Android – Interface utilisateur

Les applications mobiles sont basées sur une interface utilisateur permettant l'interaction avec l'utilisateur.
La définition de l'interface d'une activité peut être décrite en utilisant le mode design ou en écrivant directement le code dans le fichier de layout. Tous les fichiers de layout sont stockés dans le répertoire res/layout. Chaque fichier de layout est relié à une activité.
Pour tous les fichiers de layout, comme pour toutes les ressources, les noms de fichiers sont en minuscules.

1. Les écrans

Android s'adapte à plusieurs tailles d'écrans des terminaux mobiles. Une autre caractéristique de ces terminaux est la résolution de l'écran qui peut varier de 240x320 pixels à 4096x2160 pixels. Les applications devront donc s'adapter à ces différentes tailles et résolutions.

Le framework Android classe les écrans selon :

- leur taille de diagonale en quatre catégories : small (~6,35 cm), normal (~10cm), large (~18cm) et xlarge (plus de 25 cm) ;
- leur résolution en dpi. Plusieurs résolutions peuvent exister à l'intérieur d'une même catégorie. Android les classe en fonction de la densité : ldpi (120 dpi), mdpi (160 dpi), hdpi (240 dpi), xhdpi (320 dpi), xxhdpi (480 dpi) et xxxhdpi (640 dpi).

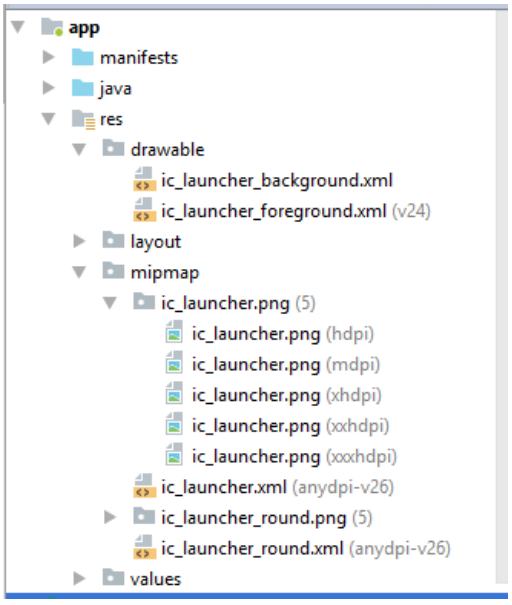
A taille d'écran strictement identique, la taille d'un composant aura une taille d'affichage différente si les résolutions sont différentes. Pour éviter cela, Android préconise d'utiliser une nouvelle unité : le dp ou dip (density-independent pixel). Le système se charge alors de convertir une valeur donnée en dip en nombre de pixels physiques en tenant compte de la densité de l'écran, ce qui permet d'avoir la même taille physique sur tous les écrans.

Android propose plusieurs unités pour spécifier une dimension :

- Dip ou dp : unité indépendante de la densité de l'écran
- Sp ou sip (scale independant pixel) : utilisée pour les tailles de police des chaines de caractères. Elle dépend de la taille préférée de la police de caractères.
- Px : pixel ; in : pouce (2.54 cm) ; mm : millimètre.

On peut spécifier plusieurs variantes pour une même image en fonction de la densité de l'écran. Android recherche la version correspondant à la catégorie de densité de l'écran. S'il ne la trouve pas, il recherche cette ressource dans une autre densité, en privilégiant celle qui lui est supérieure. S'il la trouve, il retaille l'image en appliquant un certain facteur. Ce facteur est appliqué suivant l'échelle suivante : 3-4-6-8-12-16 correspondant respectivement aux densités : ldpi, mdpi, hdpi, xhdpi, xxhdpi, xxxhdpi. Par exemple, une image de 100x100 px pour une densité normale (mdpi) devra être retaillée en 150x150 px pour une densité hdpi (facteur = 6/4 soit 1,5).

