

# Chapitre 3 : Applications Android : Composants et Interfaces

## Introduction

Une application Android est composée de plusieurs éléments qu'il faut assembler pour obtenir un tout cohérent. Elle admet une ou plusieurs interfaces graphiques organisées en vues et gabarits, avec néanmoins quelques spécificités.

Dans ce chapitre, nous allons commencer par exposer les différents composants d'une application Android, ensuite nous présentons le concept d'interface en détaillant quelques propriétés et classes de base.

### I. Composants d'une application

Cette partie traite les composants suivants : activité, service, fournisseur de contenu, intentions, récepteurs de diffusion et notifications.

#### I.1. Activité

Une activité Android représente en gros ce que l'on voit à l'écran, elle est composée :

- D'une classe JAVA héritant de la classe `AppCompatActivity` (elle-même hérite de la classe `Activity`) et implémentant les méthodes de gestion du cycle de vie de l'activité.
- D'une interface utilisateur, qui pourra être définie soit dans le code de l'activité soit de façon plus générale dans un fichier XML placé dans les ressources de l'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);
    }
}
```

Un projet Android est composé obligatoirement d'au moins une activité. Elle représente l'implémentation métier dans une application et permet de gérer l'ensemble des vues et des ressources.

Il est possible d'avoir plusieurs activités dans le même programme. Une activité doit être toujours déclarée dans le fichier de configuration « `AndroidManifest.xml` ».

Dès sa création, une activité passe par plusieurs états comme le montre la figure suivante :

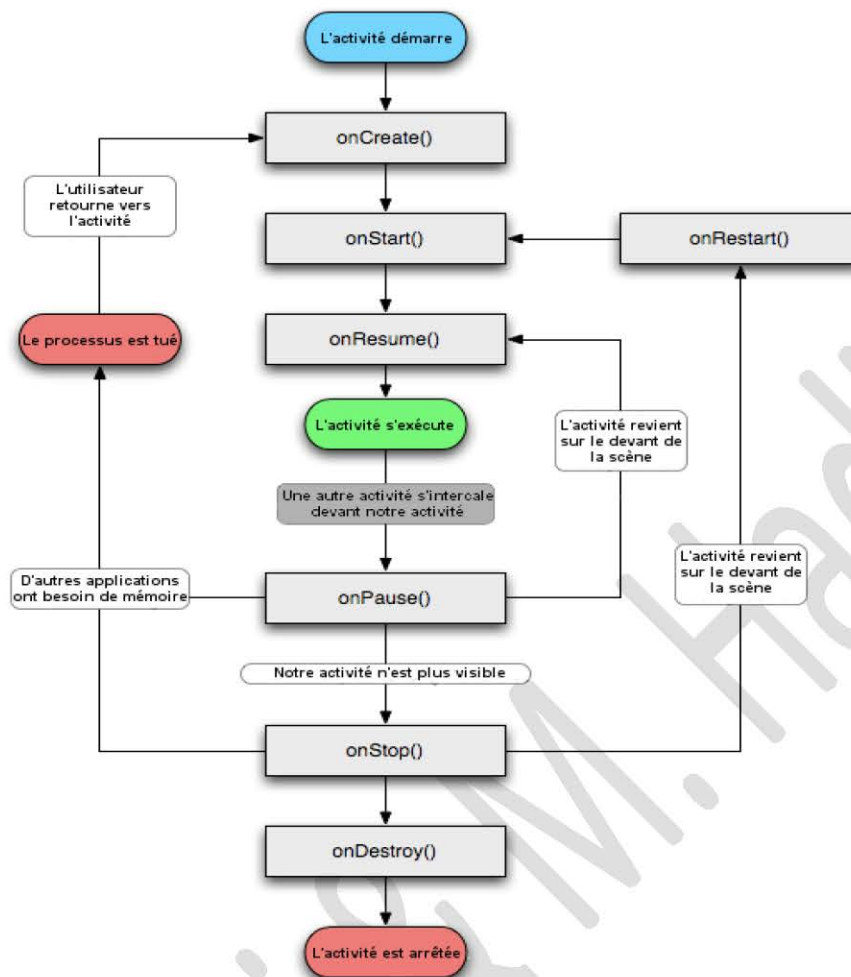


Figure 1: Cycle de vie d'une activité Android

Le cycle de vie d'une activité est parsemé d'appels aux méthodes relatives à chaque étape de sa vie. Il informe ainsi le développeur sur la suite des événements et le travail qu'il doit accomplir.

Ainsi, les différentes méthodes sont présentées ci-dessous :

- **onCreate()** : Cette méthode est appelée à la création d'une activité. Elle permet de l'initialiser. C'est ici que l'interface graphique est spécifiée avec la méthode « setContentView() »
- **onStart()** : Cette méthode est appelée quand l'application est démarrée.
- **onResume()** : Cette méthode est appelée quand l'application passe (ou repasse) en avant-plan.
- **onPause()** : Appelée quand l'application passe en arrière-plan et qu'une autre application se met devant.
- **onStop()** : Appelée quand l'application n'est plus visible.
- **onRestart()** : Appelée quand l'application redevient visible.
- **onDestroy()** : Appelée quand l'application est fermée par le système à cause d'un manque de ressources, ou par l'utilisateur à l'utilisation d'un *finish()*.

#### Remarque :

- Avant de s'arrêter, le système appelle la méthode **onSaveInstanceState()** pour que l'activité puisse enregistrer les informations d'état.
- Lorsque l'activité est recréée, il est possible de restaurer l'état de l'instance enregistrée avec **onSaveInstanceState()** avec la méthode **onRestoreInstanceState()**.

### I.2. Service

Un service est un programme lancé en arrière-plan, possède un cycle de vie similaire à une activité et n'a pas une interface utilisateur. Il sert à effectuer des traitements qui peuvent prendre du temps : chargement d'un fichier, lecture de la musique, ...

Il existe deux types de services :

- **LocalService** : s'exécute dans le même processus de l'application
