



Institut
Supérieur de
Technologie

BURKINA FASO

UNITÉ-PROGRÈS-JUSTICE

MASTER2: RÉSEAUX INFORMATIQUES ET MULTIMÉDIA

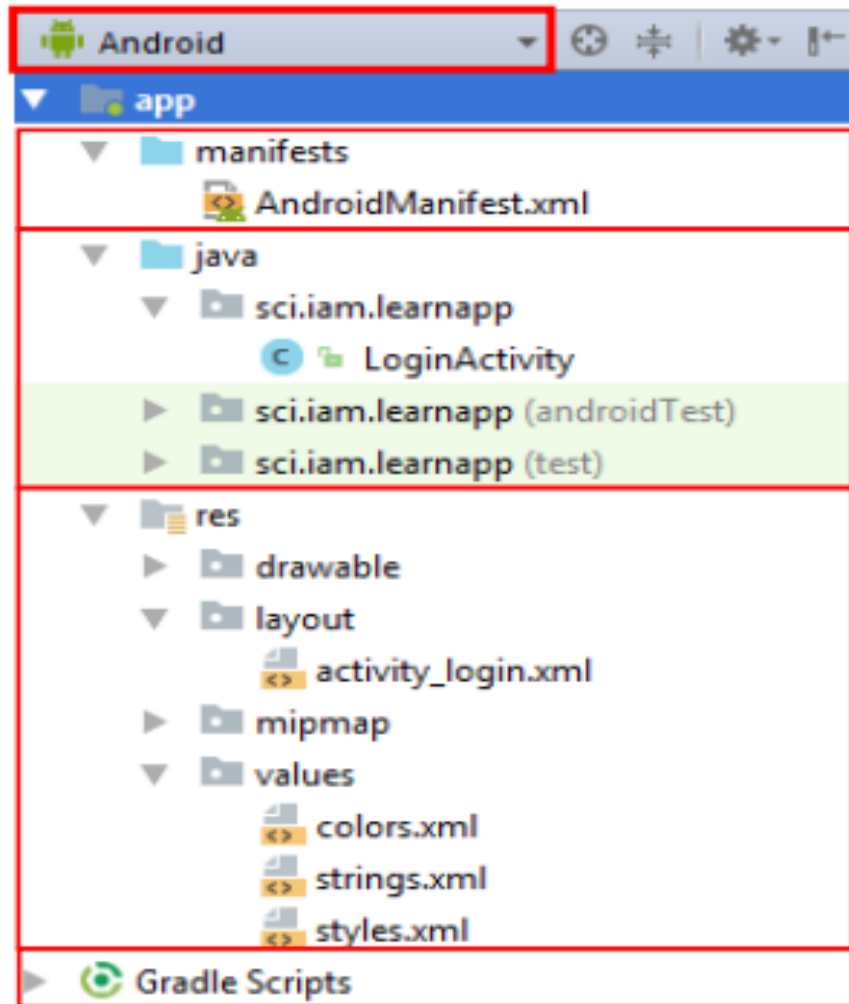
PROGRAMMATION MOBILE

Structure d'un projet Android

COMPAORE MOCTAR

16 April 2022

STRUCTURE D'UN PROJET ANDROID



Manifest

Fichiers java

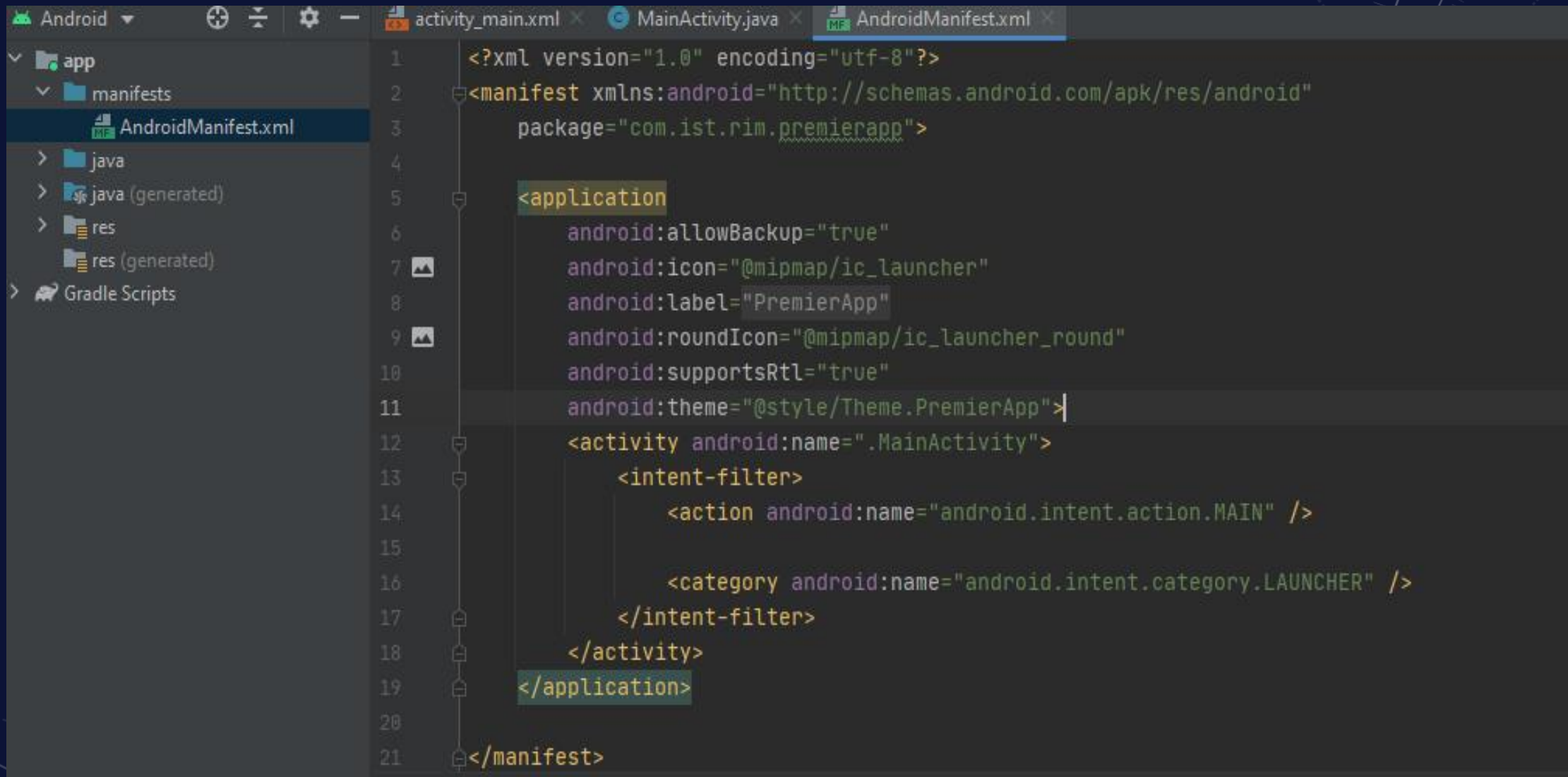
Ressources organisées
(texte, ...)

Gradle

"ANDROIDMANIFEST.XML" (1/2)

- ❖ Décrit les composants de l'application
 - ❖ Définit le point d'entrée potentiel de l'application (activity main)
- ❖ Quatre types de composants
 - ❖ Activités (activities)
 - ❖ Services (services)
 - ❖ Fournisseurs de contenu (content providers)
 - ❖ Récepteurs de diffusion (broadcast receivers)
- ❖ Décrit les permissions requises par l'application
 - ❖ Accès à l'Internet, lecture-écriture dans la liste de contacts,
 - ❖ Géo-localisation, . . .
- ❖ Décrit les besoins matériels et logiciels de l'application
 - ❖ Appareil photo, bluetooth, écran multi-touch, . . .

