Applications mobiles: Interfaces graphiques

DÉPARTEMENT D'INFORMATIQUE – CENTRE UNIVERSITAIRE DE MILA

2017/2018

1. Layout

Les éléments graphiques héritent de la classe **View.** On peut regrouper des éléments graphiques dans des **ViewGroups**.

Des **ViewGroups** particuliers sont prédéfinis: ce sont des (*layout*) qui proposent une prédispositions des objets graphiques:

LinearLayout: Dispose les éléments de gauche à droite ou du haut vers le bas

RelativeLayout: Les éléments enfants sont placés les uns par rapport aux autres

TableLayout: Disposition matricielle.

FrameLayout: Disposition en haut à gauche en empilant les éléments

GridLayout: Disposition matricielle avec N colonnes et un nombre infini de lignes

1.1. *Attributs des layouts*

❖Les attributs des layouts permettent de spécifier des propriétés supplémentaires. Les plus importants sont:

- **android:layout_width** et **android:layout_height**:
  - **="match_parent"**: L'élément remplit tout l'élément parent
  - **="wrap_content"**: Prend la place minimum nécessaire à l'affichage
- **android:orientation**: Définit l'orientation d'empilement
- **android:gravity**: Définit l'alignement des éléments

1.1. *Attributs des Layout*

Voici un exemple de **LinearLayout**:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:id="@+id/mainActivity">
</LinearLayout>
```

1.1. *Attributs des Layouts*

Exemples de ViewGroup: Un **ViewGroup** contient des éléments graphiques.

Pour construire un layout linéaire on fera:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:id="@+id/mainActivity"
        >
```

1.1. *Attributs des lyouts*

<TextView android:id="@+id/le_texte" android:text="@string/hello" **/>**

<TextView android:id="@+id/le_texte2" android:text="@string/hello2" **/>**

</LinearLayout>

1.1. *Attributs des layout*

Alors que pour empiler une image et un texte:

```
<FrameLayout  xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/mainActivity"
>
```

1.1. *Attributs des gabarits*

```
>
        <ImageView
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:src="@drawable/mon_image" />
        <TextView android:id="@+id/le_texte" android:text="@string/hello" />
</FrameLayout>
```

2. *Les Views en java*

Une interface graphique définie en XML sera aussi générée comme une ressource dans la classe statique **R**. Le nom du fichier xml, par exemple *mainActivity.xml* permet de retrouver le layout dans le code java au travers de **R.layout. *mainActivity***.

Ainsi, pour associer la première vue graphique à l'activité principale de l'application, il faut faire:

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

2. *Les interfaces en java*

- Le layout reste modifiable au travers du code, comme tous les autres objets graphiques. Pour cela, il est important de spécifier un id dans la définition XML du Layout (**android:id="@+id/*mainActivity* "**).
- 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.
- En ayant généré un *id*, on peut accéder à cet élément et agir dessus au travers du code Java.

2. Les interfaces en java

```
LinearLayout l = (LinearLayout)findViewById(R.id.mainActivity);

l.setBackgroundColor(Color.BLACK);
```

3. *Les labels de texte*

En XML:

```
<TextView
        android:id="@+id/le_texte"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:layout_gravity="center"
/>
```

3. *Les labels de texte*

En Java:

```
public class Activity2 extends Activity {
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout  LL= new LinearLayout(this);
        LL.setGravity(Gravity.CENTER); // centrer les éléments graphiques
        LL.setOrientation(LinearLayout.VERTICAL); // empiler vers le bas !
        TextView texte = new TextView(this);
```

3. *Les labels de texte*

```
        texte.setText("Programming creation of interface !");
        LL.addView(texte);
        setContentView(LL);
} }
```

4. *Les zones de texte*

En XML:

```
<EditText android:text=""
        android:id="@+id/EditText01"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
</EditText>
```

4. *Les zones de texte*

En java:

```
EditText edit = new EditText(this);
edit.setText("Edit me");
LL.addView(edit);
```

5. *Les images*

En XML:

```
<ImageView
        android:id="@+id/logo"
        android:src="@drawable/im"
        android:layout_width="100px"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal">
</ImageView>
```

5. *Les images*

En Java:

```
ImageView image = new ImageView(this);
image.setImageResource(R.drawable.im);
LL.addView(image);
```

5. *Les boutons*

En XML:

```
<Button android:text="Go !"
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
         android:layout_height="wrap_content">
</Button>
```

5. Les boutons

La gestion des événements de *click* se font par l'intermédiaire d'un listener:

```
Button b = (Button)findViewById(R.id.Button01);
b.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {
        Toast.makeText(v.getContext(), "Stop !", Toast.LENGTH_LONG).show();
}
});
```

6. Toast

- Un toast fournit un retour d'information simple sur une opération dans une petite fenêtre contextuelle.
- Les toasts disparaissent automatiquement après un délai d'attente prédéfinit.

6. Toast

- Toast.makeText(context, text, duration).show()
- Toast t=**new** Toast(**this**);
- t.setGravity(Gravity.***CENTER_VERTICAL***,0,0);
- t.setDuration(Toast.***LENGTH_LONG)***
- t.setView(myView);

7. Layout inflater

- Le Layout inflater permet d'obtenir une référence au fichier XML et ces divers **Views** tels que TextView, EditText view etc.
- Ces **Views** qui sont présents dans le XML, peuvent être convertis en objets Java afin que vous puissiez effectuer diverses opérations avec eux.

7. Layout inflater

- LayoutInflater inflat=getLayoutInflater();
- View myView=inflat.inflate(int resource: R.layout.***newview***, ViewGroup root: (ViewGroup) findViewById(R.id.***MyView***));
- TextView Tv=(TextView)myView.findViewById(R.id.***textView***);
- Tv.setText(**"Troisième année info"**);