

TP2_ Android

Construction Graphique d'une l'interface sous Android

1. Fondement de la conception d'interface sous Android

Android introduit une nouvelle terminologie pour des métaphores de programmation familières et qui sont explorés en détails dans ce TP et dans ce qui suit :

- Vues : Les vues sont les classes de base pour tous les éléments visuels d'interface (communément appelés widgets).ils sont dérivés de la classe View.
- Groupes de vues : C'est l'extension de la classe View pour former la classe ViewGroup. Cette classe permet de créer des composites (vues ou widgets) qui peuvent être interconnectées et organisées dans une disposition bien définie pour former ce qu'on appelle les Layouts.
- Fragment : Les fragments introduits par Android 3.0 servent à encapsuler des portions de l'interface. Chaque fragment contient son propre Layout et reçoit les évènements d'entrée concernés, mais il est étroitement lié à l'activité dans laquelle il doit être intégré.
- Activités : représentent les fenêtres ou les écrans d'affiche. Elles sont les équivalents Android des Windows.

2. Les activités

Le répertoire src/ de votre projet contient une arborescente de répertoires Java classique. Dans le répertoire le plus bas, vous trouvez un fichier source portant le nom de l'activité créée lors de création du projet. En ouvrant ce fichier, vous allez trouver au début le code suivant.

```
package com.example.helloworld; ← nom du package

import android.support.v7.app.ActionBarActivity;
import android.support.v7.app.ActionBar;
import android.support.v4.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;

import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.os.Build;
```

toutes les classes susceptibles d'être utilisées dans le projet

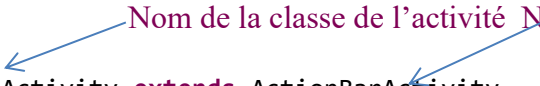
Toute classe jugée non nécessaire pour le projet doit être éliminée. Pour notre cas, on va les laisser car on va les explorer ultérieurement tout au long des TP une à une. Sous Android Studio dans sa dernière version voici le code généré lors de la création d'une application :

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Et donc en comparant les deux codes, on remarque que avec l'Eclipse Android génère dans le code tout et laisse le soin au programmeur d'enlever le non nécessaire alors que Android Studio et par souci de place mémoire il génère dans son code le minimum et laisse le soin au programmeur d'ajouter ce qui lui est nécessaire.

Nom de la classe de l'activité Nom de la classe de l'héritage

public class MainActivity **extends** ActionBarActivity

L'annotation `@Override` doit être utilisée lorsqu'une méthode redéfinit la méthode de la superclasse.

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
```

L'objet **Bundle** passé en paramètre de la méthode **onCreate** permet de restaurer les valeurs des interfaces d'une activité qui a été déchargée de la mémoire. En effet, lorsque l'on appuie par exemple sur la touche *Home*, en revenant sur le bureau, Android peut être amené à décharger les éléments graphiques de la mémoire pour gagner des ressources. Si l'on rebascule sur l'application (appui long sur *Home*), l'application peut avoir perdu les valeurs saisies dans les zones de texte. L'objet Bundle permet de sauvegarder les interfaces de l'activité pour pouvoir la restaurer

Le code suivant permet de créer l'activité de type Bundle pour pouvoir la sauvegarder.

```
    super.onCreate(savedInstanceState);
```

Le mot-clé **super** est utilisé pour appeler des fonctions définies sur l'objet parent. La méthode suivante permet d'associer la vue graphique nommée `activity_main` à l'activité principale créée dans l'objet Bundle.

