours">
        <item>C/C++</item>
        <item>Java</item>
        <item>HTML</item>
        <item>CSS</item>
        <item>JavaScript</item>
        <item>Visual Basic</item>
        <item>PHP</item>
        <item>MySQL</item>
    </string-array>

</resources>
```

On crée un tableau de chaine qui compose les éléments qu'on va utiliser pour remplir la liste

Solution – Fichier : res/layout/main.xml

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

    <ListView
        android:id="@+id/MaListe"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

</LinearLayout>
```

Avec une classe qui hérite de Activity, il faut créer son propre id pour le composant ListView.

Solution – Fichier : MainActivity.java

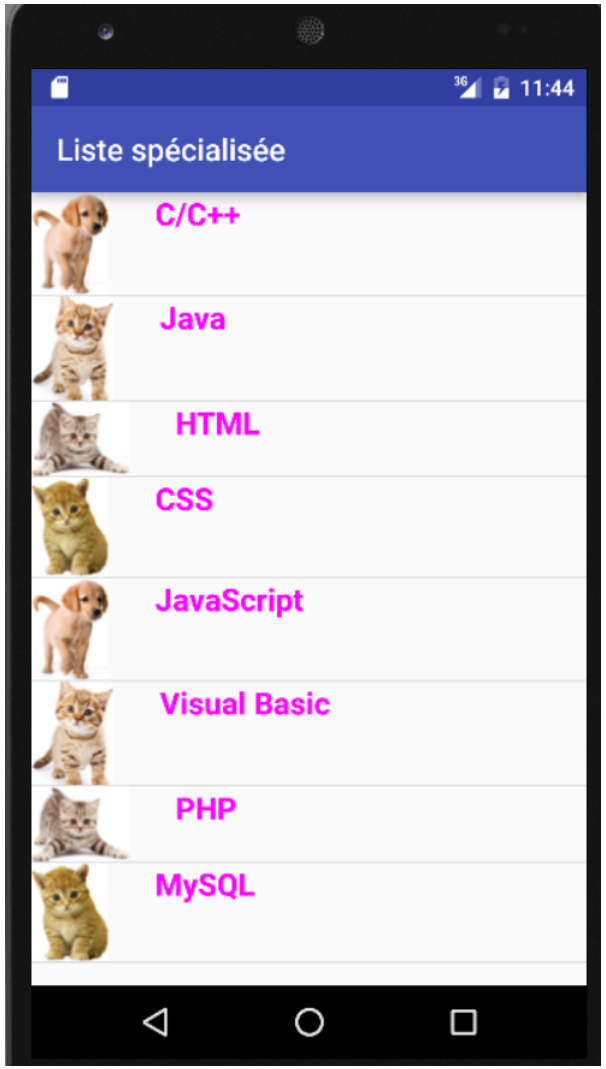
```
public class MainActivity extends Activity {

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

        Resources res = getResources();
        String[] listeCours = res.getStringArray(R.array.id_liste_cours);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, listeCours);
        ListView listeView = (ListView) findViewById(R.id.MaListe);
        listeView.setAdapter(adapter);
    }
}
```

Liste personnalisée

- On peut créer des adapters personnalisés afin de mieux gérer l’affichage et les données d’une liste.
- Voir l’exemple de l’exercice 3.

Exercice 3

- Créer une application Android qui affiche la liste des cours avec une image associée:



Solution

- Créer un nouveau projet Android (Empty Activity)
- Créer la classe Cours
  - Clic droit sur le dossier java
  - New Java Class (Nom de la classe : Cours.java)
  - Écrire le code de la diapo 17 dans le fichier Cours.java
- Créer les fichiers xml (layout_liste et item_liste : LinearLayout) nommé main
  - Clic droit sur le dossier res/layout
  - New/Layout resource File
  - File name : layout_liste et item_liste
  - Root Element : LinearLayout
- Écrire le code des diapos 18 à 19 pour éditer les fichiers XML respectifs
- Créer la classe Adapter personnalisé qui va servir à représenter les données dans la Liste
  - Clic droit sur le dossier java
  - New Java Class (Nom de la classe : AdapterListeCours qui hérite de ArrayAdapter)
  - Écrire l'explication de la diapo 20 et les codes des diapos 21 à 23 dans le fichier AdapterListeCours.java
- Modifier le code de MainActivity.java en éditant le code de la diapo 24

