Création d'interfaces utilisateur

Équipe Pédagogique Informatique@ UVCI 2019

Janvier 2019
Version 1.0

Table des matières

Objectifs

À la fin de cette leçon, l'apprenant sera capable de :

- Créer les interfaces utilisateur avec Android;
- Décrire les notions de ressources en Android
- Décrire les notions d'ergonomie des applications mobile avec Android
- Décrire les composants d'une interface Android
- Décrire les notions de styles

Introduction

la création d'interfaces utilisateur est l'objet du cours de cette semaine.

Les notions abordées sont :

- *Activités*
- *Relations entre un source Java et des ressources*
- *Layouts et vues*
- *Styles*

Cette partie porte essentiellement sur la *mise en page* .

Applications et activités

Objectifs

Créer les interfaces utilisateur avec Android;

1. Composition d'une application

L'interface utilisateur d'une application Android est composée d'écrans.

Un écran correspond à une *activité*,

Exemple :

- *afficher* des informations
- *éditer* des informations

Android permet de naviguer d'une activité à l'autre,

Exemple :

- une action de l'utilisateur, *bouton*, *menu* ou l'*application* fait aller sur l'écran suivant
- le bouton *back* ramène sur l'écran précédent.

⚠ *Attention*

Les *dialogues* et les *pop-up* ne sont pas des activités, ils se superposent temporairement à l'écran d'une *activité*.

2. Création d'un écran

Chaque écran est géré par une instance d'une sous-classe personnel de *Activity*

Sa méthode `onCreate` définit, entre autres, ce qui doit être affiché sur l'écran

```
1 public class MainActivity extends Activity {
2  @Override
3  protected void onCreate(Bundle savedInstanceState) {
4    super.onCreate(savedInstanceState);
5    setContentView(R.layout.main);
6  }
7 }
```

L'interface est mise en place par `setContentView` (identifiant de ressource).

Exercice

II

[Solution n°1 p 25]

un écran sur une application Android s'appelle :

- ☐ Action
- ☐ Interface
- ☐ Activité

Les ressources

Objectifs

Décrire les notions de ressources en Android

1. Identifiant de ressource

La méthode `setContentView` spécifie l'identifiant de l'interface à afficher dans l'écran : *R.layout.main*.

C'est un entier, identifiant d'une disposition de vues : *un layout*.

Le SDK Android (aapt) construit automatiquement une classe statique appelée : *R*

Elle ne contient que des constantes entières groupées par catégories : *id*, *layout*, *menu*. . . :

```
1 public final class R {
2    public static final class string {
3       public static final int app_name=0x7f080000;
4       public static final int message=0x7f080001;
5    }
6    public static final class layout {
7       public static final int main=0x7f030000;
8    }
9 ......
```

2. La classe R

Cette classe *R* est générée automatiquement parce que vous mettez dans le dossier `res` : dispositions, identifiants, chaînes. . .

Certaines de ces ressources sont des *fichiers XML*, d'autres sont des *images PNG*.

Ci-dessous le contenu du fichier : `res/values/strings.xml` :

```
1 <resources>
2     <string name="app_name">Mon App</string>
3 </resources>
```

Rappel : la structure d'un fichier XML

Un fichier XML est composé de : nœuds racine, éléments, attributs, valeurs, texte.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <racine>
3 <!-- commentaire -->
4 <element attribut1="valeur1"
5     attribut2="valeur2">
6     <feuille1 attribut3="valeur3"/>
7     <feuille2>texte</feuille2>
8 </element>
```

```
texte en vrac
</racine>
```

Plus de détails sur *XML ici*

3. Espaces de nommage dans un fichier XML

Dans le cas d'Android, il y a un grand nombre d'éléments et d'attributs normalisés.

Pour les distinguer, ils ont été regroupés dans le namespace android.

Dans la norme XML, le namespace par défaut n'est jamais appliqué aux attributs, donc il faut mettre le préfixe sur chacun d'eux.

Lire *cette page* pour plus d'informations sur le namespace en XML

```
<menu xmlns:android=
    "http://schemas.android.com/apk/res/android">
<item
    android:id="@+id/action_settings"
    android:orderInCategory="100"
    android:showAsAction="never"
    android:title="Configuration"/>
</menu>
```

4. Création d'une interface par programme

Il est possible de créer une interface par programme, comme avec JavaFX et Swing, mais c'est assez compliqué :

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Context ctx = getApplicationContext();
        TextView tv = new TextView(ctx);
        tv.setText("Demat !");
        RelativeLayout rl = new RelativeLayout(ctx);
        LayoutParams lp = new LayoutParams();
        lp.width = LayoutParams.MATCH_PARENT;
        lp.height = LayoutParams.MATCH_PARENT;
        rl.addView(tv, lp);
        setContentView(rl);
    }
```