- **RemoteService** : s'exécute dans des processus indépendants de l'application

### I.3. Fournisseur de contenu

Un fournisseur de contenu ou « Content Provider » sert à stocker et récupérer des données et les rendre accessibles à toutes les applications. C'est le moyen le plus connu de partager des données entre différentes applications. Par exemple, il existe un Content Provider gérant les Contacts d'un téléphone.

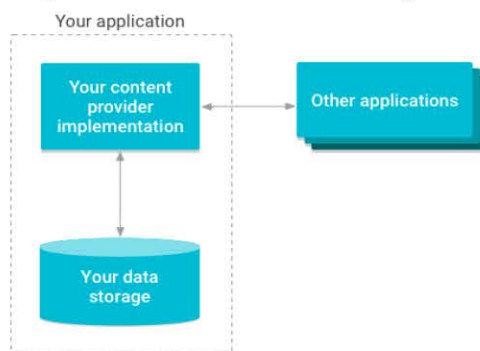


Figure 2: Fonctionnement du Content Provider

Android propose plusieurs Content Providers basiques (audio, vidéo, images, informations sur les contacts...). Un content Provider se compose d'une URI<sup>1</sup> et d'un ensemble de méthodes (Insert, Update, Delete, Query).

### I.4. Intentions

La communication interne d'Android est basée sur l'envoi et la réception de messages exprimant une opération à effectuer. Les messages peuvent être émis à destination d'un autre composant de la même application (une activité, un service...) ou vers n'importe quelle autre application. La classe `Intent` modélise un tel message.

### I.5. Récepteurs de diffusion

Pour pouvoir recevoir des intents, Android permet de créer une classe qui implémente les récepteurs de diffusion ou encore `BroadcastReceiver`. Ces objets sont conçus pour recevoir des intentions et capter des messages diffusés au niveau du système (batterie faible, l'écran est éteint,...).

### I.6. Notifications

Une notification signale une information à l'utilisateur sans interrompre ses actions en cours. Elle permet d'attirer l'attention de l'utilisateur à partir d'un service ou d'un récepteur de diffusion en affichant une alerte. Cette alerte peut être sous forme d'une icône ou d'un message accompagné d'un bip sonore.

<sup>1</sup> URI : Uniform Resource Identifier



## II. Le concept d'interface

L'interface graphique d'une application Android peut se faire de deux façons : soit par la description de l'interface dans des fichiers XML, soit par la programmation Java. Les fichiers XML qui décrivent une interface sont placés dans le répertoire « **res/layout** ». Ils sont référencés par « **R.layout.nom\_du\_fichierXML** ».



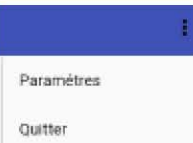
Les activités peuvent utiliser la méthode « **setContentView(R.layout.nom\_du\_fichierXML)** » pour mettre en place l'interface décrite par un tel fichier.

La classe **R** est une classe générée et mise à jour automatiquement par Android. Elle se trouve dans un sous dossier de *app\build\generated* (vue project d'Android Studio). Le fichier « *R.java* » définit une classe **R** dans laquelle sont définis les identifiants des ressources de l'application. A chaque fois qu'on rajoute une ressource à l'application, un identifiant unique est généré automatiquement dans cette classe permettant par la suite de pouvoir le référencer afin l'utiliser dans le code. A l'intérieur de la classe **R** sont définies plusieurs classes internes statiques, telles que **string**, **mipmap**, **layout**, **menu**, **id**, ....

## III. Les ressources

Les ressources Android sont des fichiers qui peuvent être utilisés dans une application mobile telles que : les images, les chaînes de caractères, les menus, les styles, les couleurs, ...

L'avantage majeur de l'utilisation des ressources est de permettre d'afficher une application Android de la même manière sur un très grand nombre de supports différents : Smartphones, Tablettes, SmartTV, ... Ces ressources se trouvent dans le dossier "app/res" et sont utilisées dans le code Java à travers la classe statique **R**.