Solution : Cours.java

```java
/**
 * Created by kfostine on 2017-09-15.
 */
public class Cours {
    private String nom;
    private int image;

    public Cours(String nom, int image) {
        this.nom = nom;
        this.image = image;
    }

    public String getNom() {
        return nom;
    }

    public int getImage() {
        return image;
    }
}
```

La classe **Cours** va servir pour représenter chaque item de la liste. Dans chaque item de la liste, on a le nom du cours et l'image du cours qui y est associée.

L'image est un entier, parce qu'on ne vas pas représenter l'image par le nom de fichier(string) mais par id de sa ressource qui est R.drawable.nomFichierImage dans le code de java.

Solution : layout_liste.xml

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

Ce layout est le layout principal de l'activité qui affiche une ListeView

Solution : item_liste.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView
        android:id="@+id/icon"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="30dp"
        android:src="@drawable/animal1"/>

    <TextView
        android:id="@+id/titre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="Arial"
        android:textColor="#FF00FF"
        android:textSize="20sp"
        android:textStyle="bold"/>
</LinearLayout>
```

Ce layout est le layout utilisé pour affiché chaque item de la ListView. Dans chaque item de la ListView on a :

- Un composant ImageView qui affiche l'image correspondant au cours en question
- Un composant TextView qui affiche le nom du cours

Ces deux composants sont alignés hozizontamement dans un LinearLayout

La classe AdapterListeCours

- La classe AdapterListeCours représente l'adapter personnalisé qui va servir à fournir des données qui vont être affichées dans le composant ListView
- Elle hérite de ArrayAdapter parce que le remplissage des données s'effectue à l'aide d'un tableau
- Chaque élément de la liste représente un objet cours
- AdapterListeCours possède les trois méthodes suivantes :
  - Un constructeur
  - Une méthode getView : chaque appel à cette méthode permet de récupérer une ligne (donnée affichée dans la liste et la vue dans laquelle les données sont affichée) se trouvant à une position donnée
  - Une méthode getCount : Retourne le nombre d'éléments de la liste

Solution : AdapterListeCours.java

```java
import android.content.Context;
import android.content.res.Resources;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;
import java.util.ArrayList;

/**
 * Created by kfostine on 2017-09-15.
 */
public class AdapterListeCours extends ArrayAdapter<Cours> {
    private Context context;
    private int layoutItemListe;
    private Resources res;
    ArrayList<Cours> listeCours;

    public AdapterListeCours(Context context, int layoutItemListe, ArrayList<Cours>listeCours){
        super(context, layoutItemListe, listeCours);
        this.context = context;
        this.layoutItemListe = layoutItemListe;
        res = context.getResources();
        this.listeCours = listeCours;
    }
```

Explication des paramètres du constructeur AdapterListeCours

- **Context** : Représente le contexte de l'application. Dans notre cas, ça va être l'activité dans laquelle la ListView va être affichée.
- **layoutItemListe** : id du layout qui représente chaque item de la liste. Dans notre cas, ça va être le layout item_liste qui se trouve dans le fichier item_liste.xml
- **listeCours** : est la collection qui comporte la liste des objets cours qui vont être affichées dans le composant ListView

Solution : AdapterListeCours.java (Suite)

```
    @Override
    public View getView(int position, View convertView, ViewGroup parent){
        View view = convertView;
        if(view == null){
            LayoutInflater layoutInflater = (LayoutInflater) context.getSystemService(
                Context.LAYOUT_INFLATER_SERVICE);
            view = layoutInflater.inflate(layoutItemListe, parent, false);
            Cours cours = listeCours.get(position);
            if(cours != null){
                TextView textViewTitreCours = (TextView) view.findViewById(R.id.titre);
                ImageView imageViewCours = (ImageView) view.findViewById(R.id.icon);
                textViewTitreCours.setText(cours.getNom());
                imageViewCours.setImageResource(cours.getImage());
            }
        }
        return view;
    }