5. Programme et ressources

Il est donc préférable de stocker l'interface dans un fichier *res/layout/main.xml* :

```
<RelativeLayout ...>
    <TextView android:text="Demat !" ... />
</RelativeLayout>
```

qui est référencé par son identifiant *R.layout.nom_du_fichier* (ici c'est main) dans le programme Java

```

    protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView(R.layout.main);
    }
```

6. Ressources de type chaînes

Dans le fichier *res/values/strings.xml*, on place les chaînes de l'application, au lieu de les mettre en constantes dans le source :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <string name="app_name">HelloWorld</string>
4     <string name="main_menu">Menu principal</string>
5     <string name="action_settings">Configuration</string>
6     <string name="bonjour">Demat !</string>
7 </resources>
```

Remarque

Intérêt : pouvoir traduire une application sans la recompiler.

7. Traduction des chaînes (localisation)

Lorsque les textes sont définis dans *res/values/strings.xml*, il suffit de faire des copies du dossier *values*, en *values-us*, *values-fr*, *values-de*, etc....

et de traduire les textes en gardant les attributs *name*.

Voici par exemple *res/values-de/strings.xml* :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <string name="app_name">HelloWorld</string>
4     <string name="main_menu">Hauptmenu</string>
5     <string name="action_settings">Einstellungen</string>
6     <string name="bonjour">Guten Tag</string>
7 </resources>
```

Le système Android ira chercher automatiquement le bon texte en fonction des paramètres linguistiques configurés par l'utilisateur.

8. Référencement des ressources texte

Voici comment affecter une ressource chaîne à une vue en Java :

```
1 TextView tv = new TextView(ctx);
2 tv.setText(R.string.bonjour);
```

R.string.bonjour désigne le texte de *<string name="bonjour">...* dans le fichier *res/values*/strings.xml*

Voici comment spécifier un titre de label dans un *layout.xml* :

```
1 <RelativeLayout>
2    <TextView android:text="@string/bonjour" />
3 </RelativeLayout>
```

@string/nom est une référence à une ressource, la chaîne de *res/values/strings.xml* ayant ce nom.

9. Identifiants et vues

La méthode *setContentView* fait afficher le formulaire défini par l'identifiant *R.layout* indiqué.

Lorsque l'application veut manipuler l'une de ses vues, elle doit faire utiliser *R.id.symbole*, ex :

Exemple :

```
1 TextView tv = (TextView) findViewById(R.id.message);
```

Remarque

remarquez la conversion de type, *findViewById* retourne une *View*, superclasse de *TextView*.

avec la définition suivante dans *res/layout/main.xml* :

```
1
2 <RelativeLayout>
3 <TextView android:id="@+id/message"
4     android:text="@string/bonjour" />
5 </RelativeLayout>
```

La notation *@+id/nom* définit un identifiant pour le *TextView*.

10. @id/nom ou @+id/nom ?

Il y a les deux notations :

- *@id/nom* pour référencer un identifiant déjà défini (ailleurs)
- *@+id/nom* pour définir (créer) cet identifiant

Exemple

le *Button btn* désigne le *TextView* titre :

```
1 <RelativeLayout xmlns:android="..." ... >
2     <TextView ...
3         android:id="@+id/titre"
4         android:text="@string/titre" />
5     <Button ...
6         android:id="@+id/btn"
7         android:layout_below="@id/titre"
8         android:text="@string/ok" />
9 </RelativeLayout>
```

11. Images : R.drawable.nom

De la même façon, les images PNG placées dans *res/drawable* et *res/mipmaps-** sont référençables :

```
1 <ImageView
2    android:src="@drawable/velo"
3    android:contentDescription="@string/mon_velo" />
```

La notation *@drawable/nom* référence l'image portant ce nom dans l'un des dossiers.

Remarque

les dossiers *res/mipmaps-** contiennent la même image à des définitions différentes, pour correspondre à différents téléphones et tablettes.

Ex: mipmap-hdpi contient des icônes en 72x72 pixels.

