



Chapitre III:

Activités et ressources



Introduction aux activités et aux ressources

Dans le contexte du développement d'applications mobiles, les activités et les ressources sont des concepts essentiels qui constituent le cœur de l'interface utilisateur et des fonctionnalités de l'application.

Les activités représentent les différents écrans ou fenêtres de l'application, tandis que les ressources sont les éléments visuels et audio qui peuplent l'interface de l'application.

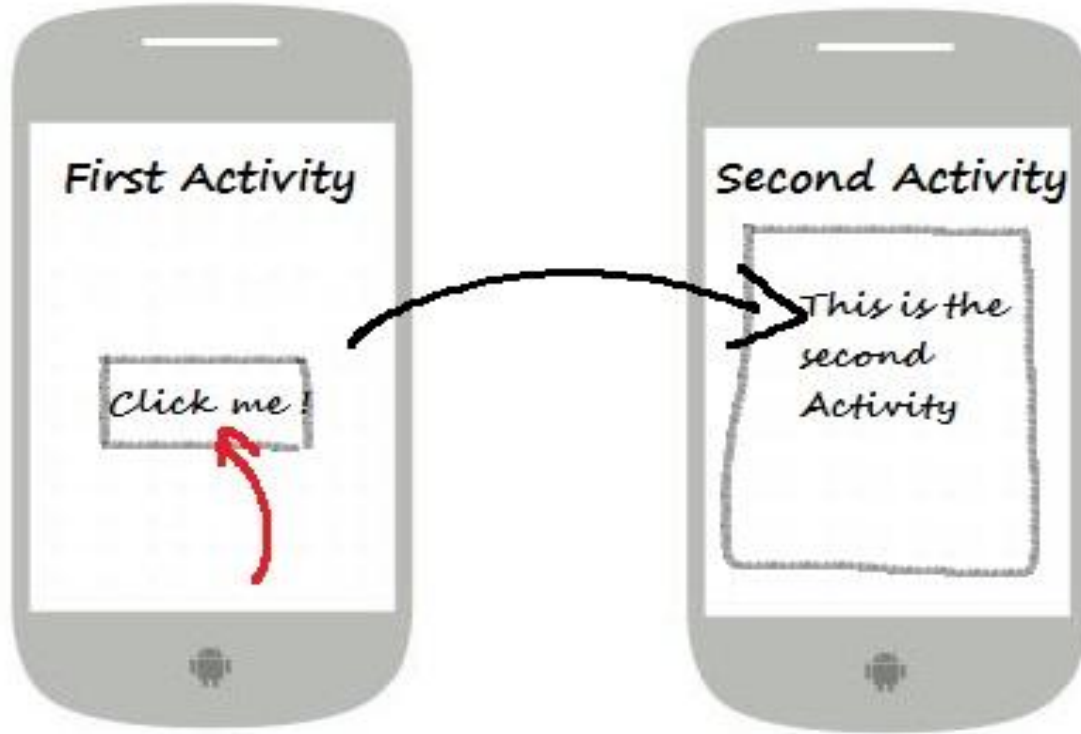
Définition de l'activité

- Dans Android, une activité est un composant d'une application qui représente un seul écran avec une interface utilisateur.
- Elle peut contenir des éléments UI tels que des boutons, des champs de texte, des images et d'autres widgets interactifs.
- Les activités sont conçues pour gérer les interactions des utilisateurs, comme les clics sur les boutons, et pour répondre aux saisies des utilisateurs.
- Elles sont une partie importante de l'architecture des applications Android et offrent aux développeurs un moyen de créer des expériences utilisateur dynamiques et interactives.

Activité : définition

- Une activité est le composant fondamental d'une application Android qui représente une **interface utilisateur**.
- Elle permet à l'utilisateur d'interagir avec l'application via une **interface spécifique**.
- Une application comportant plusieurs écrans possédera plus d'activités.
- Chaque activité est dédiée à un objectif précis, tel que :
 - Afficher des données (ex. : liste de contacts)
 - Modifier des informations (ex. : profil utilisateur)
 - Exécuter une action (ex. : lire une vidéo)