Type ressources	Description	Exemple de code	Aperçu
Dessin et image (res/drawable)	Ce dossier contient les images de types PNG, JPEG ou encore GIF. Le nom de l'image doit être en minuscule	Dans le fichier activity_main.xml <pre>&lt;ImageView     android:layout_width="match_parent"     android:layout_height="wrap_content"     android:src="@drawable/precedent" /&gt;</pre>	
Icon (res/mipmap)	Ce dossier contient l'icône de l'application	Dans le fichier AndroidManifest.xml <pre>&lt;application     android:icon="@mipmap/ic_launcher"     android:label="@string/app_name"&gt; &lt;/application&gt;</pre>	
Interface graphique (res/layout)	Les fichiers XML qui représentent la disposition des vues	Dans le fichier MainActivity.java <pre>setContentView(R.layout.activity_main);</pre>	
Menu (res/menu)	Le fichier XML qui définit un menu.	Dans le fichier main.xml <pre>&lt;menu&gt; &lt;item     android:id="@+id/param"     android:title="Paramètres" /&gt; &lt;item     android:id="@+id/quitte"     android:title="Quitter" /&gt; &lt;/menu&gt;</pre>	

res/values/strings.xml	Le fichier XML qui définit les constantes chaînes de caractères	<p>Dans le fichier strings.xml</p> <pre>&lt;resources&gt; &lt;string name="combien"&gt;Combien ?&lt;/string&gt; &lt;/resources&gt;</pre> <p>Dans le fichier activity_main.xml</p> <pre>&lt;Button     android:id="@+id/button"     android:text="@string/combien" /&gt;</pre>	Combien ?
res/values/colors.xml	Le fichier XML qui définit les couleurs	<p>Dans le fichier colors.xml</p> <pre>&lt;resources&gt; &lt;color name="gray"&gt;#D3D3D3&lt;/color&gt; &lt;/resources&gt;</pre> <p>Dans le fichier activity_main.xml</p> <pre>&lt;EditText     android:background="@color/gray"     android:text="Hello" /&gt;</pre>	Hello

## IV. Classes de base des interfaces

### IV.1. La classe View

La classe `View` n'est pas utilisée directement mais constitue la classe mère des composants graphiques. Elle est décrite ici pour éviter de répéter ses propriétés dans les autres classes d'interface (`TextView`, `EditText`, `Button`,...)

`View` est la classe dont héritent toutes les classes utilisées pour réaliser des interfaces. Ses propriétés et ses méthodes se retrouvent donc dans tous les éléments de l'interface.

#### 1.1. Identifiant

<b>En XML</b>	Pour créer en XML un identifiant nommé <code>mon_id</code> , on utilise la notation suivante : <pre>android:id="@+id/mon_id"</pre>
<b>En JAVA</b>	Cet identifiant peut être référencé en Java par : <pre>findViewById(R.id.mon_id)</pre>

#### 1.2. Visibilité

<b>En XML</b>	<pre>android:visibility="valeur"</pre> <p>Les valeurs possibles pour <code>valeur</code> sont : <code>visible</code>, <code>invisible</code> (la place est conservée) et <code>gone</code> (la place n'est pas conservée).</p>
<b>En JAVA</b>	<pre>vue.setVisibility(View.valeur);</pre> <p><code>valeur</code> : <code>VISIBLE</code>, <code>INVISIBLE</code> ou <code>GONE</code></p>

#### 1.3. Fond

<b>En XML</b>	<pre>android:background="valeur"</pre> <p>Cette instruction permet de définir une couleur ou bien une image d'arrière-plan :</p> <pre>android:background="#0000FF" android:background="@drawable/monimage"</pre>
---------------	--



## Cours : Développement Mobile

	L'image est placée dans <code>app/res/drawable/</code> , s'appelle <code>monimage</code> et peut être de type <code>png</code> , <code>jpg</code> , ou <code>gif</code> .  <b>ATTENTION :</b> Les noms des images ne doivent utiliser que des minuscules ou des chiffres.