Dans l'exemple ici, l'image ic_launcher.png est disponible dans toutes les versions sauf ldpi. C'est donc la version mdpi qui sera transformée (multipliée par le facteur : ¾ soit 0.75) qui sera utilisée si besoin.



Android, pour répondre à la problématique de densité d'écran, propose des répertoires spécifiques à chaque catégorie d'écran. Par exemple, pour les images, si le développeur souhaite fournir à l'application une version de chaque visuel adaptée à chaque densité d'écran, il devra avoir la structure suivante :
/res/drawable : pour stocker les images standards (densité mdpi)
/res/drawable-ldpi : images pour les densités d'écran ldpi
/res/drawable-hdpi : images pour les densités d'écran hdpi
…

2. Mode programmatique et mode déclaratif

Il existe deux manière de concevoir une interface utilisateur :

- Le mode déclaratif : la description de l'interface est réalisée dans des fichiers XML séparés qui se trouvent dans le répertoire /res/layout. Ce mode favorise une architecture MVC. *C'est ce que nous allons utiliser.*
- Le mode programmatique : les composants sont déclarés dans les fichiers Java de l'activité (comme en Swing). L'inconvénient de ce mode est de mélanger la conception de l'interface et le code de l'application.

Pour le mode déclaratif, Android Studio permet d'intervenir sur un fichier de layout directement en l'éditant (onglet Text) ou via une interface graphique (onglet Design).

Rappel : Dans le cas du mode déclaratif, il faut lier le layout à l'activité correspondante. On réalise cette liaison dans la méthode `onCreate()` de l'activité à l'aide de la méthode : `setContentView(R.layout.<nom_layout>)`.

3. Vues

L'élément de base d'une interface utilisateur est la vue (classe View). Tous les autres éléments héritent de cette vue. La classe ViewGroup est un conteneur de vues, elle hérite également de la classe View.

Les propriétés importantes d'une vue sont :

Attribut	**Valeur**	**Signification**
`android:alpha`	`0 / 1`	Opacité du composant 0 : complètement transparent 1 : complètement opaque
`android:background`	`Drawable/color`	Elément à utiliser pour le fond de la vue
`android:clickable`		Spécifie si la vue doit réagir au clic
`android:contentDescription`	`Text`	Description du contenu de la vue
`android:id`	`Text`	Identifiant unique de la vue. Utilisé par view.findViewById() ou Activity.findViewById().
`android:minHeight`	`Valeur`	Hauteur minimale de la vue
`android:minWidth`	`Valeur`	Largeur minimale de la vue
`android:onClick`	`Text`	Nom de la méthode de l'activité qui traite le clic sur la vue
`android:padding`	`Valeur (pixels)`	Dimension de la marge interne pour les 4 côtés de la vue

`android:paddingBottom`
`android:paddingTop`
`android:paddingLeft`
`android:paddingRight`
`android:paddingHorizontal`
`android:paddingVertical`
	`Valeur`	Dimension de la marge interne basse
		Dimension de la marge interne haute
		Dimension de la marge interne gauche
		Dimension de la marge interne droite
		Dimension de la marge interne G/D
		Dimension de la marge interne H/B

`android:visibility`		Visibilité du composant
`android:tag`	`Text`	Associe un objet à la vue

La liste complète des attributs communs aux vues Android est disponible à l'adresse : http://developer.android.com/reference/android/view/View.html#lattrs

4. Les layouts

Les layouts sont des conteneurs de vues (ViewGroup) prédéfinis fournis par le framework Android. Chaque conteneur propose un style d'agencement des vues les unes par rapport aux autres ou par rapport aux conteneurs parents.

Les principaux conteneurs de vues sont : `LinearLayout`, `RelativeLayout`, `TableLayout`, `ConstraintLayout`, GridLayout.