    @Override
    public int getCount() {
        return listeCours.size();
    }
}
```

Voir l'explication de la méthode getView à la diapo suivante.

La méthode getCount permet d'indiquer le nombre d'éléments contenus dans le composant ListView

La méthode getView de la classe AdapterListeCours

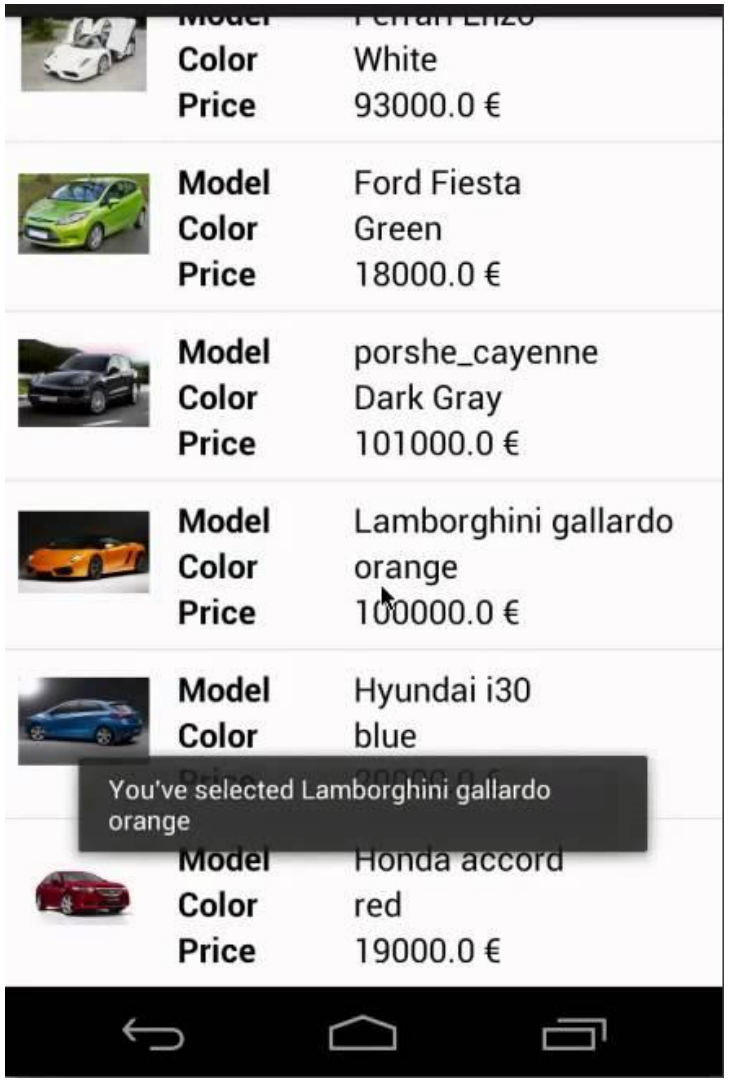
- La Méthode **getView(int position, View convertView, ViewGroup parent)**
  - Permet de récupérer la vue à appliquer à une ligne **donnée(argument position**)
  - La vue représentant une ligne de la liste est passée en argument à la méthode (**argument convertView**)
  - La vue parente (dans notre cas ListView) est passée en paramètre
- La vue personnalisée doit être chargée à l'aide de la méthode inflate. Une fois chargée elle sera passée en paramètre à la méthode getView (lors des prochains appels)
- Il faut récupérer les données à afficher dans la vue

Solution : MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout_liste);
        ArrayList<Cours> listeCours = new ArrayList<>();
        initialiserListe(listeCours);
        AdapterListeCours adapter = new AdapterListeCours(this, R.layout.item_liste, listeCours);
        ListView listView = (ListView) findViewById(R.id.idListeCours);
        listView.setAdapter(adapter);
    }

    private void initialiserListe(ArrayList<Cours> listeCours){
        listeCours.add(new Cours("C/C++", R.drawable.animal1));
        listeCours.add(new Cours("Java", R.drawable.animal2));
        listeCours.add(new Cours("HTML", R.drawable.animal3));
        listeCours.add(new Cours("CSS", R.drawable.animal4));
        listeCours.add(new Cours("JavaScript", R.drawable.animal1));
        listeCours.add(new Cours("Visual Basic", R.drawable.animal2));
        listeCours.add(new Cours("PHP", R.drawable.animal3));
        listeCours.add(new Cours("MySQL", R.drawable.animal4));
    }
}
```