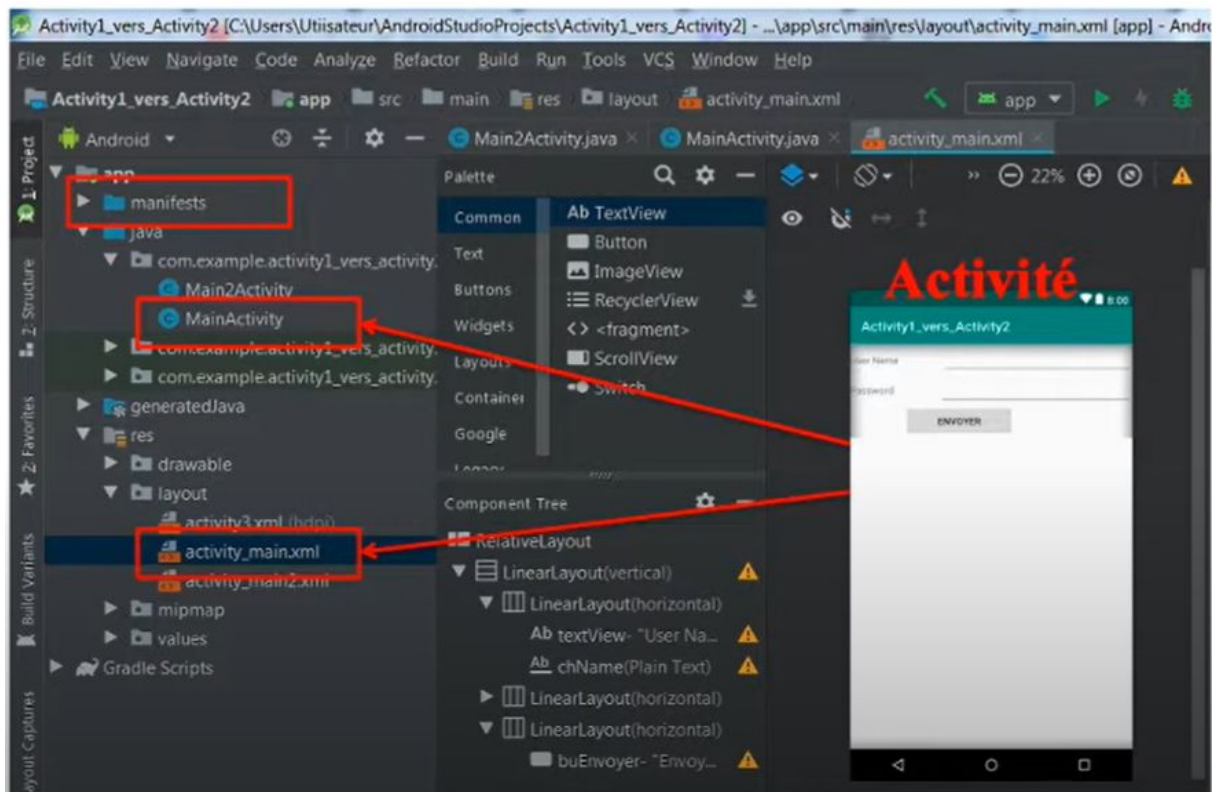
Activité : Exemple



Activité : Fichiers de base (1/2)

Chaque **activité** de votre application est liée à 3 principaux fichiers :

- Le fichier XML (**Manifest**) où elle est déclarée.
- Un **fichier xml** dans le répertoire /res/layout/ qui décrit son design
- Un **fichier Java** pour la contrôler.



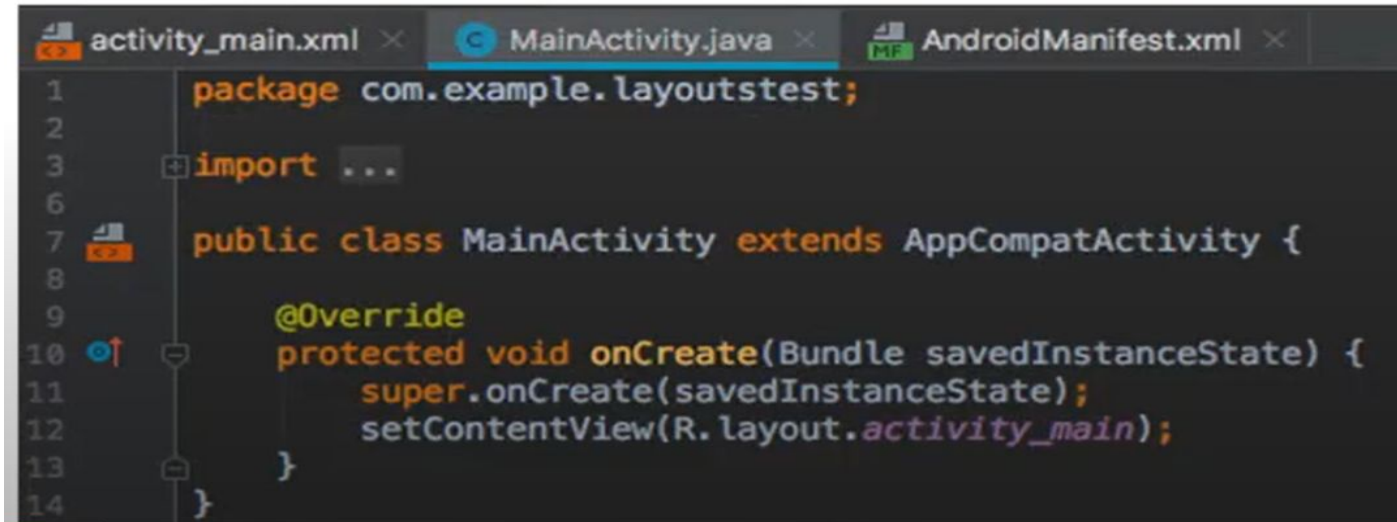
Activité : Fichiers de base (2/2)