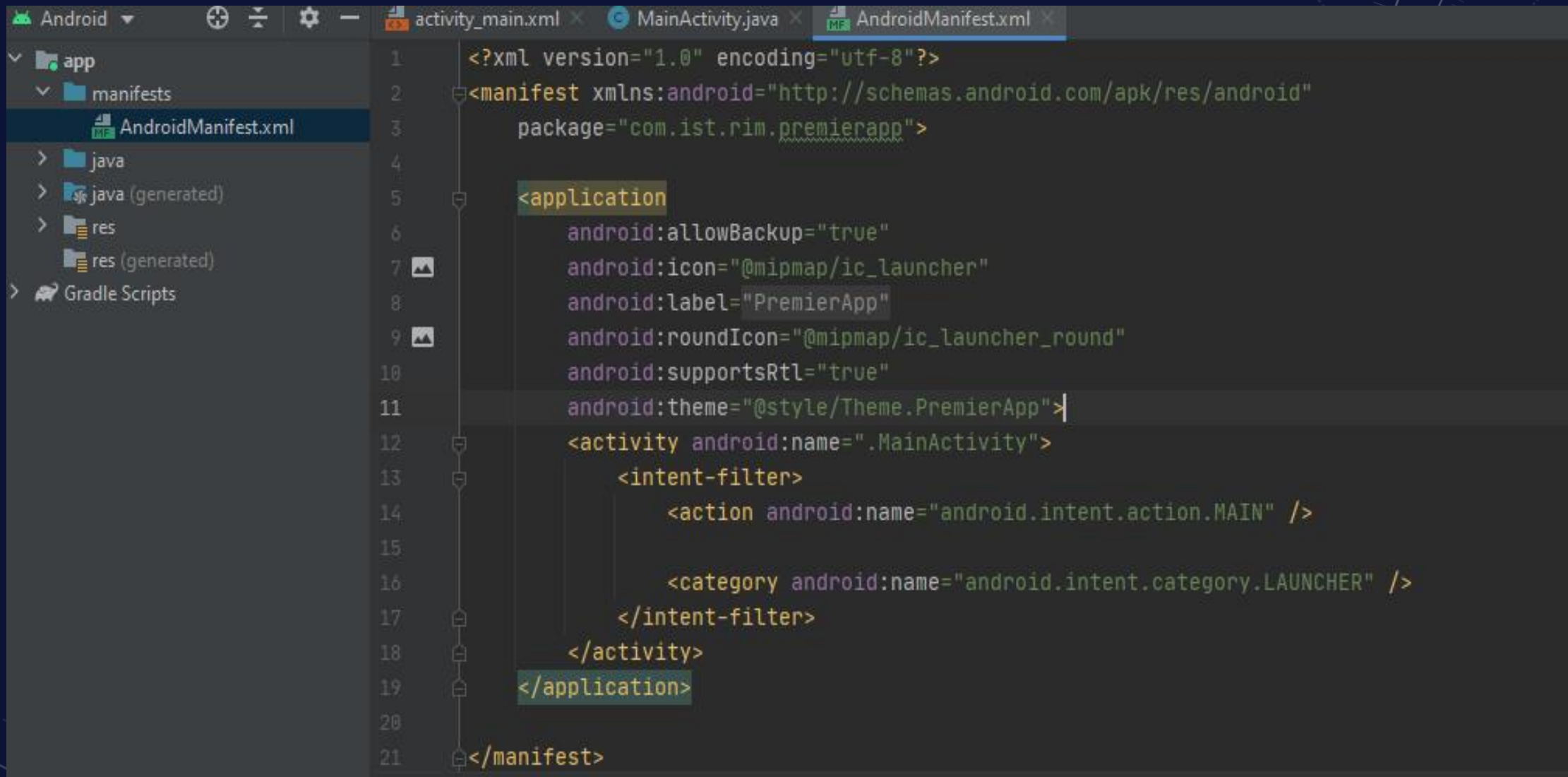
"ANDROIDMANIFEST.XML" (2/2)



```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3          package="com.ist.rim.premierapp">
4
5      <application
6          android:allowBackup="true"
7          android:icon="@mipmap/ic_launcher"
8          android:label="PremierApp"
9          android:roundIcon="@mipmap/ic_launcher_round"
10         android:supportsRtl="true"
11         android:theme="@style/Theme.PremierApp">
12         <activity android:name=".MainActivity">
13             <intent-filter>
14                 <action android:name="android.intent.action.MAIN" />
15
16                 <category android:name="android.intent.category.LAUNCHER" />
17             </intent-filter>
18         </activity>
19     </application>
20
21 </manifest>
```

Fichier XML obligatoire, à la racine du projet et sont obligatoires category.LAUNCHER : point d'entrée de l'application

"ANDROIDMANIFEST.XML" (2/2)



```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3          package="com.ist.rim.premierapp">
4
5      <application
6          android:allowBackup="true"
7          android:icon="@mipmap/ic_launcher"
8          android:label="PremierApp"
9          android:roundIcon="@mipmap/ic_launcher_round"
10         android:supportsRtl="true"
11         android:theme="@style/Theme.PremierApp">
12         <activity android:name=".MainActivity">
13             <intent-filter>
14                 <action android:name="android.intent.action.MAIN" />
15
16                 <category android:name="android.intent.category.LAUNCHER" />
17             </intent-filter>
18         </activity>
19     </application>
20
21 </manifest>
```

Fichier XML obligatoire, à la racine du projet et sont obligatoires category.LAUNCHER : point d'entrée de l'application

ACTIVITY

Une activité = un écran graphique

- ✓ définie dans `./java/`
- ✓ souvent c'est un cas d'utilisation UML
- ✓ contrôle les éléments définis par du code Java

Hérite de:

- ✓ `android.app.Activity` ou
- ✓ `android.support.v7.app.AppCompatActivity;`

Interface graphique (IHM):

- ✓ associé à une vue (`./res/layout/`) certains éléments (texte, dimension, couleur) sont définis dans `./res/values`

ACTIVITY

Cycle de vie

Le changement d'état d'une activité provoque le déclenchement de la méthode callback correspondante

Méthodes callback :

```
void onCreate(...)  
void onStart()  
void onRestart()  
void onResume()  
void onPause()  
void onStop()  
void onDestroy()
```