```
    setContentView(R.layout.activity_main);
```

Le fichier `R.java` contient la définition de la classe des ressources `R`. cette classe est générée par Android. Eclipse régénère automatiquement la classe `R`, dès que des changements sur le projet le nécessitent.

Le reste du code on l'expliquera au fur et à mesure qu'on avance dans les TP.

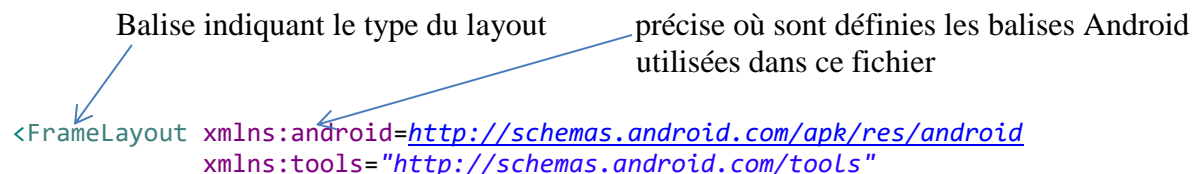
3. Les Layouts

Les Layouts sont utilisés pour organiser les widgets et optimiser l'utilisation de l'écran du mobile. Ils proposent une prédisposition des objets graphiques. Les layouts appartiennent à 4 classes :

1. Frame Layout : il épingle chaque widget dans le coin supérieur gauche et permet de les empiler les uns sur les autres. Si par exemple vous souhaitez faire un album photo, il vous suffit de mettre plusieurs éléments dans le FrameLayout et de ne laisser qu'une seule photo visible, en laissant les autres invisibles grâce à l'attribut `android:visibility` (cet attribut est disponible pour toutes les vues). Pareil pour un lecteur de PDF, il suffit d'empiler toutes les pages dans le FrameLayout et de n'afficher que la page actuelle, celle du dessus de la pile, à l'utilisateur.
2. Linear Layout : il aligne chaque vue ou widget verticalement ou horizontalement.
3. Relative Layout : Il permet de définir les positions de chaque vue par rapport aux autres et aux limites de l'écran.
4. Grid Layout : Introduit par Android 4.0, il utilise une grille rectangulaire de fines lignes et colonnes. Il est plus souple que les autres layouts.

Il est préférable de créer ces layouts avec l'éditeur des layouts plutôt que de le créer manuellement en fichier XML ou avec des instantiations java.

Maintenant ouvrez votre projet fait au niveau de TP1 et cliquez sur le répertoire `res` puis sur `layout`. Sous ce répertoire, vous trouvez un fichier portant le nom de votre layout et d'extension XML, il contient le code XML du layout. Juste à droite de la structure de votre projet, vous avez une petite fenêtre appelée palette, elle contient les layouts et tous les styles de widgets qui peuvent figurer dans ces derniers. Cliquez maintenant sur le répertoire `layout` de la palette et vous allez découvrir les types de layouts précédemment définis. Dans une fenêtre à côté de la palette, vous avez la représentation graphique de votre layout. Ouvrez le fichier XML de votre layout, il contient le code suivant :



```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
```

A l'intérieur de cette balise, on y trouve un ensemble d'attributs selon le format

```
    plateforme:caractéristique="valeur"

    android:id="@+id/container"
```

Chaque layout a un identificateur définie par l'attribut id, comme le montre l'écriture ci-dessus. Le '+' signifie que cet id est nouveau et doit être généré dans la classe R. Un id sans '+' signifie que l'on fait référence à un objet déjà existant.

Chaque layout a un ensemble d'attributs, les plus importants, pour le moment, sont width et height qui définissent la largeur et la hauteur de la vue. Ils peuvent prendre deux valeurs, fill_parent ou match_parent et wrap_content. Les deux premières valeurs indiquent que la vue doit s'étendre pour occuper tout l'espace de l'activité à laquelle elle appartient. La deuxième valeur, indique que la vue doit occuper uniquement l'espace nécessaire à son affichage clair.

```
android:layout_width="match_parent"
android:layout_height="match_parent"

tools:context="com.example.helloworld.MainActivity"
```

L'attribut context définit le contexte d'affichage de votre vue qui dans notre cas l'activité principale.

Fin de la balise frame layout