12. Tableau de chaînes : R.array.nom

Voici un extrait du fichier *res/values/arrays.xml* :

```xml
<resources>
    <string-array name="planetes">
        <item>Mercure</item>
        <item>Venus</item>
        <item>Terre</item>
        <item>Mars</item>
    </string-array>
</resources>
```

Dans le programme Java, il est possible de faire :

```
Resources res = getResources();
String[] planetes = res.getStringArray(R.array.planetes);

```

13. Autres notations

D'autres notations existent :

- *@style/nom* pour des définitions de *res/style*
- *@menu/nom* pour des définitions de *res/menu*

Certaines notations, *@package:type/nom* font référence à des données prédéfinies, comme :

- *@android:style/TextAppearance.Large*
- *@android:color/black*

Il y a aussi une notation en *?type/nom* pour référencer la valeur de l'attribut nom,

Exemple : *?android:attr/textColorSecondary.*

QUIZZ

IV

Exercice 1

[Solution n°2 p 25]

Dans quel répertoire se trouve les ressources android ?

- ☐ res/.....
- ☐ ressources/....
- ☐ resources/...

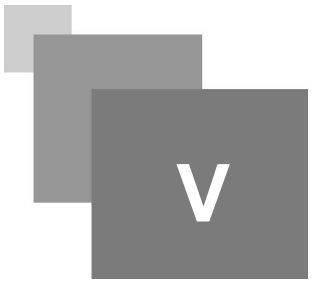
Exercice 2

[Solution n°3 p 25]

Quel est le principal type de fichier utilise dans Android ?

- ○ PNG
- ○ XLM
- ○ XML

Mise en page



Objectifs

Décrire les notions d'ergonomie des applications mobile avec Android

1. Structure d'une interface Android

Un écran Android de type formulaire est généralement composé de plusieurs vues.

on peut citer :

- *TextView*, *ImageView* : titre, image
- *EditText* : texte à saisir
- *Button*, *CheckBox* : bouton à cliquer, case à cocher

Ces vues sont alignées à l'aide de groupes sous-classes de *ViewGroup*, éventuellement imbriqués :

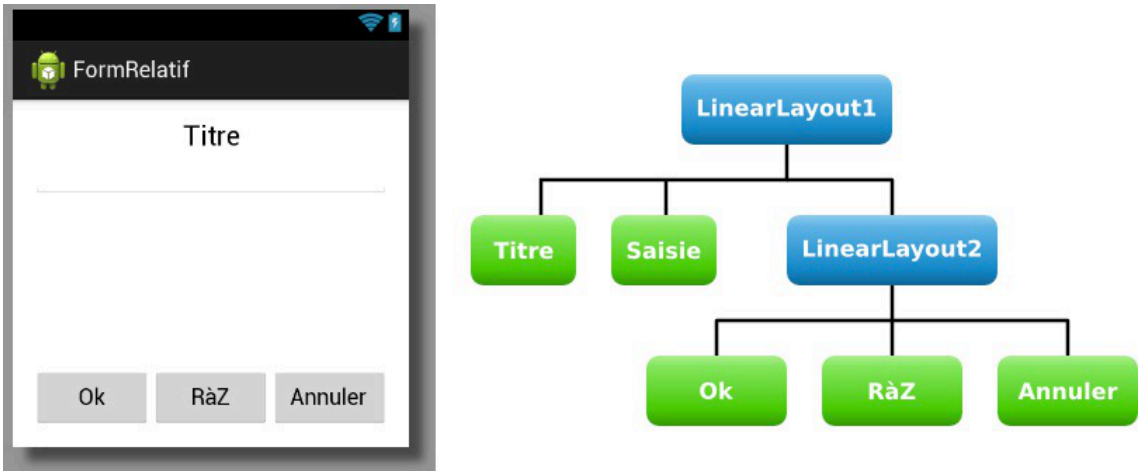
- *LinearLayout* : positionne ses vues en ligne ou colonne
- *RelativeLayout* : positionne ses vues l'une par rapport à l'autre
- *TableLayout* : positionne ses vues sous forme d'un tableau

Cliquez-ici pour plus de détails sur les groupes

2. Arbre des vues

2.1. Présentation

Les groupes et vues forment un arbre :



2.2. Représentation en XML

Cet arbre s'écrit en XML :

```
1 <LinearLayout android:id="@+id/groupe1" ...>
```

```
    <TextView android:id="@+id/titre" .../>
    <EditText android:id="@+id/saisie" .../>
    <LinearLayout android:id="@+id/groupe2" ...>
        <Button android:id="@+id/ok" .../>
        <Button android:id="@+id/raz" .../>
        <Button android:id="@+id/annuler" .../>
    </LinearLayout>
</LinearLayout>
```

Complément

Pour plus de détails sur les arbres de vues *ici*

3. Paramètres de positionnement

La plupart des groupes utilisent des paramètres de placement sous forme d'attributs XML.