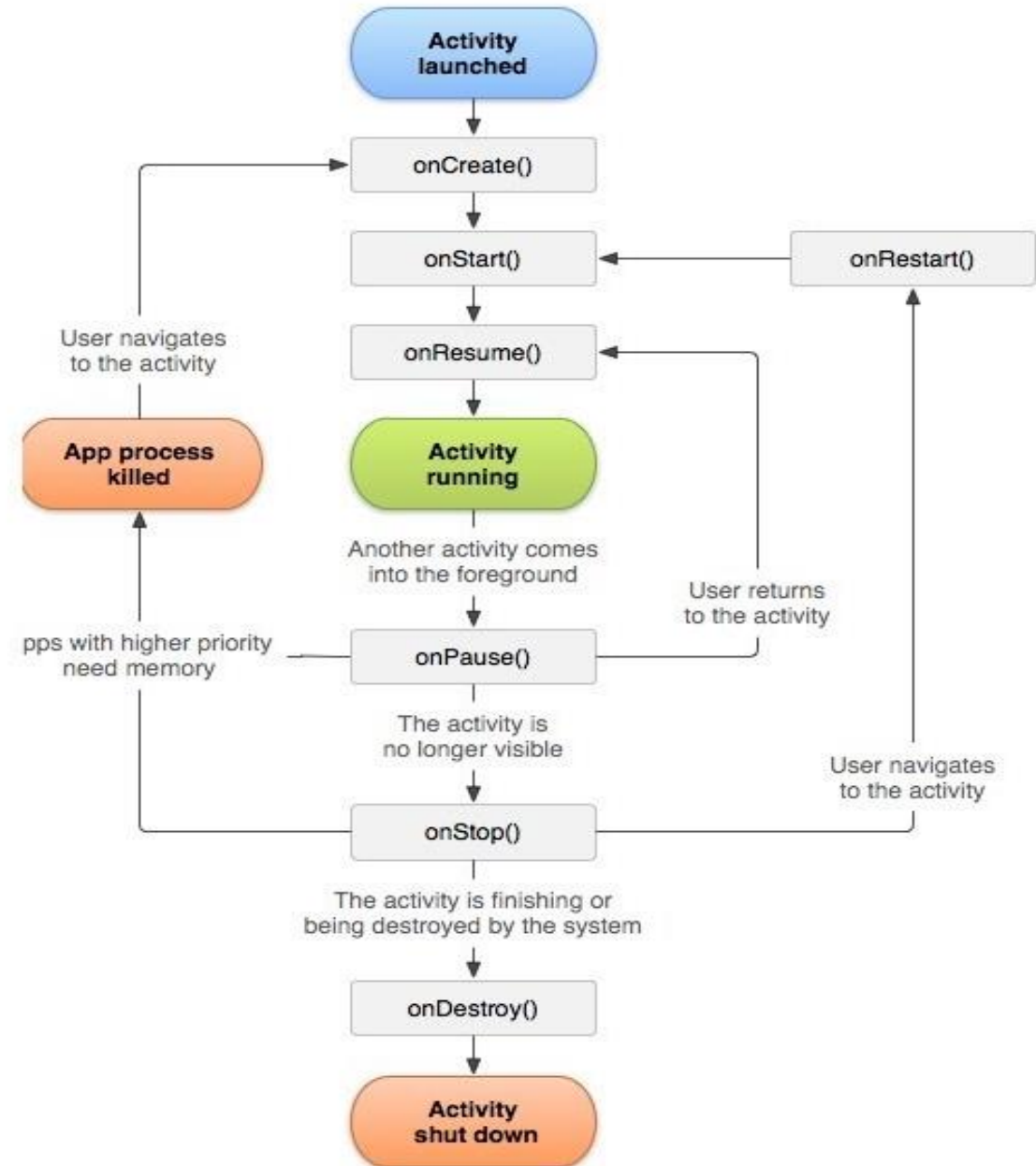
ACTIVITY

Cycle de vie

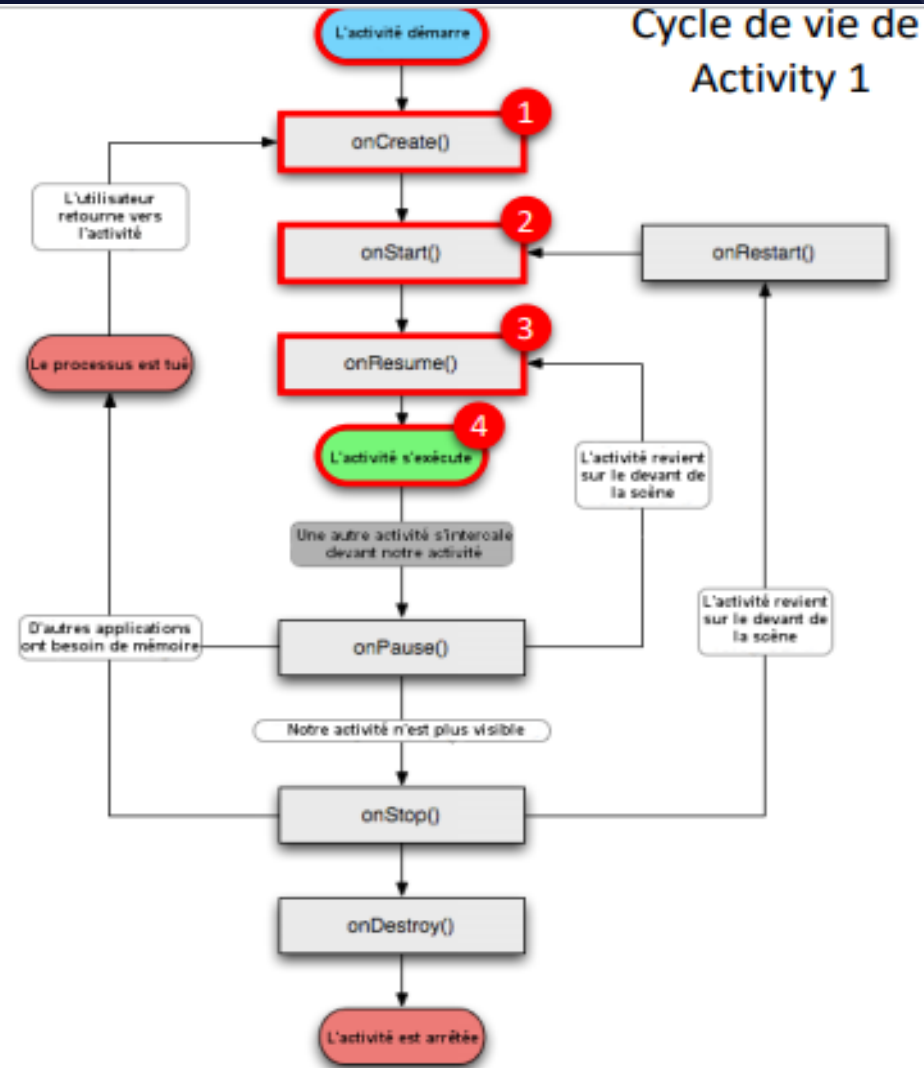
```
1 void onCreate(...) // lorsque l'activité est créée
2 void onStart()      // lorsque l'activité est démarrée (visible)
3 void onRestart()    // lorsque l'activité redémarre après un arrêt
4 void onResume()     // lorsque l'activité redémarre après une pause
5 void onPause()      // lorsque l'activité est en pause (invisible)
6 void onStop()       // lorsque l'activité s'arrête
7 void onDestroy()    // lorsque l'activité est détruite
```