Exercice 4

- Écrire une application Android qui crée une ListeView personnalisée qui affiche la liste des voitures d'un concessionnaire de véhicule en respectant le format de l'écran suivant :
  - Modèle (à choisir vos propres modèle)
  - Couleur
  - Prix (à choisir vos propres devises)
  - Image du véhicule (à choisir vos propres images)

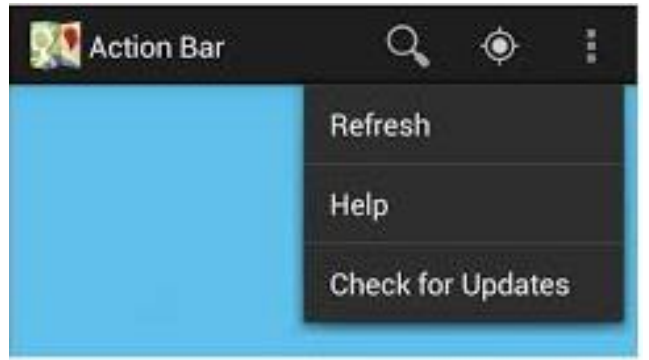


ActionBar

- Une barre d'action (Action Bar) est un élément qui se place en haut de l'écran de l'application et persiste quelque soit l'écran (sauf si le contraire est spécifié dans un écran)
- L'objectif d'une ActionBar est de :
  - Rendre facilement accessibles les actions les plus importantes d'une application
  - Permettre de naviguer plus facilement dans l'application