Les fichiers de layout définis dans le répertoire `/res/layout` contiennent en premier élément un conteneur de vue. Ces ressources sont considérées être des ressources pour des écrans de taille normale. Si le développeur souhaite fournir différents layouts pour une même interface (écran) selon la catégorie de la taille de l'écran physique, il faut créer un répertoire avec la taille de l'écran : `/res/layout-small/<fichier_layout>.xml`, `/res/layout-large/<fichier_layout>.xml`…

Si le développeur souhaite créer des layouts différents selon l'orientation (portrait/landscape), il devra créer un répertoire : `/res/layout-port/<fichier_layout>.xml` et un répertoire `/res/layout-land/<fichier_layout>.xml`

Syntaxe :

```
<?xml version="1.0" encoding="utf-8"?>
<TypeLayout 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"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="sio.lc.fr.firstapplication.MainActivity">
</TypeLayout>
```

La 1ère ligne indique la version XML et le type d'encodage utilisés.
La deuxième ligne indique le type de layout utilisé (`LinearLayout, RelativeLayout`….) suivi de ses propriétés.
Le layout racine doit spécifier l'attribut `xmlns:android` qui définit l'espace de noms android qui sera utilisé pour définir les attributs.

Les conteneurs définissent les attributs suivants qui doivent être renseignés par tous les composants enfants :

`android:layout_width`	`match_parent`
	`wrap_content`
	`dimension`
		Même largeur que le conteneur parent moins les marges internes
		Largeur nécessaire plus les marges internes
		Valeur de la largeur. Privilégier en dp

`android:layout_height`	`match_parent` `wrap_content`	Même hauteur que le conteneur parent Hauteur nécessaire

`android:layout_marginBottom`
`android:layout_marginTop`
`android:layout_marginLeft`
`android:layout_marginRight`
	`Valeur`	Marge dans la direction indiquée

5. Widgets

Un widget est un composant de l'interface graphique. La déclaration de ces composants peut être implémentée dans les fichiers de layouts XML ou directement dans les activités Java. Nous allons étudier la définition de ces composants dans les fichiers de layouts.

Syntaxe :

```
<Type_de_widget
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/nom_ressource"
    android:id=="@[+][paquetage :]id/nom_ressource"
    …
/>
```

5.1. Définir un identifiant

Dans le code Java, on a besoin parfois d'utiliser des références sur les composants de la vue. Pour cela, le composant doit avoir un identifiant lors de sa déclaration dans le fichier de layout.

La syntaxe pour définir un identifiant sur un composant est la suivante : `@[+][package :]id/nom_ressource`

`@` : indique au système que le reste de la chaîne de caractères est une ressource
`+` : indique que l'identifiant doit être créé
`id/` : indique que la ressource rajoutée sera de type identifiant
`nom_ressource` : nom utilisé comme référence sur le composant.

Exemple : `@+/tvTitre`

Dans le code Java, la constante suivante : `R.id.nom_ressource` permet de référencer le composant.
Pour manipuler le composant dans le code de l'activité, on va se servir de la méthode `findByViewId()` de la classe `Activity` :

```
View findViewById(int id)
```

La méthode prend en paramètre l'identifiant du composant, à savoir : `R.id.nom_ressource`. Elle renvoie un objet de type `View` qu'il faut convertir en composant typé.

Exemple : `TextView tvTitre = (TextView) findViewById(R.id.tvTitre) ;`

L'obtention des références sur les composants du fichier de layout doit être effectuée dans la méthode onCreate() de l'activité.

5.2. Le composant TextView

Le composant **`TextView`** permet de visualiser du texte sur un écran (activité). Il possède un certain nombre d'attributs, dont les plus importants sont :

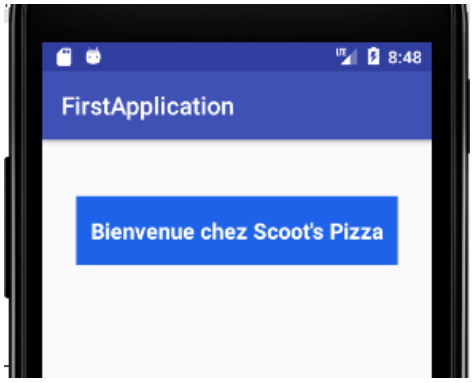
Attribut	Signification
`android:autoLink`	Les liens hypertextes et les adresses mail sont converties en liens cliquables
`android:ellipsize`	Définit comment le texte doit s'afficher s'il est plus long que la vue
`android:gravity`	Indique comment doit être positionné le texte dans la vue (composant)
`android:height`	Hauteur du composant
`android:width`	Largeur du composant
`android:lines`	Nombre exact de lignes à afficher
`android:maxHeight`	Hauteur maximale du composant
`android:maxLines`	Nombre maximum de lignes à afficher