ACTIVITY

Cycle de vie

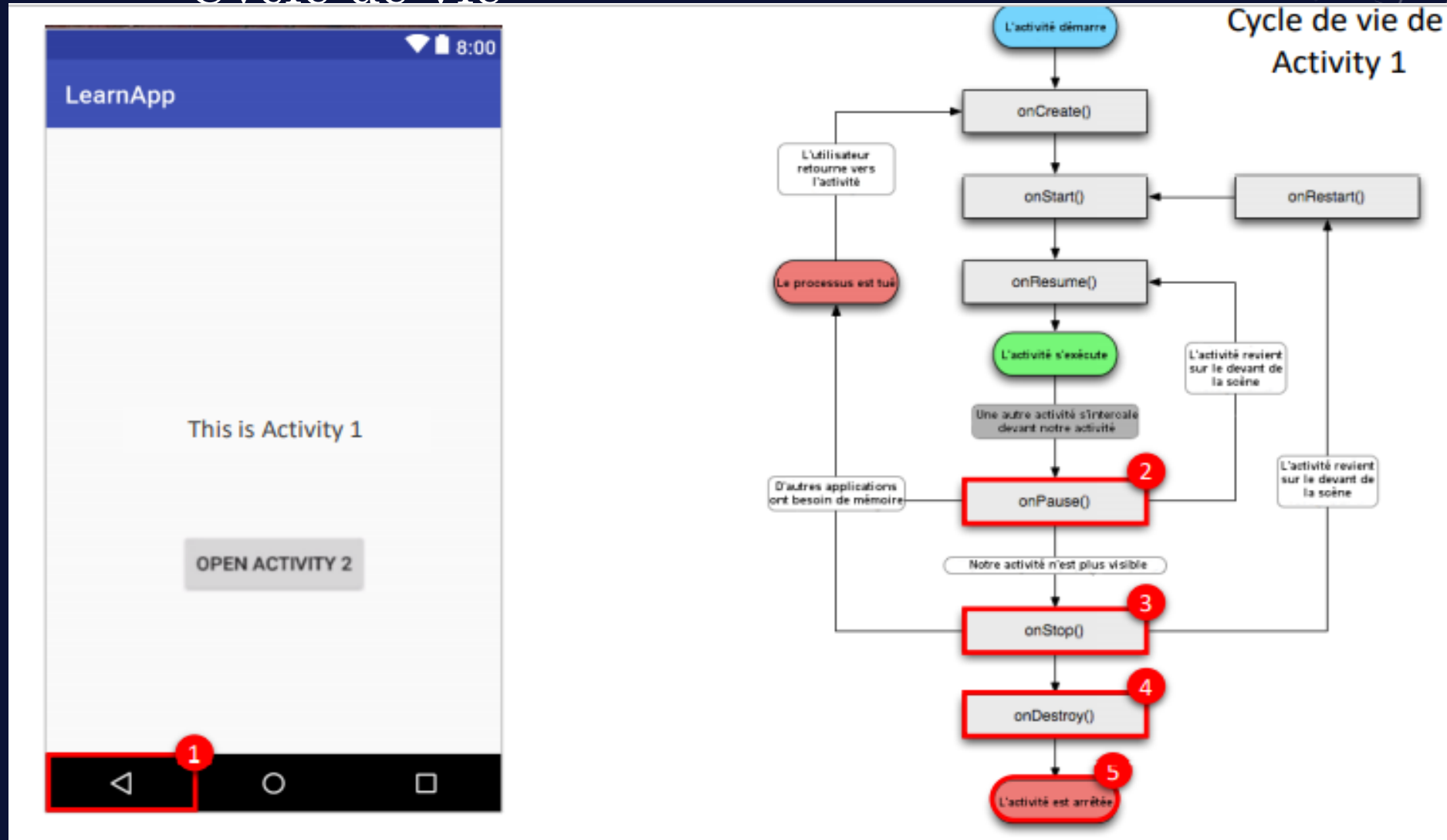


Cycle de vie



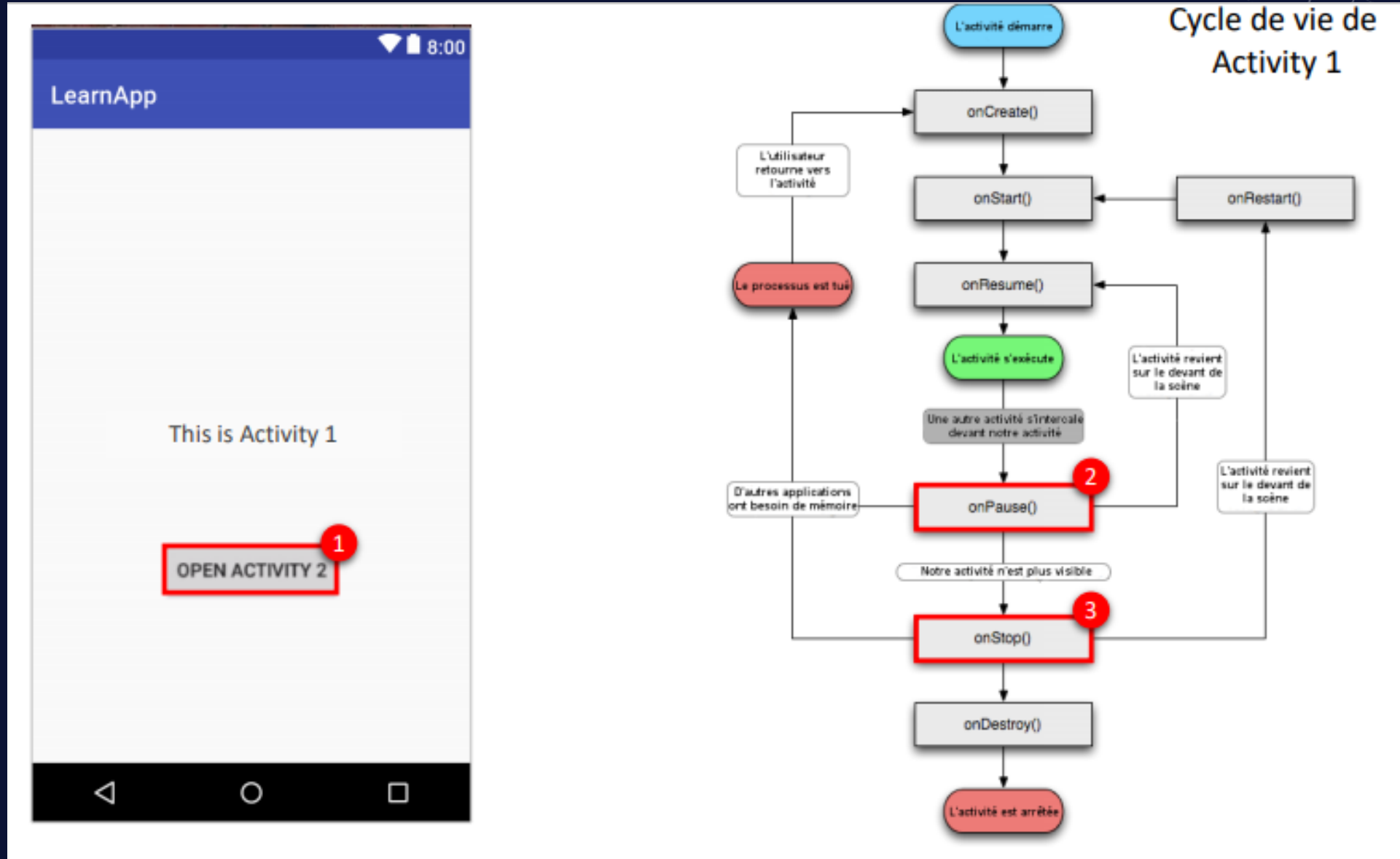
ACTIVITY

Cycle de vie