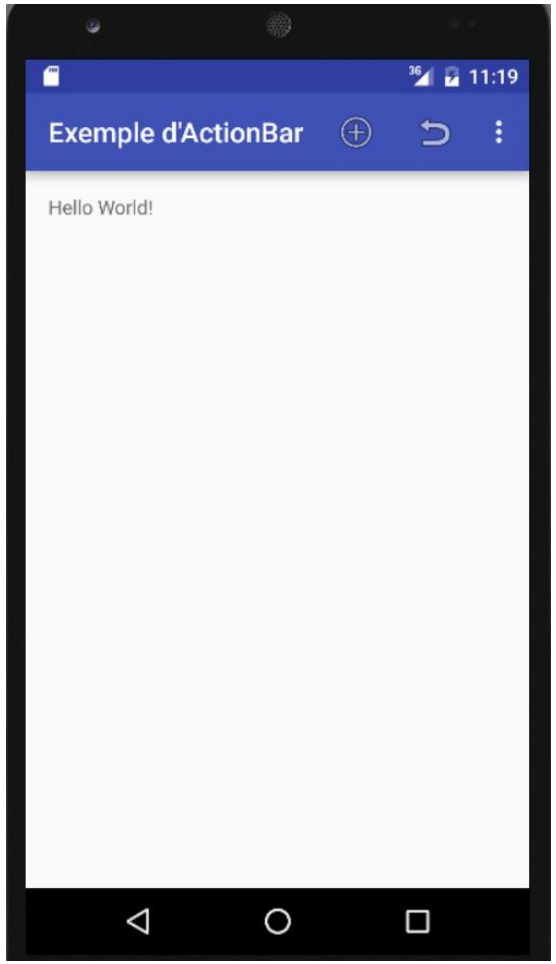
ActionBar

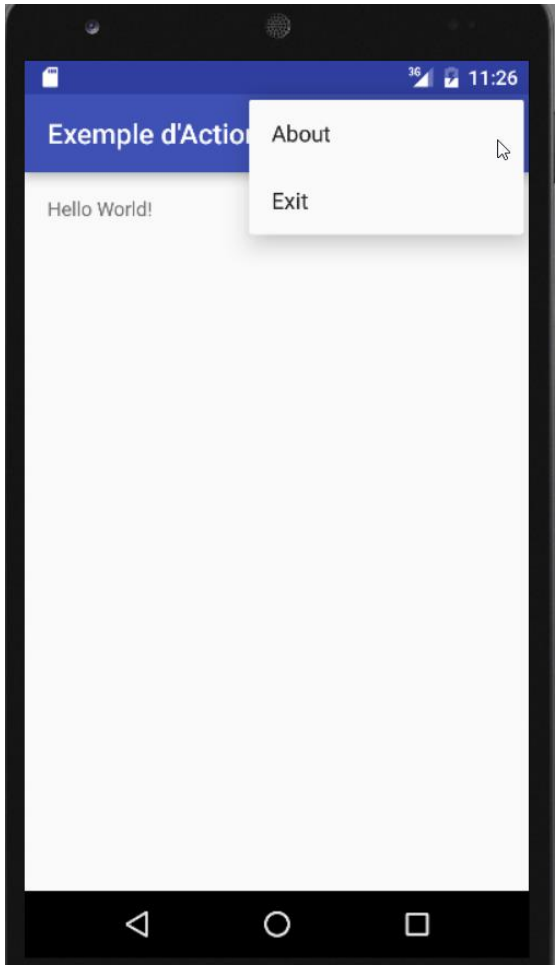
- L'objectif d'une ActionBar est de :
  - Rendre facilement accessibles les actions les plus importantes d'une application
  - Permette de naviguer plus facilement dans l'application
- Une ActionBar se compose en général de plusieurs parties :
  - L'icône de l'application : Permet d'établir l'identité d'une application et de
  - Navigation entre vues : Cet élément permet à l'utilisateur de naviguer entre plusieurs vues
  - Boutons d'actions : Affiche les fonctionnalités les plus importantes d'une application pour un accès rapide et simple
  - Actions supplémentaires : Permet d'accéder aux fonctionnalités (moins importantes) de l'application (Les 3 petits points) 