- Pour déclarer une activité dans le fichier Manifest XML, on utilise la balise **<activity>**.
- La balise **<intent-filter>** indique la première activité qui se lance au démarrage de l'application.

```
<activity
  android:name=".MyActivity"
  android:label="@string/app_name"
  | <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

Activité : définition


- Les activités des applications Android sont implémentées en Java par des classes d'activités : ce sont des sous-classes de **android.app.Activity**.



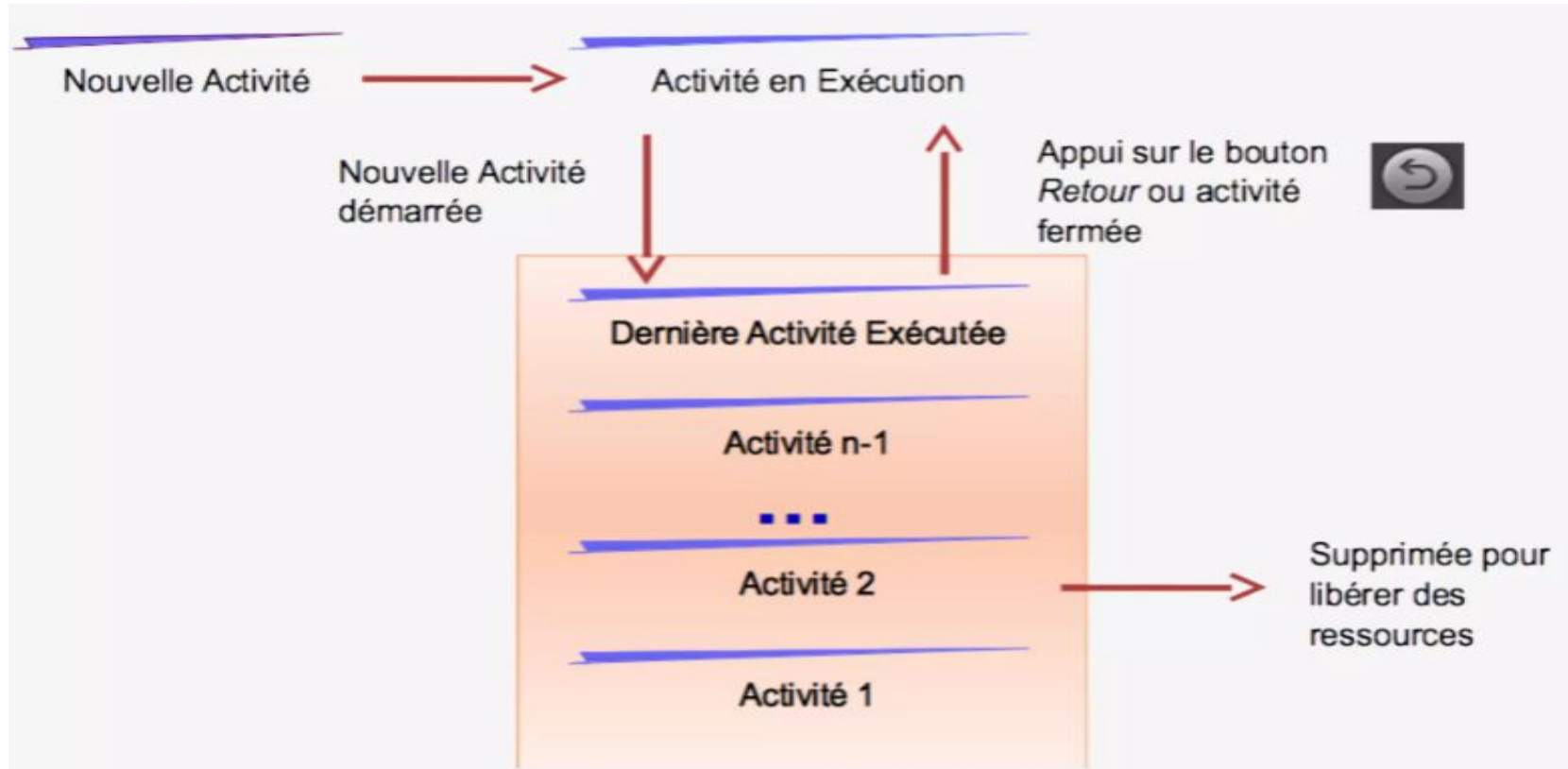
The screenshot shows the MainActivity.java file in an Android Studio editor. The code defines a package, imports necessary classes, and implements the MainActivity class which extends AppCompatActivity. The onCreate method is overridden to call super.onCreate and setContentView.