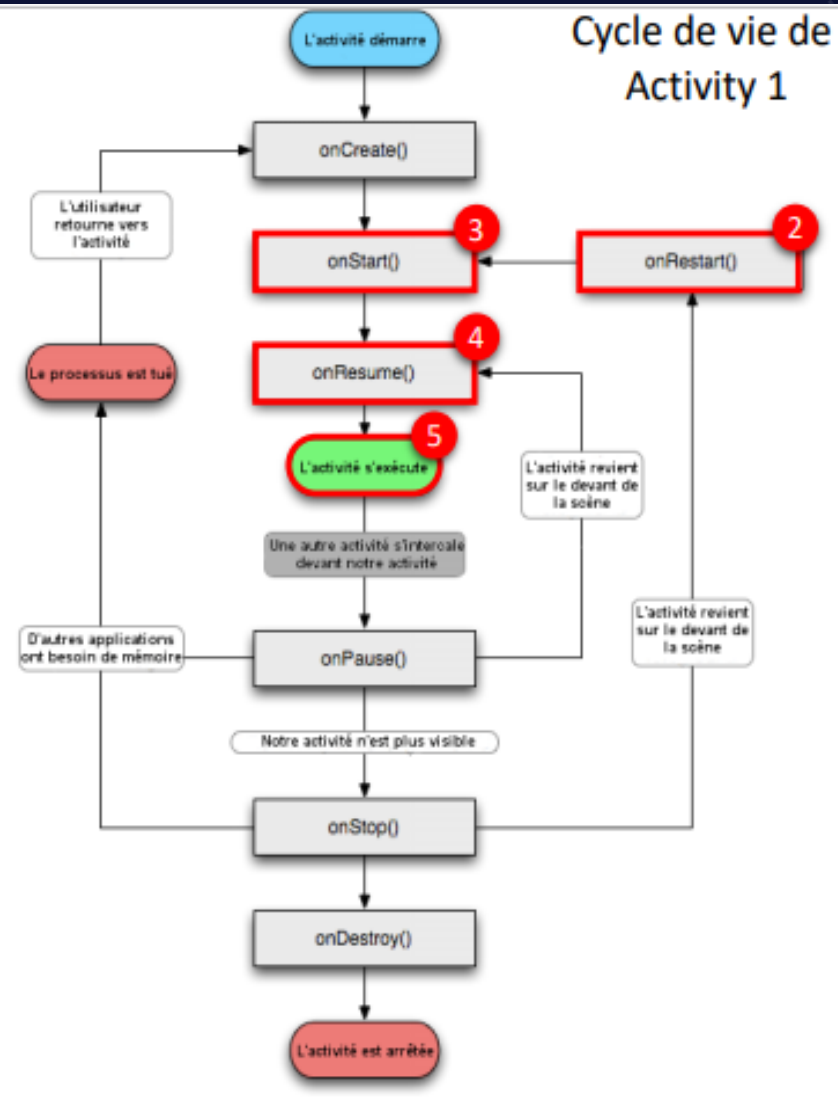
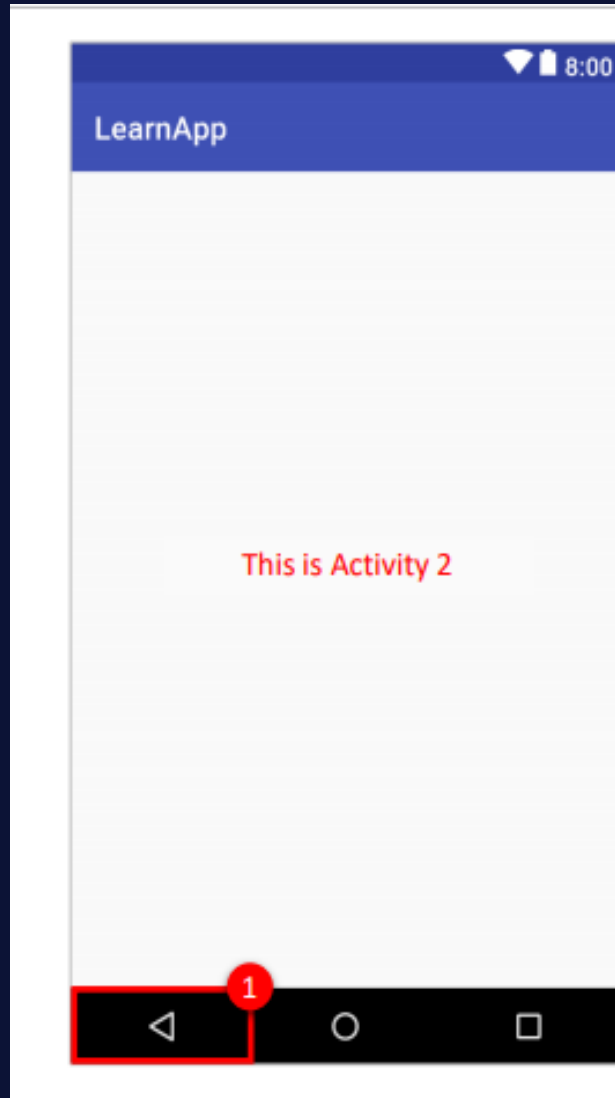
ACTIVITY

Cycle de vie

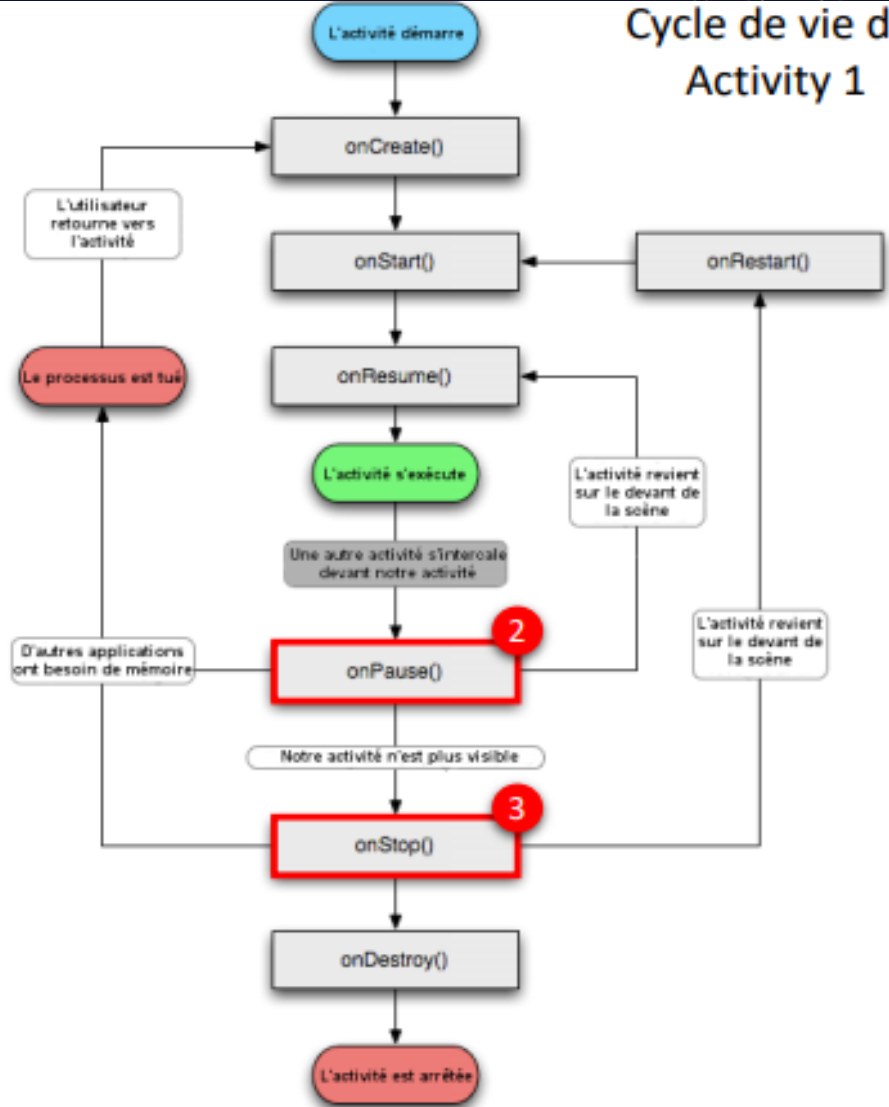
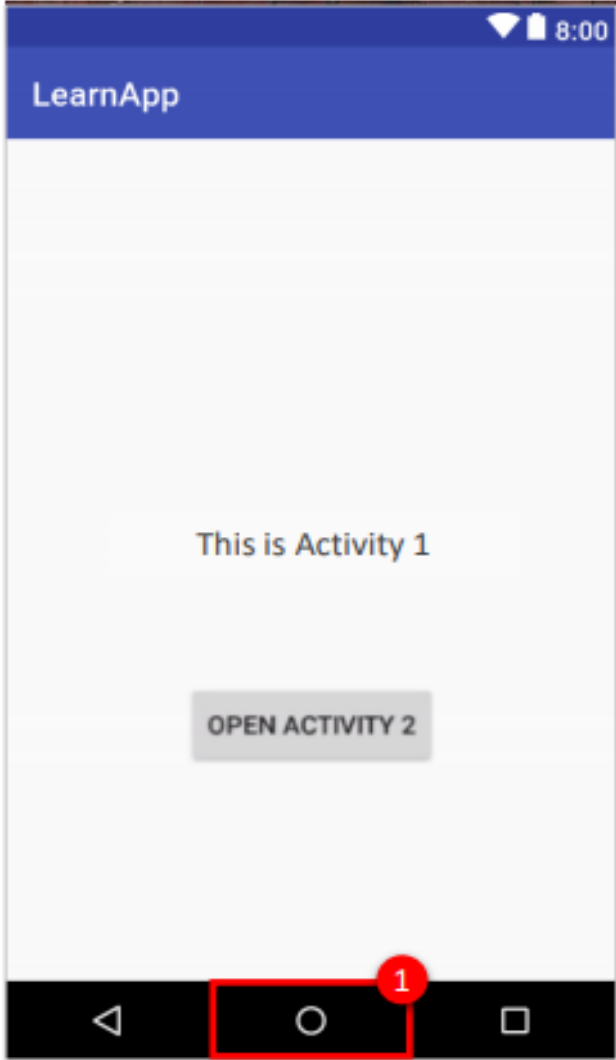