android:text	Texte à afficher ou nom de la ressource qui contient le texte à afficher (**@string/**nom_ressource) dans le fichier /res/values/string.xml
android:textcolor	Couleur du texte (#nnn n allant de 0 à f) ou nom de la ressource qui définit la couleur (**@color**/nom_ressource) dans le fichier /res/values/colors.xml
android:textSize	Taille du texte. Privilégier l'unité sp (scaled-pixel).
android:textStyle	Style du texte (bold, italic, bolditalic) ou nom de la ressource qui contient le style (**@style**/nom_ressource) à utiliser dans le fichier /res/values/styles.xml
android:typeface	Type de typographie à utiliser (normal, monospace, serif, sans-serif…)
android:drawableTop android:drawableBottom android:drawableRight android:drawableLeft	Afficher une ressource de type drawable (image) à côté du texte selon la direction demandée (haut, bas, gauche, droite).

Les valeurs indiquant des tailles peuvent être indiquées dans le fichier /res/values/dims.xml et indiquées avec : @dimen/nom_ressource.

Exemple :

```
<TextView
    android:id="@+id/tv_bienvenue"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="46dp"
    android:paddingHorizontal="12dp"
    android:paddingVertical="16dp"
    android:text="@string/tv_bienvenue"
    android:textSize="18sp"
    android:textStyle="bold"
    android:typeface="serif"
    android:background="#1e62ea"
    android:textColor="#fff"
/>
```



/res/layout/**strings.xml**

```
<resources>
    <string name="app_name">FirstApplication</string>
    <string name="tv_bienvenue">"Bienvenue chez Scoot's Pizza"</string>
    <string name="nouveau_titre">"Scoot's Pizza"</string>
</resources>
```

Si on souhaite modifier le texte du composant depuis le code **java** :

```
TextView tvBienvenue = (TextView) findViewById(R.id.tv_bienvenue);
tvBienvenue.setText(R.string.nouveau_titre);
```

5.3. Le composant EditText

Le composant **EditText** permet à l'utilisateur de saisir du texte. Dès que l'utilisateur clique sur le composant, le clavier virtuel apparaît. EditText hérite du composant TextView et donc, de ses propriétés.

Le composant EditText propose la propriété **inputType** qui permet d'indiquer le type de texte à saisir (tél, mail, texte…).

Valeur	Description
number	Le contenu est de type nombre. Le clavier numérique est proposé quand on clique sur le champ de saisie.
phone	Le contenu est un numéro de téléphone. Le format va dépendre de la configuration du terminal.
Text	Le contenu est de type texte.
textPersonName	Le contenu est un nom de personne
textEmailAddress	Le contenu est une adresse mail. Le clavier proposé contient le caractère @
textMultiLine	Le texte est sur plusieurs lignes. On peut indiquer le nombre de lignes à l'aide de l'attribut minLines
textCapWords	Le contenu est de type texte. Chaque mot commence par une majuscule
textCapSentences	Le contenu est de type texte. Chaque phrase commence par une majuscule.