Par exemple, telle vue à droite de telle autre, telle vue la plus grande possible, telle autre la plus petite.

Ces paramètres sont de deux sortes :

- ceux qui sont demandés pour toutes les vues :
  - `android:layout_width`
  - `android:layout_height`,
- ceux qui sont demandés par le groupe englobant et qui en sont spécifiques, comme
  - `android: layout_weight`,
  - `android:layout_alignParentBottom`,
  - `android:layout_centerInParent. . .`

4. Paramètres généraux

Toutes les vues doivent spécifier ces deux attributs :

- largeur de la vue : `android:layout_width`
- hauteur de la vue : `android:layout_height`

Ils peuvent valoir :

- `"wrap_content"` : la vue est la plus petite possible
- `"match_parent"` : la vue est la plus grande possible
- `"valeurdp"` : une taille fixe, ex : "100dp" mais c'est peu recommandé, sauf 0dp pour un cas particulier,

Complément : Unité de mesure dp

Les dp sont une unité de taille indépendante de l'écran.

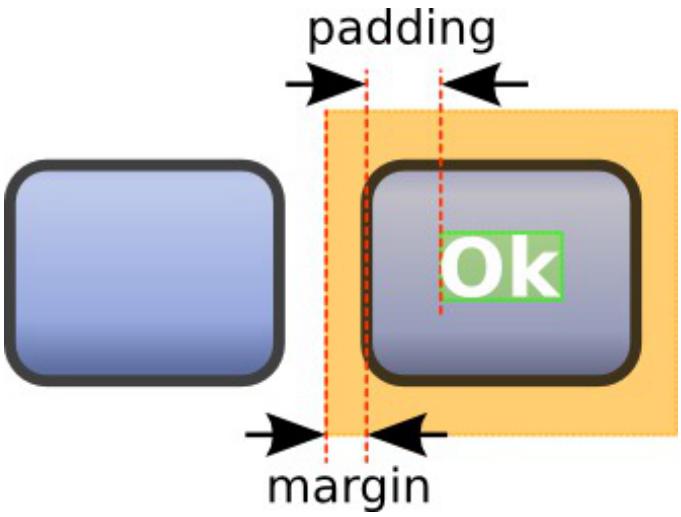
100dp font 100 pixels sur un écran de 100 dpi (100 dots per inch) tandis qu'ils font 200 pixels sur un écran 200dpi.

Ça fait la même taille apparente.

5. Autres paramètres géométriques

Il est possible de modifier l'espacement des vues, comme en css:

- *Padding* espace entre le texte et les bords, géré par chaque vue
- *Margin* espace autour des bords, géré par les groupes



6. Marges et remplissage

On peut définir les marges et les remplissages séparément sur chaque bord (*Top, Bottom, Left, Right*), ou identiquement sur tous

```
<Button
    android:layout_margin="10dp"
    android:layout_marginTop="15dp"
    android:padding="10dp"
    android:paddingLeft="20dp" />
```

7. Groupe de vues LinearLayout

Il range ses vues soit *horizontalement*, soit *verticalement*

Il faut seulement définir l'attribut, `android:orientation` à "*horizontal*" ou "*vertical*".

Exemple

```
<LinearLayout android:orientation="horizontal"
android:layout_width="match_parent"
android:layout_height="wrap_content">
<Button android:text="Ok"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
<Button android:text="Annuler"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
</LinearLayout>
```

8. Pondération des tailles

8.1. Présentation

Une façon intéressante de spécifier les tailles des vues dans un `LinearLayout` consiste à leur affecter un *poids* avec l'attribut `android:layout_weight`.

- Un `layout_weight` égal à 0 rend la vue la plus petite possible
- Un `layout_weight` non nul donne une taille correspondant au rapport entre ce poids et la somme des poids des autres vues

Pour cela, il faut aussi fixer la taille de ces vues (Exemple : `android:layout_width`) soit à

"wrap_content", soit à "0dp".

- Si la taille vaut *"wrap_content"*, alors le poids agit seulement sur l'espace supplémentaire alloué aux vues.
- Mettre *"0dp"* pour que ça agisse sur la taille entière.