ACTIVITY

Cycle de vie

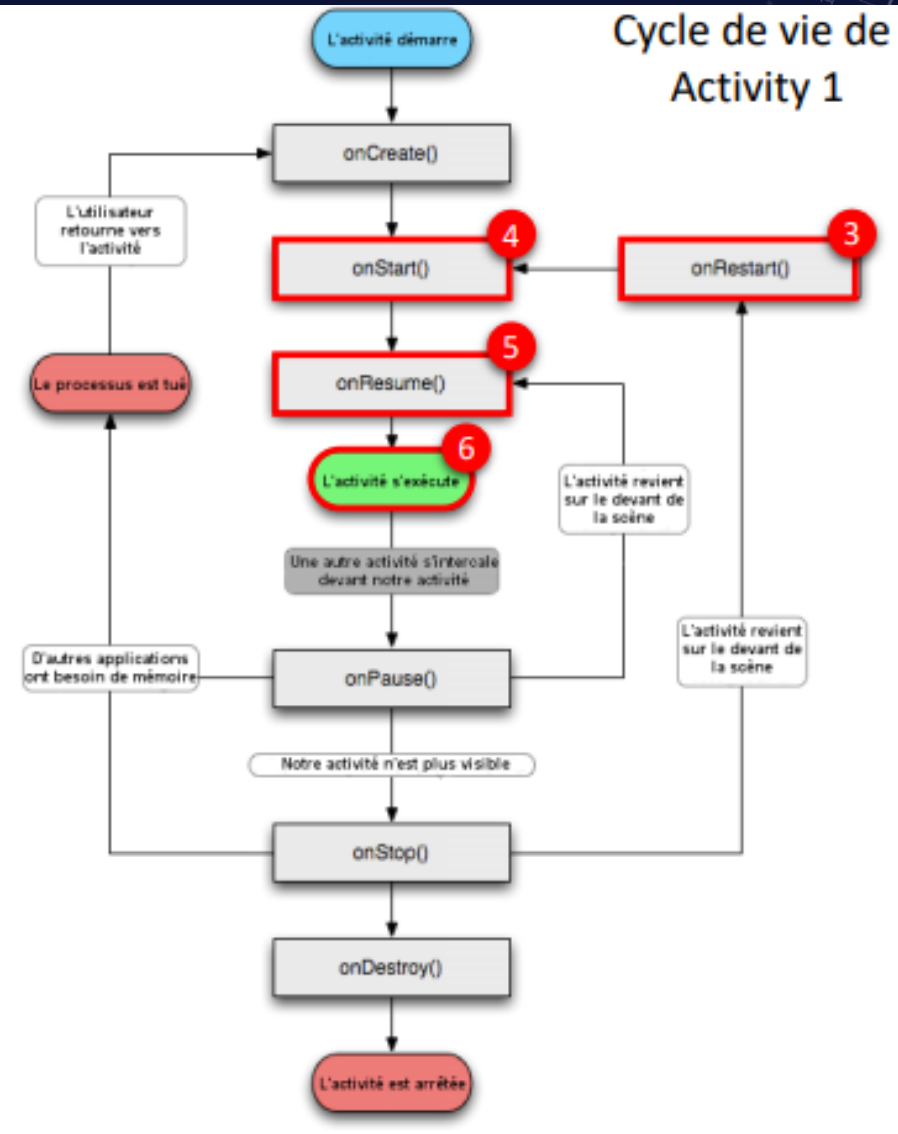
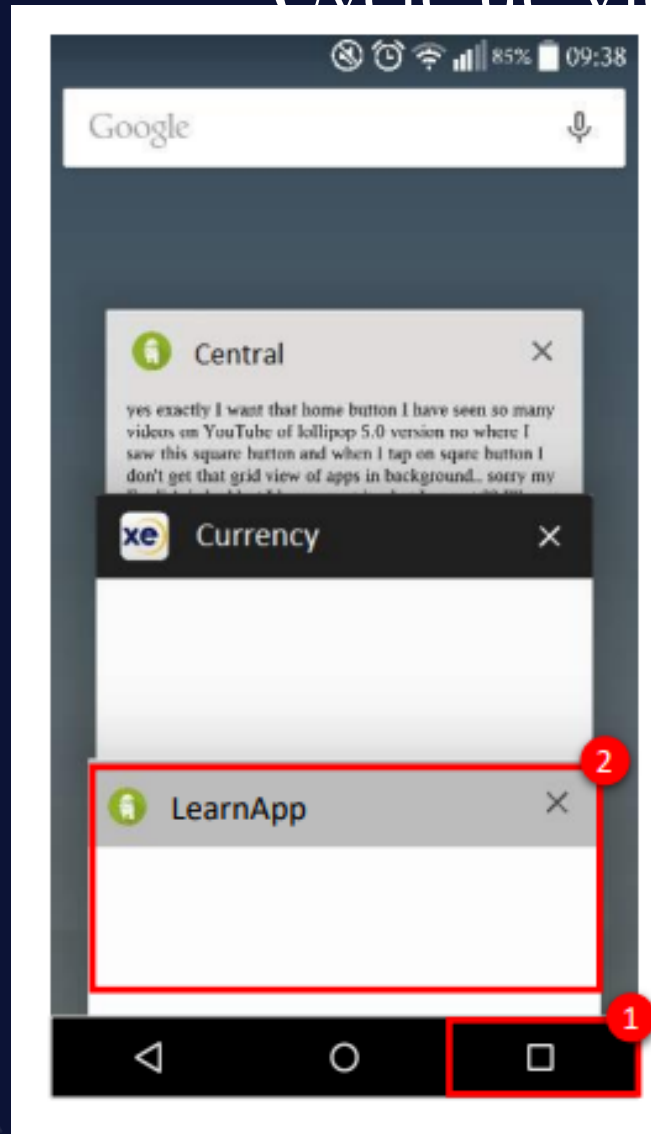