```
1 package com.example.layoutstest;
2
3 import ...
4
5
6
7 public class MainActivity extends AppCompatActivity {
8
9     @Override
10    protected void onCreate(Bundle savedInstanceState) {
11        super.onCreate(savedInstanceState);
12        setContentView(R.layout.activity_main);
13    }
14 }
```


Pile d'activité

- Les activités dans une application sont gérées par **une pile d'activités**.
- Quand une nouvelle activité se lance, elle se met tout en haut de la **pile** et devient **l'activité en exécution**.
- L'activité précédente reste dessous de la pile.
- Elle ne revient au premier que si la nouvelle activité est fermée.
- Si l'utilisateur clique sur le bouton Retour,  l'activité suivante dans la pile devient active .

Pile d'activité

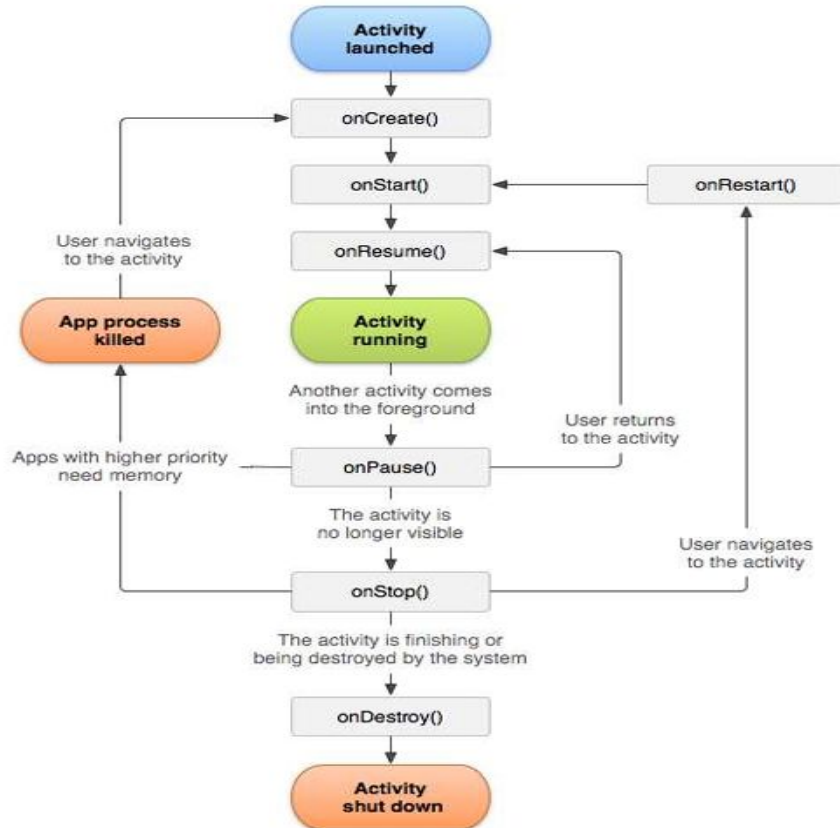


Activité : État d'activité

- Une activité peut se trouver dans trois états qui se différencient surtout par leur visibilité :
 - **Active** : l'activité est lancée par l'utilisateur, elle s'exécute en premier plan ;
 - **En pause** : l'activité est en pause par l'utilisateur, elle s'exécute et est partiellement visible à l'écran, mais elle n'est plus en premier plan. Une notification ou une autre activité lui a volé le focus.
 - **Stoppé** : l'activité a été lancée par l'utilisateur, mais n'est plus au premier plan et est invisible.
 - **Morte** : l'activité n'est pas lancée.

En général, le passage entre les états de cycle de vie d'une activité est assuré par les méthodes de **callback** suivantes :

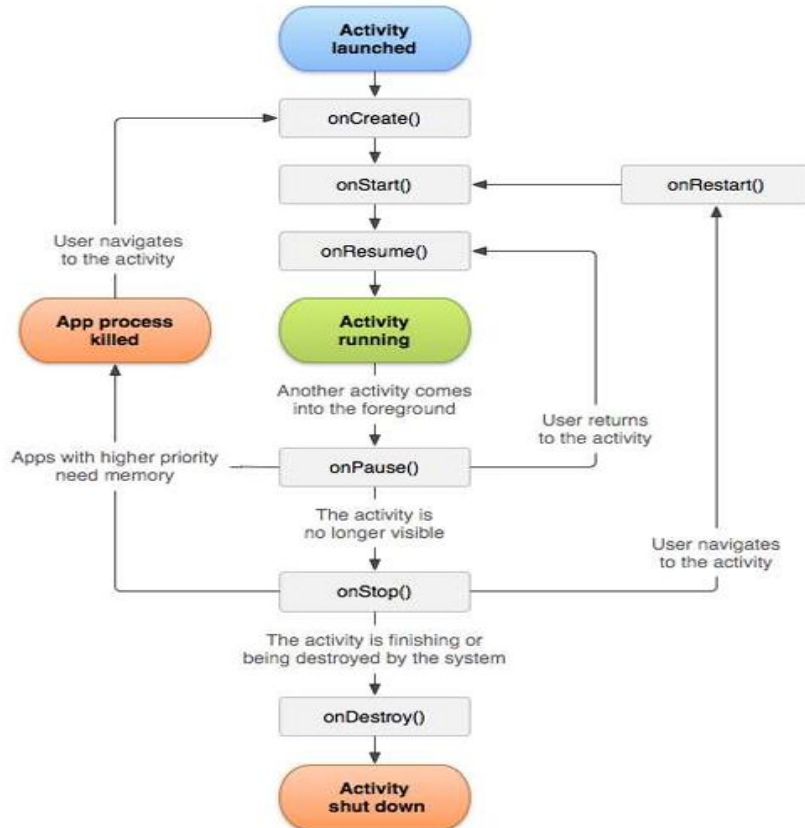
Activité : Cycle de Vie



OnCreate () :

- Méthode exécutée quand l'**activité est créée**.