Exercice 5

- Réaliser une application qui affiche l'écran suivant avec l'ActionBar





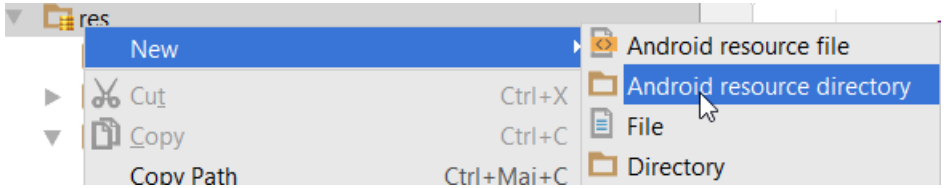
Solution – fichier : strings.xml

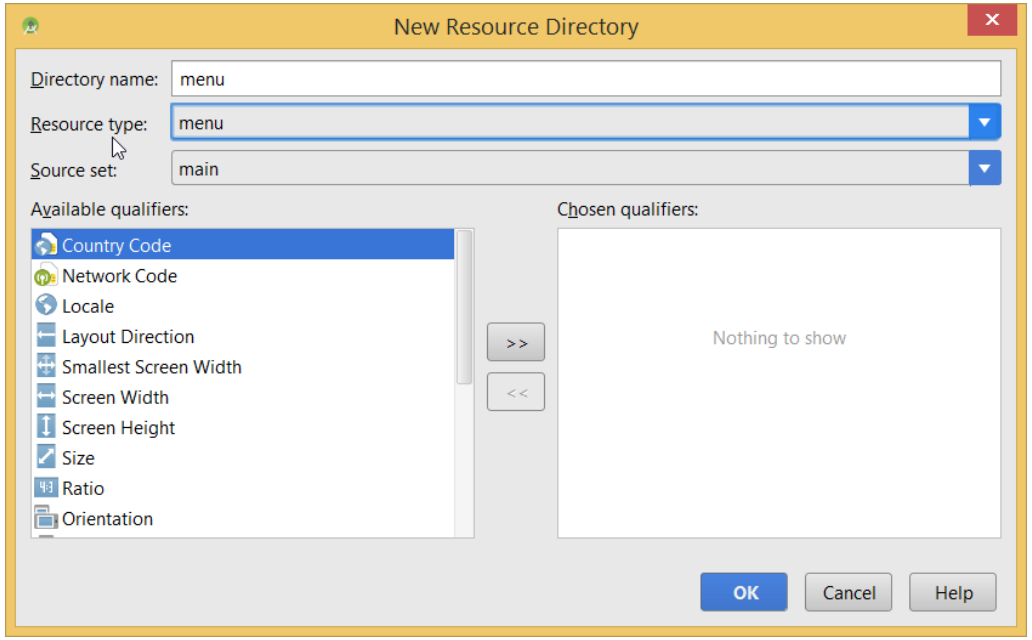
```
<resources>
    <string name="app_name">Exemple d\'ActionBar</string>
    <string name="add">Add</string>
    <string name="reset">Reset</string>
    <string name="about">About</string>
    <string name="exit">Exit</string>
</resources>
```

Définition du nom de quelques actions à présenter dans l'ActionBar dans le fichier Strings.xml

Solution : Création de dossier menu qui contiendra les actions de la barre d'action

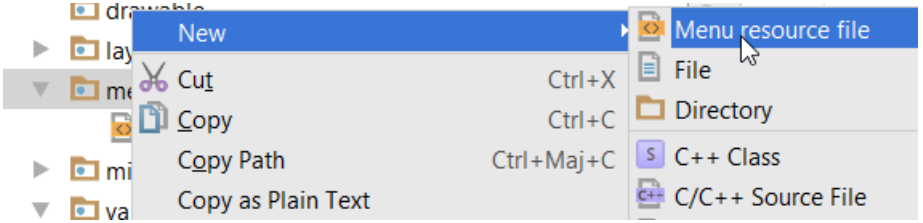
- Clic droit sur le dossier res du projet
- New/Android ressource directory
- Resource type : menu
- Nom du dossier : menu