En JAVA	<pre>vue.setBackgroundColor(Color.BLUE); vue.setBackgroundResource(R.drawable.monimage);</pre>

### 1.4. Taille

Pour le développement d'application Android, il existe 6 unités de mesure :

- `px` (pixel) : il est déconseillé de l'utiliser vu que la taille d'un pixel varie d'un terminal mobile à un autre.
- `mm` (millimètre) : elle est basée sur la taille physique de l'écran.
- `in` (inch ou pouce) :  $1\text{in} \approx 2,54\text{ cm}$ .
- `pt` (point) :  $1\text{pt} = 1/72\text{ pouce}$ .
- `dp` (Density independent Pixel ou encore Densité de pixels indépendant) :  $1\text{dp} = 160\text{ dpi}$  (dots per inch). Cette unité est utilisée pour la mise en page des éléments.
- `sp` (Scale independent Pixel ou encore Echelle de pixels indépendant) : Utilisée pour les tailles de polices. Généralement,  $1\text{sp} = 1\text{dp}$  mais dans certaines situations  $1\text{sp}$  peut valoir plus comme pour le cas des utilisateurs malvoyants (option d'accessibilité activée).

En XML	Les propriétés XML de la taille de la classe View sont : <pre>android:minwidth="unité" android:minheight="unité"</pre>
En JAVA	<pre>vue.setMinimumHeight(unité); vue.setMinimumWidth(unité);</pre>

### Remarque :

La valeur de l'unité en XML peut être : `px`, `dp`, `sp`, `in`, `mm` ou `pt` tandis que en JAVA elle ne peut être qu'en `px`.

### 1.5. Gravité

En XML	La propriété XML permettant de définir comment se placent les éléments contenus par ce conteneur est : <pre>android:gravity="valeur"</pre> <p>Où <code>valeur</code> peut prendre les valeurs : <code>top</code>, <code>bottom</code>, <code>left</code>, <code>right</code>, <code>center_vertical</code>, <code>fill_vertical</code>, <code>center_horizontal</code>, <code>fill_horizontal</code>, <code>center</code>, <code>fill</code>.</p>
En JAVA	<pre>btnAfficher.setGravity(valeur);</pre> <p>Où <code>valeur</code> peut prendre les valeurs : <code>Gravity.TOP</code>, <code>Gravity.BOTTOM</code>, <code>Gravity.LEFT</code>, <code>Gravity.RIGHT</code>, <code>Gravity.CENTER_VERTICAL</code>, <code>Gravity.FILL_VERTICAL</code>, <code>Gravity.CENTER_HORIZONTAL</code>, <code>Gravity.FILL_HORIZONTAL</code>, <code>Gravity.CENTER</code>, <code>Gravity.FILL</code></p>

### 1.6. Marges internes

En XML	Les propriétés XML des marges internes de la classe View sont : <pre>android:paddingBottom="unité" android:paddingLeft="unité"</pre>
--------	---

	<pre>android:paddingRight="unité" android:paddingTop="unité"</pre>
<b>En JAVA</b>	<pre><b>vue</b>.setPadding(marge_gauche, marge_haut, marge_droite, marge_bas);</pre>

## IV.2. La classe ViewGroup

Un écran Android est généralement composé de plusieurs *vues* : *TextView*, *ImageView*, *EditText*, *Button*, *CheckBox*,...

### 2.1. Taille

<b>En XML</b>	<pre>android:layout_height="unité" android:layout_width="unité"</pre>
<b>En JAVA</b>	<pre><b>vueGroupe</b>.setMinimumHeight(<b>unité</b>); <b>vueGroupe</b>.setMinimumWidth(<b>unité</b>);</pre>

### 2.2. Marges externes

<b>En XML</b>	<pre>android:layout_marginBottom="unité" android:layout_marginLeft="unité" android:layout_marginRight="unité" android:layout_marginTop="unité"</pre>
<b>En JAVA</b>	<pre><b>vueGroupe</b>.setPadding(marge_gauche, marge_haut, marge_droit, marge_bas);</pre>

## Conclusion

Dans ce chapitre, nous avons introduit au début les composants d'une application Android ainsi que ses ressources et présenté également le concept d'interface en détaillant ses classes de base.

Le chapitre suivant permet d'introduire les différents types de conteneurs utilisés dans une application Android.