- Si l'activité est la première d'une application, cette méthode est donc appelée quand l'utilisateur exécute l'application.

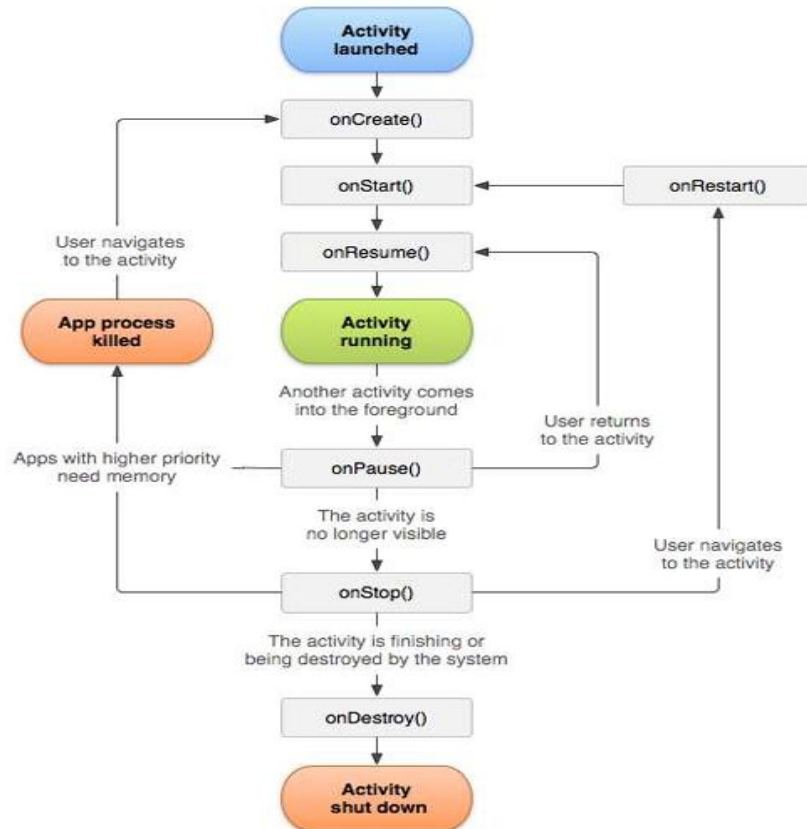
Activité : cycle de vie



onStart () :

- Méthode exécutée après **`onCreate()`** ou **`onRestart()`**.
- À cette étape, **l'activité devient visible** par l'utilisateur.

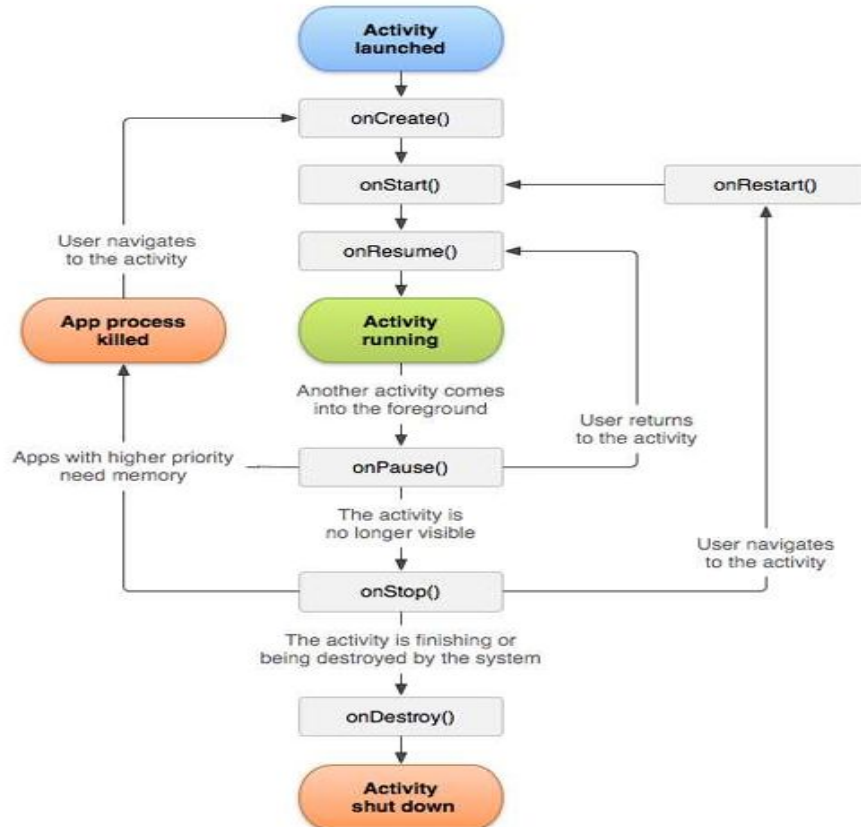
Activité : cycle de vie



OnResume () :

- Appelée juste *avant* que l'activité ne commence à interagir avec l'utilisateur.
- À cette étape, l'activité est *en haut* de la pile.
- Toujours suivie de *onPause()*.

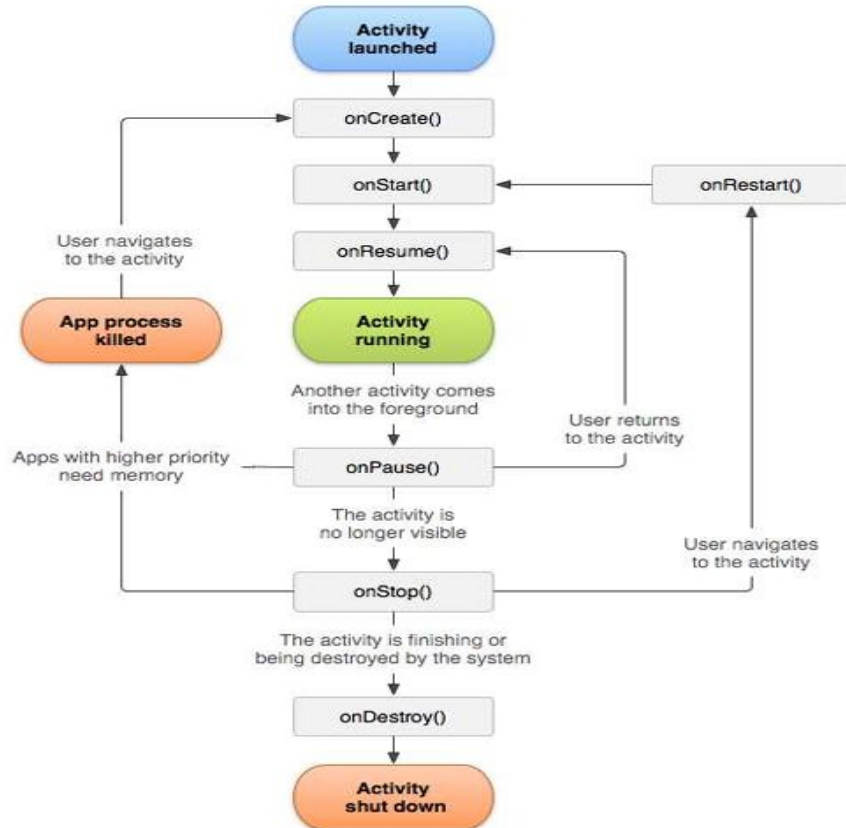
Activité : cycle de vie



OnPause () :

- Méthode exécutée à chaque fois que :
 - L'utilisateur passe **à une autre activité**.
 - L'utilisateur **demande un finish()** sur cette activité.
 - Le système a besoin **de libérer de la mémoire**.
- La méthode est **exécutée systématiquement** avant la méthode **`onResume()`** et **`onStop()`**.

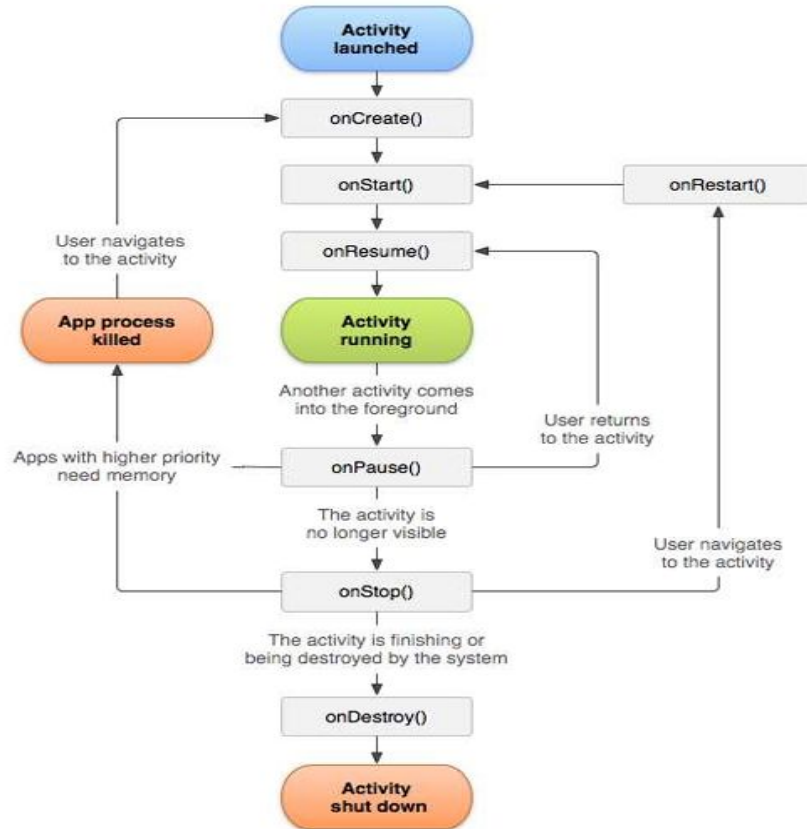
Activité : cycle de vie



OnStop() :

- Appelée quand l'activité **n'est plus visible** par l'utilisateur.
- Peut arriver si :
 - L'activité est détruite.
 - Une autre activité a repris son exécution.
- Suivie par ***onRestart()*** ou ***onDestroy()*** si l'activité va **disparaître**.

Activité : cycle de vie



OnDestroy() :

- Appelée quand l'activité est **détruite**.
- Dernier appel que l'activité va recevoir.
- Peut intervenir si :
 - L'activité **se termine** (appel de `finish()`).
 - Le système **détruit temporairement** cette instance de l'activité pour gagner de l'espace.
- On distingue entre les deux scénarios par ***isFinishing()***.

Motivation: Les ressources

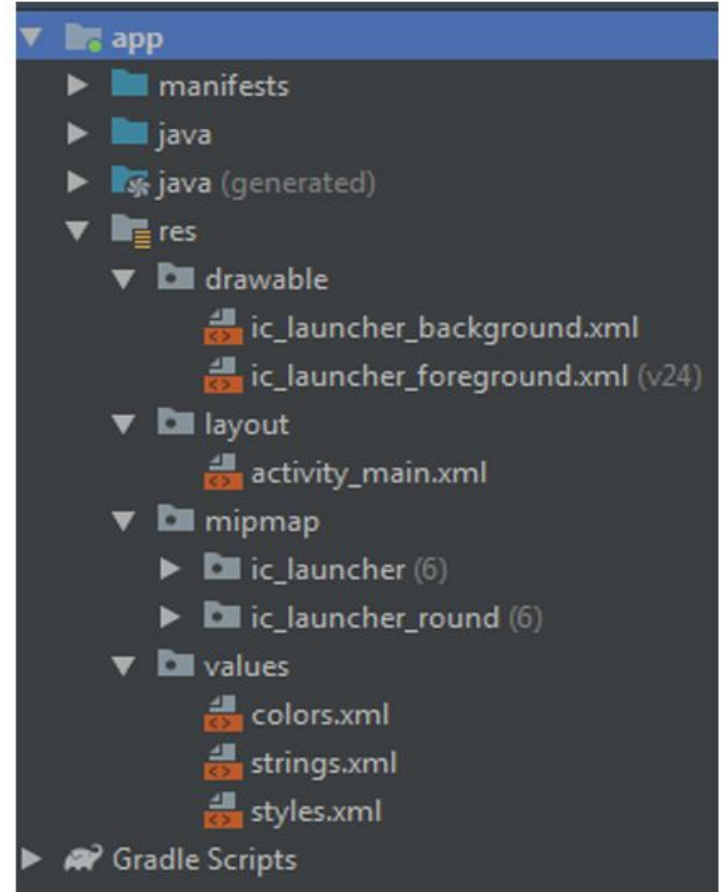