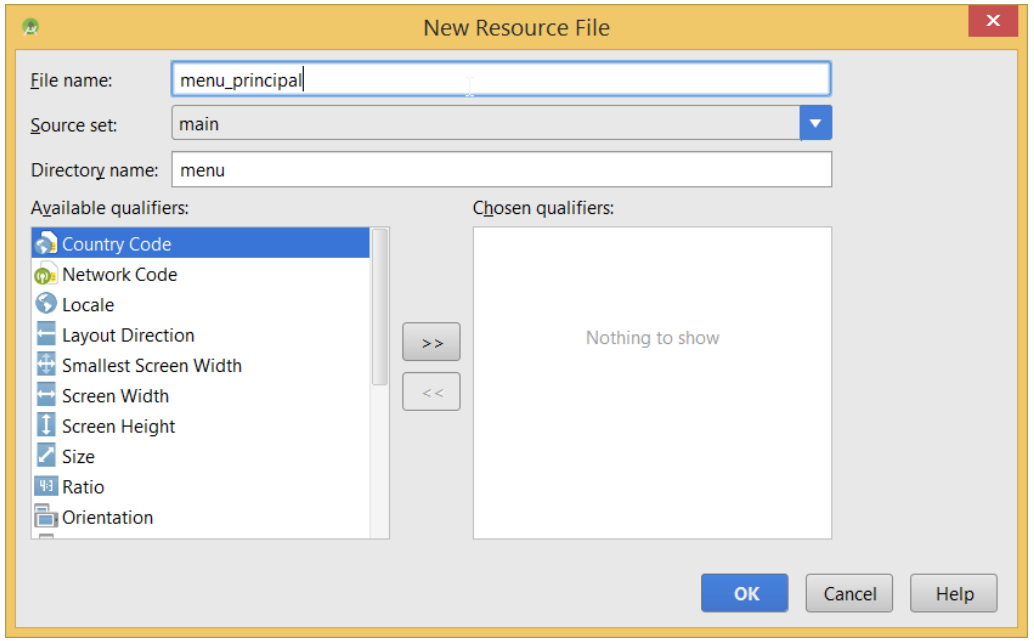




Solution : Création du fichier menu_principal.xml dans le dossier res/menu

- Clic droit sur menu
- New/Menu resource File
- Nom du fichier : menu_principal
- Ajouter les code de la diapo suivante dans le fichier menu_principal.xml





Code du fichier menu_principal.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity">

    <item
        android:id="@+id/add"
        android:icon="@android:drawable/ic_menu_add"
        app:showAsAction="always"
        android:title="@string/add"/>
    <item
        android:id="@+id/reset"
        android:icon="@android:drawable/ic_menu_revert"
        app:showAsAction="always"
        android:title="@string/reset"/>
    <item
        android:id="@+id/about"
        app:showAsAction="never"
        android:title="@string/about">
    </item>
    <item
        android:id="@+id/exit"
        app:showAsAction="never"
        android:title="@string/exit">
    </item>
</menu>
```

Voir les explications dans les deux prochaines diapos

Les actions de ActionBar

- La déclaration des actions qui composent une ActionBar commence par la balise ***menu***
- Chaque action correspond à un élément de type ***item***
- Chaque item peut posséder plusieurs attributs dont les plus importants :
  - **Un identifiant** : permet d'interagir avec l'action au moment du clic
  - **Une icône** : qui représente l'action qui sera stockée dans le dossier drawable ou une icône par défaut d'Android
  - **Un titre** : Le titre de l'action sera affiché si l'utilisateur effectue un long appui sur l'icône (permet à l'utilisateur de voir le nom de l'action) ou si l'action se trouve dans la section (autres actions)
  - **L'ordre des actions** : (**android:orderInCategory**= "2") attribut qui permet de spécifier l'ordre d'affichage des actions
  - **Le comportement de l'actionBar** (**android:showAsAction**) : qui peut prendre les valeurs suivantes : ***ifRoom***, ***withText***, ***never***, ***always***, ***collapseActionView***

Le comportement de l'actionBar

- **ifRoom** : placer l'élément dans l'ActionBar seulement si une place est disponible (recommandé)
- **withText** : inclure le texte de l'élément dans l'actionBar
- **never** : Ne jamais placer dans la barre d'action (toujours placés dans la partie autres actions)
- **always** : Toujours placer l'élément dans la barre d'action.
  - Il est recommander de ne pas trop utiliser cette valeur sauf pour les fonctionnalités critiques
- **collapseActionView** : cette action est retraçable depuis l'Api 14

Code de l'activité : MainActivity.java

- **Surcharger la méthode onCreateOptionsMenu**

```
public class MainActivity extends AppCompatActivity {

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

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

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch(item.getItemId()) {
            case R.id.add:
                //add the function to perform here
                return(true);
            case R.id.reset:
                //add the function to perform here
                return(true);
            case R.id.about:
                //add the function to perform here
                return(true);
            case R.id.exit:
                //add the function to perform here
                return(true);
        }
        return(super.onOptionsItemSelected(item));
    }
}
```

Références

- https://developer.android.com/guide/platform/index.html
- Guide de développement d'applications Java pour Smartphones et Tablettes – Sylvain Hébuterne et Sebastien Perochon – Eni – 2015