```
tools:ignore="MergeRootFrame" </>
```

tools:ignore permet d'ignorer certains problèmes qui peuvent survenir lors de l'écriture de notre layout. Dans notre cas, le fait d'écrire la balise Merge au lieu de Frame layout.

4. Les composantes graphiques

La brique élémentaire de construction des interfaces graphiques est donc le widget. Ces composantes sont créées graphiquement et le code xml est généré automatiquement dans le fichier layout.

4.1 Le TextView

C'est le widget le plus basique. Evidemment la classe définit de nombreux attributs Java et ses équivalents XML pour gouverner finement sa représentation (couleur, police, dimension,...) et son comportement (conversion automatique des adresses email, numéros de téléphones,...).

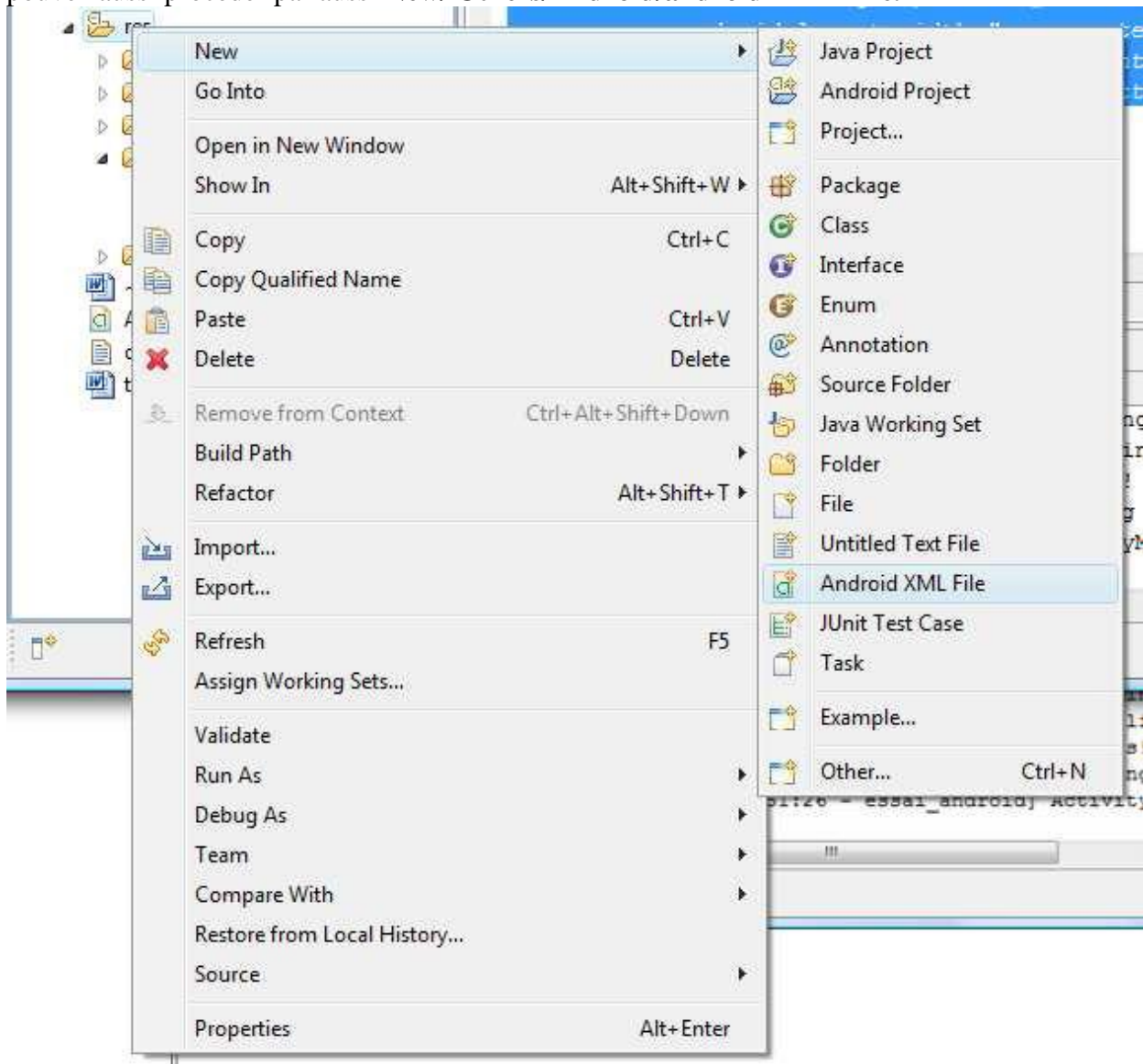
Dans votre représentation graphique du layout, faites glisser le widget TextView dans votre layout, vous allez remarquer qu'Android le place automatiquement en haut gauche (car on a le frame layout). Basculez sur votre fichier XML maintenant, vous allez voir le code suivant qui s'ajoute automatiquement.

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="TextView" />
```

C'est la balise XML pour la codification du TextView, on peut ajouter l'attribut style comme suit : android:textStyle='italic' ou android:textStyle='bold' ou encore l'attribut typeface pour définir le type de police comme suit : android:typeface='serif'.

Travail à faire

Pour pouvoir ajouter d'autres widgets, il faut changer de type de layout, pour ceci nous allons choisir le type le plus simple qui est le `LinearLayout`. En travaillant sur le même projet `Helloandroid`, faites un clic droit sur **res**, puis choisissez **New / Android XML File**. Vous pouvez aussi procéder par aussi **New/ Others/Android/android XML File**.



Choisir comme nom : **main_linear_layout** et comme type `LinearLayout` puis appuyer sur Next. Une autre fenêtre montrant quelques widgets qu'on peut ajouter au niveau de notre layout, appuyer sur finish. Le fichier xml généré contient le code suivant :

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

android:orientation peut être "vertical" ou "horizontal".

1. Ajouter le même textview que le frame layout et mettre android:text="Login". Pour espacer les différents widget dans le layout, on peut introduire l'attribut android:padding.

2. Ajouter la ligne suivante dans tous les widgets utilisés afin qu'ils soient plus ergonomiques android:padding="5dp".

Le dp, density independant pixels, est une unité abstraite basée sur la densité physique de l'écran

$$dp = (\text{résolution de l'écran en dpi} / 160) \text{ pixels}$$

Pour un écran de résolution 320dpi, 1dp=2pixels.