Exemples :

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/et_nom"
    android:inputType="textCapWords"/>

<EditText
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/et_mail"
    android:inputType="textEmailAddress"/>
<EditText
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/et_matricule"
    android:inputType="number"/>
```

5.4. Le composant Button

Ce widget permet de valider une action en cliquant sur le bouton (appui tactile). Le composant Button hérite de TextView également. L'action sur le bouton déclenche un événement qu'il faut intercepter pour traiter l'action. Il faut donc écouter l'événement sur le bouton, c'est ce qu'on appelle un écouteur.
Il y a plusieurs façons d'intercepter un événement.

Méthode 1 - mode déclaratif :

Depuis la version Android 1.6, il est possible d'inclure dans le fichier de layout XML, l'attribut nommé **android:onClick** dont la valeur spécifie la méthode à exécuter quand on clique sur le bouton. Cette méthode sera alors implémentée dans le fichier Activity associé au fichier de layout XML dans la méthode onCreate().

Le fichier de layout activity.xml	Le fichier d'activité correspondant activity.java

`<Button`
`  android:text="nomBouton"`
`  android:id="@+id/idBouton"`
`  android:onClick="nomMethode"`
`  …`
`/>`
	`public class <…>Activity extends Activity {`
	`  public void nomMethode(View vue) {`
	`  // Traiter le clic sur le bouton`
	`  }`
	`}`

Exemple :

Fichier components_layout.xml :

```
<Button
    …
    android:text="Valider"
    android:id="@+id/idValider"
    android:onClick="bValider"
    …/>
```

Fichier MainActivity.java :

```
public class MainActivity extends AppCompatActivity {
    TextView tvTitre;
    EditText etNom;
    Button bValider;
    ….
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.components_layout);
        …
        bValider = (Button) findViewById(R.id.idValider);
        etNom = (EditText) findViewById(R.id.idNom);
        …
    }
    public void bValider(View vue) {
        String nom = etNom.getText().toString() ;
        String message = new String ("Authentification réussie "+ " "+ nom);
        Toast.makeText(this, message, Toast.LENGTH_LONG).show();
    }
}
```

Méthode 2 : L'activité est un écouteur

L'activité est l'écouteur de l'événement sur le bouton. Pour cela, elle doit implémenter l'interface onClickListener. Dans le fichier du layout.xml, la définition du bouton ne doit pas contenir l'attribut android:onClick mais doit absolument créer l'identifiant du bouton.

Le fichier de layout main_activity.xml	Le fichier d'activité correspondant MainActivity.java

<Button
android:text="nomBouton"
android:id="@+id/idBouton"
 	
…
/>
	public class MainActivity extends AppCompatActivity
	implements View.OnClickListener
	{
	Button leBouton;
	@Override
	protected void onCreate(Bundle savedInstanceState) {
	…
	leBouton = (Button) findViewById(R.id.idBouton);
	leBouton.setOnClickListener(this);
	}
	public void onClick(View vue) {
	…
	}
	}

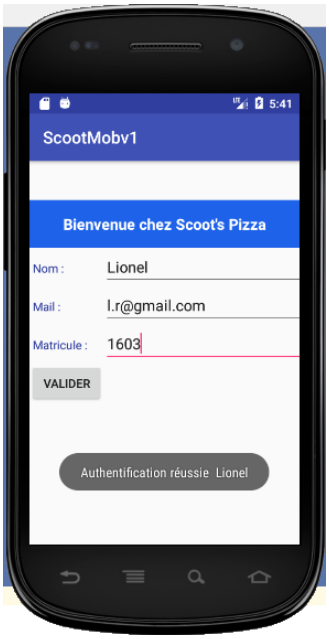
Exemple :

Fichier components_layout.xml :

```
<Button
    …
    android:text="Valider"
    android:id="@+id/idValider"
    android:onClick="bValider"
    …/>
```

Fichier MainActivity.java :

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    TextView tvTitre;
    EditText etNom;
    Button bValider;
    …
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.components_layout);
        …
        etNom = (EditText) findViewById(R.id.idNom);

        bValider = (Button) findViewById(R.id.idValider);
        bValider.setOnClickListener(this);

    }

    public void onClick(View vue) {
      String nom = etNom.getText().toString() ;
      String message ;

      message = new String ("Authentification réussie " + nom);
      Toast.makeText(this, message, Toast.LENGTH_LONG).show();
    }
}
```



Méthode 3 : l'écouteur est une classe anonyme

En Java, une classe anonyme est une classe définie sans nom, par dérivation d'une super classe ou par implémentation d'une interface. Cette méthode de traitement apparaît dans de nombreux exemples de programmes Android Java/Swing et sera nécessaire pour l'exploitation de certains composants graphiques.