- Un fichier contenant des informations statiques pour l'application.
- Stocké en dehors du code Java, ne change pas pendant l'exécution.
- Permet à l'application de s'adapter aux différents écrans, densités et configurations matérielles.
- Adapte facilement l'application sans changer le code.
- **Exemple d'adaptation** : Préparer plusieurs versions d'images pour différentes résolutions d'écran.

Les ressources

- Les ressources sont des fichiers structurés d'une sorte qu'Android ait la possibilité de choisir une ressource, parmi d'autres, qui va être utilisée pour s'adapter au matériel sur lequel s'exécute l'application.
- L'objectif est de faciliter l'exécution des applications pour toutes les situations différentes.
- Le seul moyen qui permet aux ressources d'être adaptées aux différents types de terminaux est l'utilisation du langage de description **XML**.

Les ressources

- Android, installé sur une variété de terminaux, requiert une organisation structurée des ressources pour offrir une expérience d'affichage cohérente.
- Les ressources sont organisées dans des répertoires spécifiques au sein d'un dossier principal nommé **res**.
- Chaque type de ressource est associé à un répertoire distinct, facilitant la gestion et l'adaptation selon les terminaux.



La classe R

- Les ressources d'une application Android sont accessibles via la classe statique **R**. Elle est générée automatiquement par Android lors de la compilation de l'application.
- Elle référence chaque ressource (images, layouts, chaînes de texte, etc.) dans le code.
- Grâce à **R**, les ressources peuvent être facilement appelées et manipulées depuis le code, sans avoir besoin de chemin d'accès spécifique.

La classe R