8.2. Exemple de poids

Voici 4 *LinearLayout horizontaux* de 3 boutons ayant des poids égaux à leurs titres. En 3e ligne, les boutons ont une largeur de 0dp

9. Groupe de vues TableLayout

9.1. Présentation

C'est une variante du LinearLayout : les vues sont rangées en lignes de colonnes bien tabulées. Il faut construire une structure XML comme celle-ci.

```
<TableLayout ...>
    <TableRow>
        <item 1.1 .../>
        <item 1.2 .../>
    </TableRow>
    <TableRow>
        <item 2.1 .../>
        <item 2.2 .../>
    </TableRow>
<TableLayout>
```

⚠ *Attention*

les <TableRow> n'ont aucun attribut.

9.2. Largeur des colonnes d'un TableLayout

Ne pas spécifier *android:layout_width* dans les vues d'un *TableLayout*, car c'est obligatoirement toute la largeur du tableau.

Seul la balise *<TableLayout>* exige cet attribut.

Deux propriétés intéressantes permettent de rendre certaines colonnes étirables. Fournir les numéros

(première = 0).

- *android:stretchColumns* : numéros des colonnes étirables
- *android:shrinkColumns* : numéros des colonnes réductibles

```
<TableLayout
    android:stretchColumns="1,2"
    android:shrinkColumns="0,3"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
```

10. Groupe de vues RelativeLayout

10.1. Présentation

C'est le plus complexe à utiliser mais il donne de bons résultats. Il permet de spécifier la position relative de chaque vue à l'aide de paramètres complexes : (*LayoutParams*)

- Tel bord aligné sur le bord du parent ou centré dans son parent :
  - *android:layout_alignParentTop*,
  - *android:layout_centerVertical. . .*
- Tel bord aligné sur le bord opposé d'une autre vue :
  - *android:layout_toRightOf*,
  - *android:layout_above*,
  - *android:layout_below. . .*
- Tel bord aligné sur le même bord d'une autre vue :
  - *android:layout_alignLeft*,
  - *android:layout_alignTop. . .*

10.2. Utilisation d'un RelativeLayout

Pour bien utiliser un *RelativeLayout*, il faut commencer par définir les vues qui ne dépendent que des bords du Layout : celles qui sont collées aux bords ou centrées.

```
<TextView android:id="@+id/titre"
    android:layout_alignParentTop="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentLeft="true" .../>
```

Puis créer les vues qui dépendent des vues précédentes.

```
<EditText android:layout_below="@id/titre"
    android:layout_alignParentRight="true"
    android:layout_alignParentLeft="true" .../>
```

Et ainsi de suite.

Complément : Autres groupements

Il existe de nombreux autres groupements, qui sont des sous-classe de *ViewGroup*.

Impossible de faire un inventaire dans ce cours, en revanche vous pouvez les consulter *ici*

QUIZZ

VI

Exercice 1

[Solution n°4 p 25]

Cochez les vues d'une inetrface Android

- ☐ TextView
- ☐ LinearLayout
- ☐ ImageView
- ☐ RelativeLayout
- ☐ EditBox
- ☐ CheckBox
- ☐ TableLayout
- ☐ Button

Exercice 2

[Solution n°5 p 25]

Cochez les attributs de ce groupe de vues "<TableRow>

- ☐ <item 1.1 .../>
- ☐ android:stretchColumns="1,2"
- ☐ android:layout_width
- ☐ aucun

Composants d'interface

VII

Objectifs

Décrire les composants d'une interface Android

1. Vues

Android propose un grand nombre de vues,

- *Textes : titres, saisies*
- *Boutons, cases à cocher*
- *Curseurs*

Beaucoup ont des variantes. Ex: saisie de texte = no de téléphone ou adresse ou texte avec suggestion ou . . .

À noter que les vues évoluent avec les versions d'Android, certaines changent, d'autres disparaissent.

Complément

Consulter la doc en ligne de toutes ces vues. On les trouve dans le package *android.widget*.