Cycle de vie



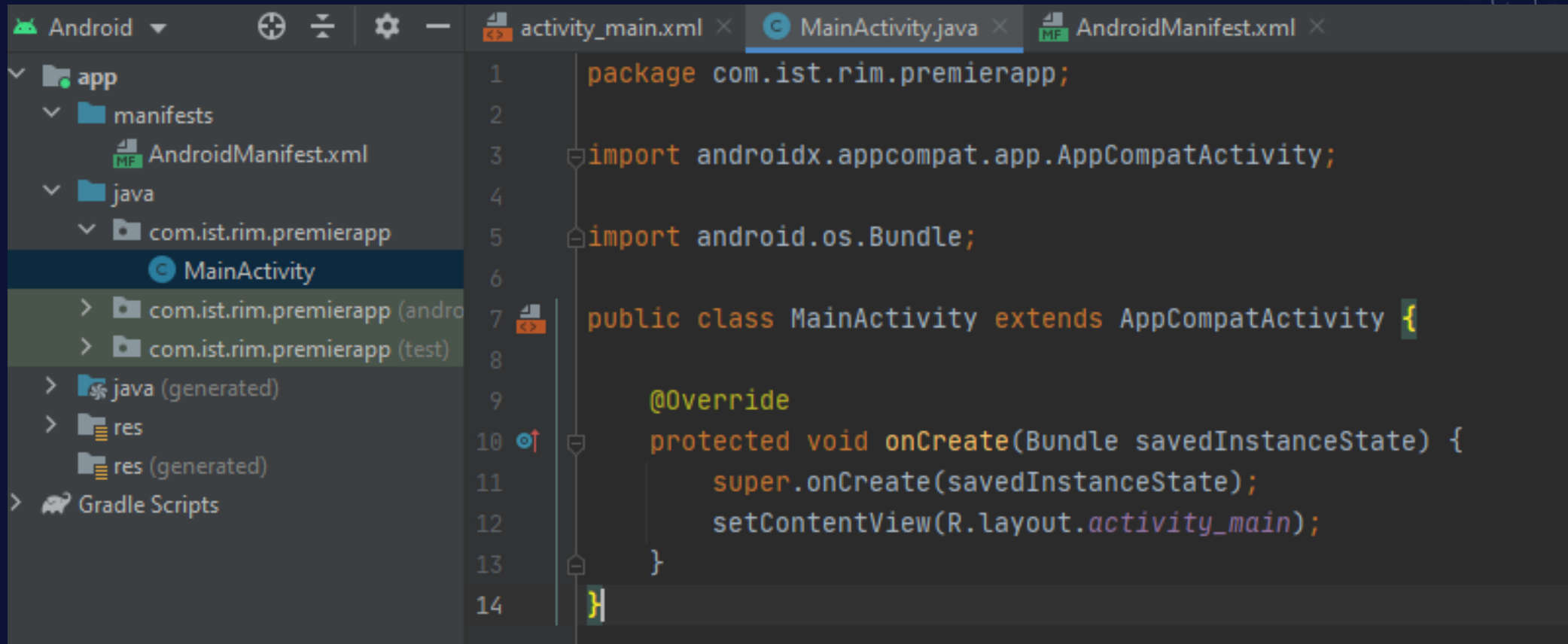
ACTIVITY

Cycle de vie



ACTIVITY

Exemple



```
1 package com.ist.rim.premierapp;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.os.Bundle;
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 }
```

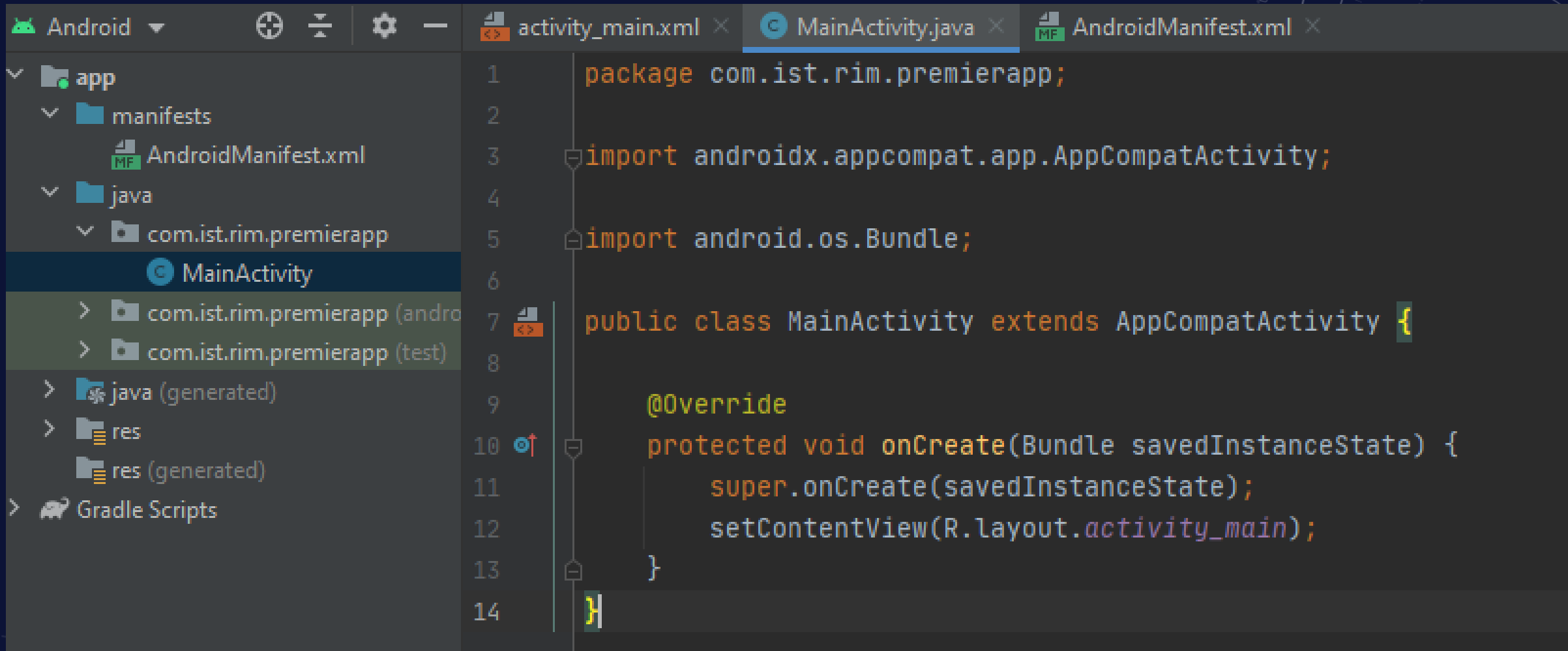
Pour chaque méthode callback, appeler la méthode sur super

Exemple : dans **onCreate()**, appel à **super.onCreate()**

Le **Bundle** mémorise l'état de l'activité lorsqu'elle passe en arrière-plan

ACTIVITY

Exemple



```
1 package com.ist.rim.premierapp;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.os.Bundle;
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 }
```

`setContentView(int layout)` associe à l'activité un Layout référencé par layout

RESSOURCES "./RES/"

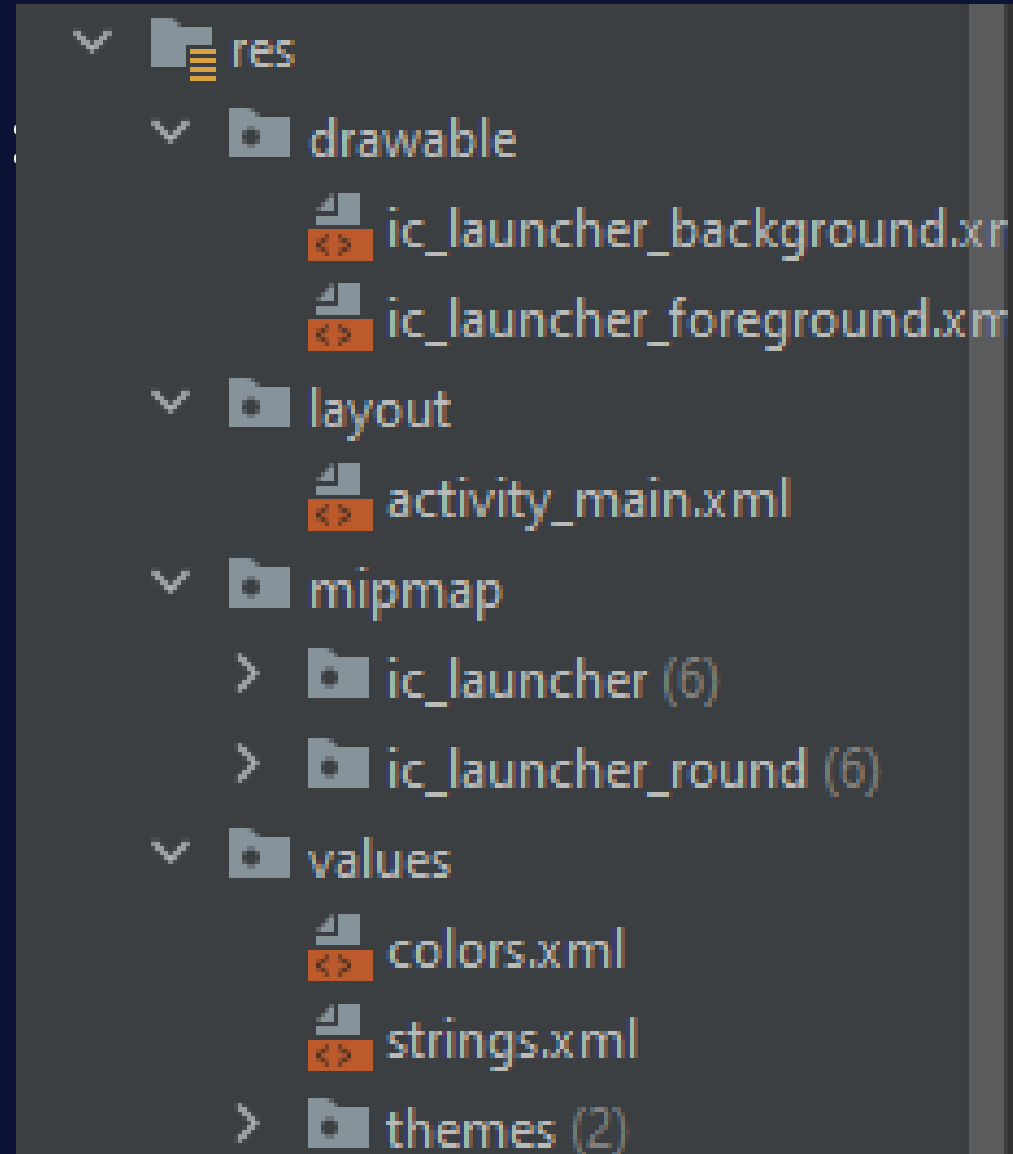
Types de ressources :