```
public final class R {  
    public static final class string {  
        public static final int bienvenue=0x7f040000;  
        public static final int texte_bouton_quitter=0x7f040001;  
        public static final int texte_titre_ecran=0x7f040002;  
    };  
    public static final class layout {  
        public static final int ecran_de_demarrage=0x7f030001;  
        public static final int ecran_principal=0x7f030000;  
    };  
    public static final class drawable {  
        public static final int image_android=0x7f020000;  
    };  
};
```

Structure des Répertoires de Ressources

Dessin et Image (res/drawable) :

- Contient des images aux formats PNG, JPEG, GIF, et des fichiers XML pour les dessins vectoriels (scalables sans perte de qualité).
- Pour créer une ressource drawable:
 - Copier l'image dans le dossier *drawable*.
- Pour référencer une ressource drawable:
 - Utiliser la syntaxe « *@drawable/nom de l'image* »
- **Exp:** *android:background="@drawable/My_Image"*
- Pour référencer une ressource drawable dans un
- **Programme Java :** *R.drawable.nom_de_l'image*

Structure des Répertoires de Ressources

Icones de lancement (res/mipmap)

- Stocke les icônes d'application de différentes densités pour assurer un rendu correct sur tous les écrans.
- Pour créer une ressource Mipmap, copier l'icône dans le dossier *mipmap*.
- Pour référencer une ressource Mipmap :
 - Utiliser la syntaxe « *@mipmap/nom de l'image* »
- **Exp:** *android:icon="@mipmap/My_Icon"*
- Pour référencer une ressource mipmap dans un
- **Programme Java :** *R.mipmap.nom_de_l'icone;*

Structure des Répertoires de Ressources

Variables diverses (**res/values**)

- Regroupe des ressources comme :
 - Chaînes de caractères (**strings.xml**) pour gérer le texte de l'application,
 - Dimensions (**dimens.xml**) pour les marges, espacements et tailles,
 - Couleurs (**colors.xml**) pour définir les palettes de couleurs,
 - Styles (**styles.xml**) pour uniformiser l'apparence visuelle de l'interface.
- Simplifie la personnalisation et l'adaptation aux différents écrans et langues.

Structure des Répertoires de Ressources

Création de la ressource COLOR :

res/values/color.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<color name="Green"> #51B000 </color>
<color name="Blue"> #303F9F </color>
<color name="Pink"> #FF4081 </color>
</resources>
```