2. TextView

Le plus simple, il affiche un texte statique, comme un titre. Son libellé est dans l'attribut *android:text*.

```
1 <TextView
2     android:id="@+id/tvtitre"
3     android:text="@string/titre"
4         ... />
```

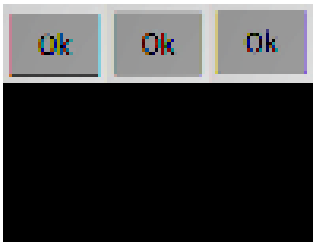
On peut le changer dynamiquement :

```
1 TextView tvTitre = (TextView) findViewById(R.id.tvtitre);
2 tvTitre.setText("blablabla");
```

3. Button

L'une des vues les plus utiles est le *Button* :

L'une des vues les plus utiles est le *Button* :




```
1
2 <Button
3     android:id="@+id/btn_ok"
4     android:text="@string/ok"
5     ... />
```

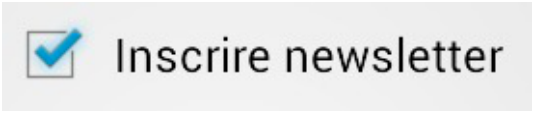
- En général, on définit un identifiant pour chaque vue active, ici : *android:id="@+id/btn_ok"*
- Son titre est dans l'attribut *android:text*.

4. Bascules

4.1. CheckBox

Les *CheckBox* sont des cases à cocher :

Les *CheckBox* sont des cases à cocher :




```
1 <CheckBox
2     android:id="@+id/cbx_abonnement_nl"
3     android:text="@string/abonnement_newsletter"
4     ... />
```

4.2. ToggleButton

Les *ToggleButton* sont une variante

Les *ToggleButton* sont une variante

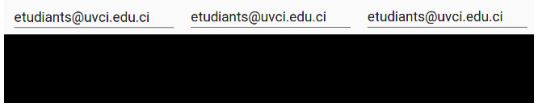
c'est une vue qui a deux états :

- *texte actif* : *android:textOn*
- *texte inactif* : *android:textOff*.

5. EditText

Un *EditText* permet de saisir un texte :

Un *EditText* permet de saisir un texte :




```
1 <EditText
2         android:id="@+id/editText"
```

```
3         android:layout_width="wrap_content"
4         android:layout_height="wrap_content"
5         android:ems="10"
6         android:inputType="textPersonName"
7         android:text="etudiants@uvci.edu.ci"
8         tools:layout_editor_absoluteX="77dp"
9         tools:layout_editor_absoluteY="75dp" />
```

L'attribut *android:inputType* spécifie le type de texte : adresse,téléphone, etc. Ça définit le clavier qui est proposé pour la saisie.

Complément

Lire *la référence Android* pour connaître toutes les possibilités.

Styles

VIII

Objectifs

Décrire les notions de styles

1. Styles et thèmes

1.1. Présentation

Un style permet de modifier l'apparence d'une vue, (pareil comme en HTML+CSS):

- Police de caractères et tailles pour les textes
- Couleurs, images. . .
- Géométrie par défaut des vues : taille, espacement, remplissage. . .

Remarque

Un thème est un style appliqué à toute une activité ou application.

Plus de détails sur les thèmes *ici*

1.2. Définir un style

Il faut créer un fichier XML dans `res/value` :

```
1 <resources>
2     <style name="Elegant"
3         parent="@android:style/TextAppearance.Medium">
4         <item name="android:textColor">#010101</item>
5         <item name="android:typeface">serif</item>
6     </style>
7 </resources>
```

L'attribut `name` identifie le style, et `parent` le rattache à un autre pour héritage des propriétés non définies ici

Complément

Voir les *styles* et les *thèmes* prédéfinis

1.3. Utiliser un style

Il suffit de le préciser dans la définition de la vue :

```
1 <TextView
2     style="@style/Elegant"
3     android:text="@string/titre" />
```

1.4. Utiliser un thème

Un thème est simplement un style appliqué partout dans l'application

Cela se spécifie dans le fichier *AndroidManifest.xml*

```
<application
    android:theme="@style/Elegant"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
        ...>
        ...
</application>
```

⚠ *Attention*

Attention, si votre style n'est pas complet, vous aurez une erreur.

* *
*

Fin !

A la semaine prochiane pour les *écouteurs* et les *activités*

Solutions des exercices

> Solution n°1

Exercice p. 7

- ☐ Action
- ☐ Interface
- ☑ Activité

> Solution n°2

Exercice p. 13

- ☑ res/.....
- ☐ ressources/....
- ☐ resources/...

> Solution n°3

Exercice p. 13

- ○ PNG
- ○ XLM
- ◉ XML

> Solution n°4

Exercice p. 19

- ☑ TextView
- ☐ LinearLayout
- ☑ ImageView
- ☐ RelativeLayout
- ☑ EditBox
- ☑ CheckBox
- ☐ TableLayout
- ☑ Button

Exercice p. 19

> Solution n°5

- ☐ <item 1.1 .../>
- ☐ android:stretchColumns="1,2"
- ☐ android:layout_width
- ☑ aucun