La définition du bouton dans le fichier de layout est identique à la méthode 2. Seul le code de l'activité java diffère.

Fichier <…>Activity.java :

```
public class <…>Activity extends AppCompatActivity {
    Button lebouton;

    protected void onCreate(Bundle savedInstanceState) {

        …
        lebouton = (Button) findViewById(R.id.idbouton);

        View.OnClickListener ecouteur = new View.OnClickListener() {
                                            @Override
                                            public void onClick(View v) {
                                                …
                                            }
                                        }
        ) ;

        leBouton.setOnClickListener(ecouteur);
    }
}
```

Autre version :

```
public class <…>Activity extends AppCompatActivity {
    Button lebouton;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.<…>_layout);

        …
        lebouton = (Button) findViewById(R.id.idbouton);

        View.OnClickListener ecouteur =

      leBouton.setOnClickListener(new View.OnClickListener() {
                                    @Override
                                    public void onClick(View v) {
                                        …
                                    }
                                }
                               ) ;
    }
}
```

6. Les styles

Comme pour les feuilles de styles CSS, le framework Android permet de créer ses propres styles qu'on regroupe dans le fichier /res/values/styles.xml.

Pour créer un style, on lui spécifie un nom et on liste les différents attributs du style sous forme d'items.

Exemple : styles.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<style name="MyTitleStyle">
    <item name="android:textColor">#000</item>
    <item name="android:textSize">24sp</item>
    <item name="android:background">#fff</item>
    <item name="android:layout_gravity">center_horizontal</item>
</style>
</resources>
```

Les styles supportent la notion d'héritage. L'attribut parent de la balise `<style>` permet d'indiquer le style dont il va hériter. Toutes les propriétés définies dans le style parent seront reprises par le style enfant et celles qui sont redéfinies se substitueront aux propriétés parentes.

Pour appliquer un style, on utilise l'attribut `android:theme` :

```
<android:theme="@style/nomStyle">
```

Un style peut être appliqué pour toute l'application (définition du style dans la balise <application>), pour une activité spécifique (balise <activity>) ou pour un composant (balise du composant) dans le fichier `AndroidManifest.xml`.

L'utilisation de styles prédéfinis permet de modifier rapidement l'interface graphique d'une application.

TP1. Application GSB-RV Visiteurs –

Activité : Implémentation de l'activité « afficher un rapport de visite »

Cette activité représente l'étape 5 de l'UC « Consulter un compte-rendu ». Elle consiste à afficher l'ensemble des informations relatives à un rapport de visite. Cette vue sera appelée vue3.

Phase 1 : Construire l'interface graphique de l'activité

1. Implémenter le fichier de layout afficher_rv_layout.xml
2. Implémenter l'activité AfficherRvActivity.java correspondant.
3. Créer un objet rapportVisite initialisé avec un rapport de visite existant.
4. Remplir la vue3 avec l'objet créé.

Phase 2 : Afficher un rapport de visite sous Symfony

1. Implémenter la route : /visiteurs/<matricule>/rapport/<numero_rapport>
2. Ecrire le code de la méthode getUnRapportVisite() qui permet de récupérer, depuis la base de données, le rapport de visite référencé par numero_rapport du visiteur référencé par matricule.
3. Afficher les informations du rapport dans un navigateur.

Phase 3 : Récupérer un rapport de visite depuis le serveur Rest (Symfony)

1. Dans le fichier AfficherRvActivity, initialiser cette fois-ci l'objet rapportVisite avec les données retournées par le serveur Rest.
2. Remplir de nouveau la vue3 avec l'objet rapportVisite.

Remarque : Cette dernière phase est à réaliser après le cours sur le client REST.