4.2 EditText

L'EditText est une extension du TextView, ce widget est un champ de texte éditable. Des masques de saisie peuvent être rattachés. Dans la palette de votre layout linéaire, cliquez sur TextFields, vous allez voir tous les types de textes éditables, vous n'avez qu'à cliquer sur le masque voulu et le déplacer sur votre layout, le code xml de l'EditText sera généré automatiquement dans le fichier xml du layout. Pour notre TP, on va choisir un TextView de type person name pour la saisie du login utilisateur par exemple.

Les attributs de base pour les textes éditables sont :

```
<EditText
    android:id="@+id/editText1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    >
    <requestFocus />
```

android:ems donne la taille de police utilisée pour ce texte de saisie, 1em vaut 2inch. Le requestFocus permet de mettre en garde au layout du texte éditable pour lui dire qu'on attend de l'utilisateur une saisie de texte. Pour notre cas, un autre attribut est ajouté au fichier xml :

```
    android:inputtype="TextPersonName"
```

il indique le masque de saisie du champ éditable.

3. Ajouter un TextView Password et un EditText de type password.

4.3 Bouton

La classe Button permet de créer un bouton qui pourra être actionné par l'utilisateur pour déclencher un traitement donné. Pour ajouter un bouton à notre layout, il suffit de basculer à la représentation graphique et dans palette de cliquer sur le répertoire From widget et de faire glisser le ok button jusqu'au layout, le code xml suivant est généré

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button" />
```

Changer le texte du bouton a `android:text="Test_Button"`.

Deux autres types de boutons peuvent aussi être utilisés sous android :

- **ToggleButton** : bouton a deux états pouvant servir d'alternative à une checkBox. Il est tout particulièrement pratique lorsque l'appui du bouton doit lancer une action et changer un état (comme lancer ou stopper quelque chose).
- **checkBox** : bouton a deux états représentant une tâche/attribut à cocher.

4.Ajouter un ToggleButton en faisant glisser l'icône de ToggleButton dans le layout. Le texte xml suivant s'ajoute dans le fichier xml du layout.

```
<ToggleButton
    android:id="@+id/toggleButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="ToggleButton" />
```

Ce qu'on voit afficher sur l'icône est l'état off, alors on va changer le texte pour ce bouton dans le cas où ce bouton est activé et son état désactivé. Ajouter les deux lignes suivantes dans le fichier xml et dans la balise de ToggleButton.

```
    android:textOn="Allume"
    android:textOff="Eteint"
```

5.Ajouter un checkBox en faisant glisser l'icône de checkBox dans le layout et changer le texte de bouton comme suit `android:text="etat_civil"`. Ajuster graphiquement la taille du bouton et vous allez voir que ceci se fait directement dans le fichier xml par un changement au niveau des attributs width ou height.

4.4 RadioGroup

La classe RadioGroup sert à afficher des boutons radio. Les boutons radio définissent un ensemble d'options parmi lesquelles l'utilisateur ne peut choisir qu'une.

6.Ajouter un RadioGroup en faisant glisser son icône dans le layout. Le texte xml suivant s'ajoute :

```
<RadioGroup
    android:id="@+id/radioGroup1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" > } Attributs généraux du groupe

    <RadioButton
        android:id="@+id/radio0"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:text="RadioButton" /> } Attributs spécifique a un bouton radio

    <RadioButton
        android:id="@+id/radio1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="RadioButton" /> } changer ce texte a Rouge

    <RadioButton
        android:id="@+id/radio2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" /> } changer ce texte a vert
```



```

        android:layout_height="wrap_content"
        android:text="RadioButton" />
    </RadioGroup>

```

← changer ce texte a bleu

Ce groupe contient une valeur choisie par défaut, le premier bouton radio c'est pourquoi il porte l'attribut android :checked="true".

4.5 Bouton Image

ImageButton est une sous-classe d'ImageView destinée à recevoir des interactions utilisateur comme un bouton. Lors d'un clic, l'image se verra entourée d'un cadre orange.

7. Ajouter un bouton image, cliquer dans palette sur le répertoire Image&Media et faites glisser l'icône ImageButton jusqu'au layout, une boîte de dialogue s'ouvre alors devant vous et vous demande soit de choisir un icône parmi ceux disponible sous android, soit de créer votre propre icône. Dans la liste fournis par android, choisissez le logo d'android (le dernier icône proposé sur la liste) et cliquez sur OK. Le texte xml correspondant est le suivant :

```

<ImageButton
    android:id="@+id/imageButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher" />

```

Dans le répertoire Image&Media, on y trouve aussi l'icône imageView qui permet d'afficher une image dans le layout d'ailleurs il a les mêmes attributs que l'ImageButton sauf que dans l'attribut android:src, vous devez donner le fichier contenant la petite image et qui doit être sauvegardé sous le répertoire res/drawable/.

8. Choisissez quelques widgets et refaites les mêmes étapes mais maintenant en utilisant le Grid Layout, puis Relative layout pour voir la différence dans la gestion des vues.