Structure des Répertoires de Ressources

Utilisation de la ressource COLOR :

```
<Button  
    android:id="@+id/BT2"  
    android:text="Button2"  
    android:textColor="@color/Pink"  
    android:background="@color/Green"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"/>
```



Structure des Répertoires de Ressources

Création de la ressource Dimension :

res/values/**Dimens.xml**

```
<resources>
    <dimen name="marge_intern">16dp</dimen>
    <dimen name="marge_extern">16dp</dimen>
    <dimen name="large_button">120dp</dimen>
    <dimen name="haut_button">60dp</dimen>
</resources>
```

Structure des Répertoires de Ressources

Utilisation de la ressource dimensions :

<Button

android:id="@+id/BT2"

android:text="Button2"

android:textColor="@color/Pink"

android:background="@color/Green"

android:layout_width="@dimen/large_button"

android:layout_height="@dimen/haut_button"

android:layout_margin="@dimen/marge_extern"

android:Padding="@dimen/marge_intern"/>



Structure des Répertoires de Ressources

Création de la ressource String :

res/values/**Strings.xml**

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<resources>
```

```
< string name="App_Name"> CourseAndroid</string>
```

```
< string name="Msg"> Welcome </string>
```

```
< string name="Qst"> Give your Question </string>
```

```
</resources>
```

Structure des Répertoires de Ressources

Utilisation de la ressource String :

- Affecter une ressource String à une vue (View) en Java :
 - *`EditText txt = (EditText) this.findViewById(R.id.t);`*
 - *`txt.setText(R.string.Qst);`*
- Pour spécifier un titre de label dans un layout.xml :
 - *`@string/nom`*

```
<TextView
    android:text="@string/ Qst ."
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/textView" />
```

Structure des Répertoires de Ressources

Ressource style:

- Les styles permettent de changer l'aspect général de l'application, d'un élément ou d'une partie (l'apparence de tous les textes de l'application).
- Un style est une collection de propriétés qui spécifie le design d'une vue, d'un élément ou d'une application.
- Un style peut spécifier différentes propriétés : Padding, Margin, Style de texte, Couleur du texte, Taille du texte, etc.
- Un style se définit dans un fichier ressource XML ([styles.xml](#)).

Structure des Répertoires de Ressources

Création de la ressource style:
res/values/**Styles.xml**

```
<style name="TextStyle2">
    <item name="android:textColor">@color/textColor2</item>
    <item name="android:textStyle">italic|bold</item>
    <item name="android:textSize">@dimen/textSize</item>
    <item name="android:background">@color/btnBackground2</item>
    <item name="android:layout_gravity">center_horizontal</item>
    <item name="android:gravity">center_horizontal</item>
    <item name="android:padding">40dp</item>
</style>
```

Structure des Répertoires de Ressources

Utilisation de la ressource Style :

res/values/[Styles.xml](#)

```
<TextView  
    android:theme="@style/TextStyle2"  
    android:text="Text Style 2"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id/textView8"  
    android:layout_margin="10dp" />
```

Text Style 2