❖ Layouts et menus

❖ Icones et images

❖ Valeurs (values) :

- ✓ strings,
- ✓ attrs,
- ✓ styles,
- ✓ colors,
- ✓ dimens,



Images

Interfaces
graphiques

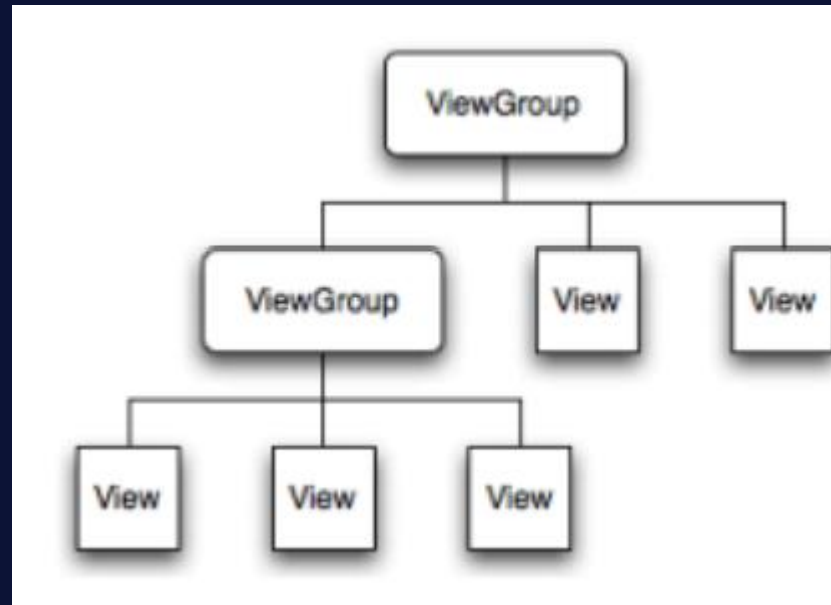
Icones

Valeurs
constantes

RESSOURCES : LAYOUTS

Un layout définit la structure visuelle d'une interface graphique, d'une activité par exemple. Il est possible de déclarer un layout de 2 façons :

- ❖ Déclarer les éléments de l'interface graphique dans un fichier XML situé dans le dossier res/layout/,
- ❖ Instancier et manipuler les éléments du layout dans le code source Java au moment de l'exécution



Android fournit une variété de vues, distinguée en **widjets et **layouts****

RESSOURCES : LAYOUTS

widget

- ❖ Un widget est une vue élémentaire héritant de la classe View. il permet d'afficher du contenu et d'interagir avec l'application.
- ❖ Les widgets sont souvent sensibles à des évènements (clic, frappe de clavier, ...).
- ❖ Quelques widgets : Button, EditText, TextView, CheckBox et RadioButton.

Layout

- ❖ Conteneur de vues
- ❖ Instance de la classe ViewGroup
- ❖ Exemples : LinearLayout, TableLayout, RelativeLayout, FrameLayout, ScrollView

RESSOURCES : LAYOUTS

Types de ViewGroup

LinearLayout



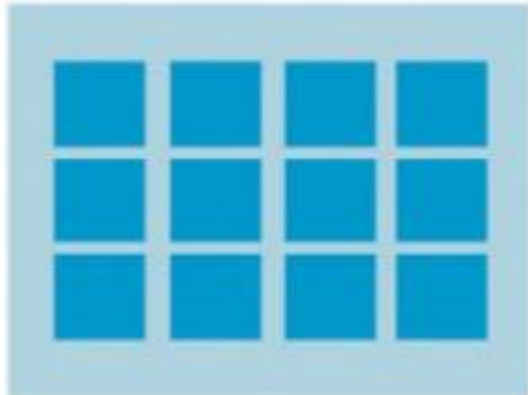
RelativeLayout



WebView



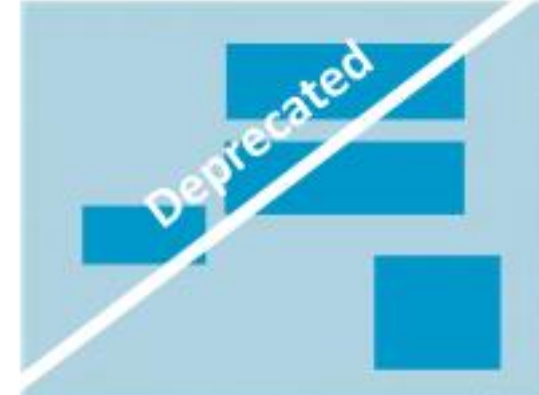
GridLayout



FrameLayout



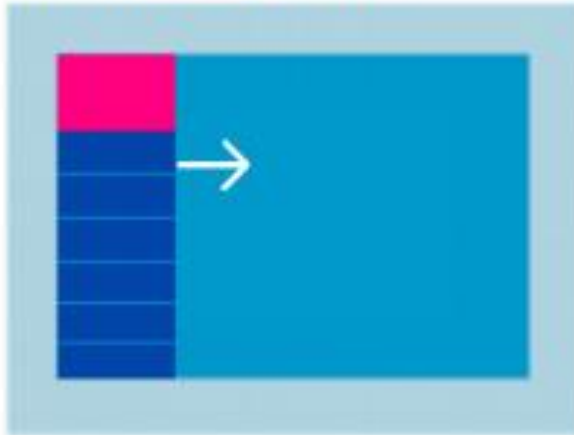
AbsoluteLayout



RESSOURCES : LAYOUTS

Types de ViewGroup

DrawerLayout



CoordinatorLayout



SwipeRefreshLayout



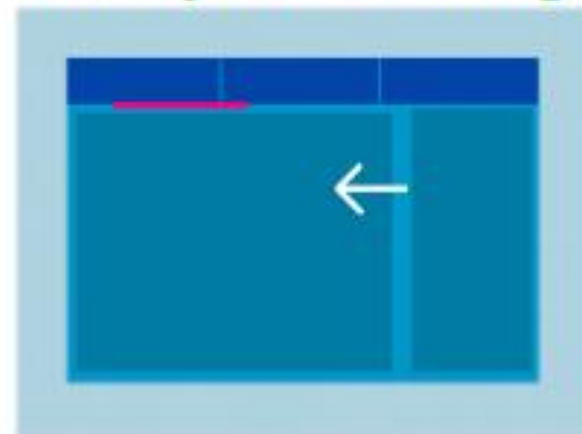
SlidingPaneLayout



TableLayout



TabLayout-ViewPager



RESSOURCES : LAYOUTS

Exercice:

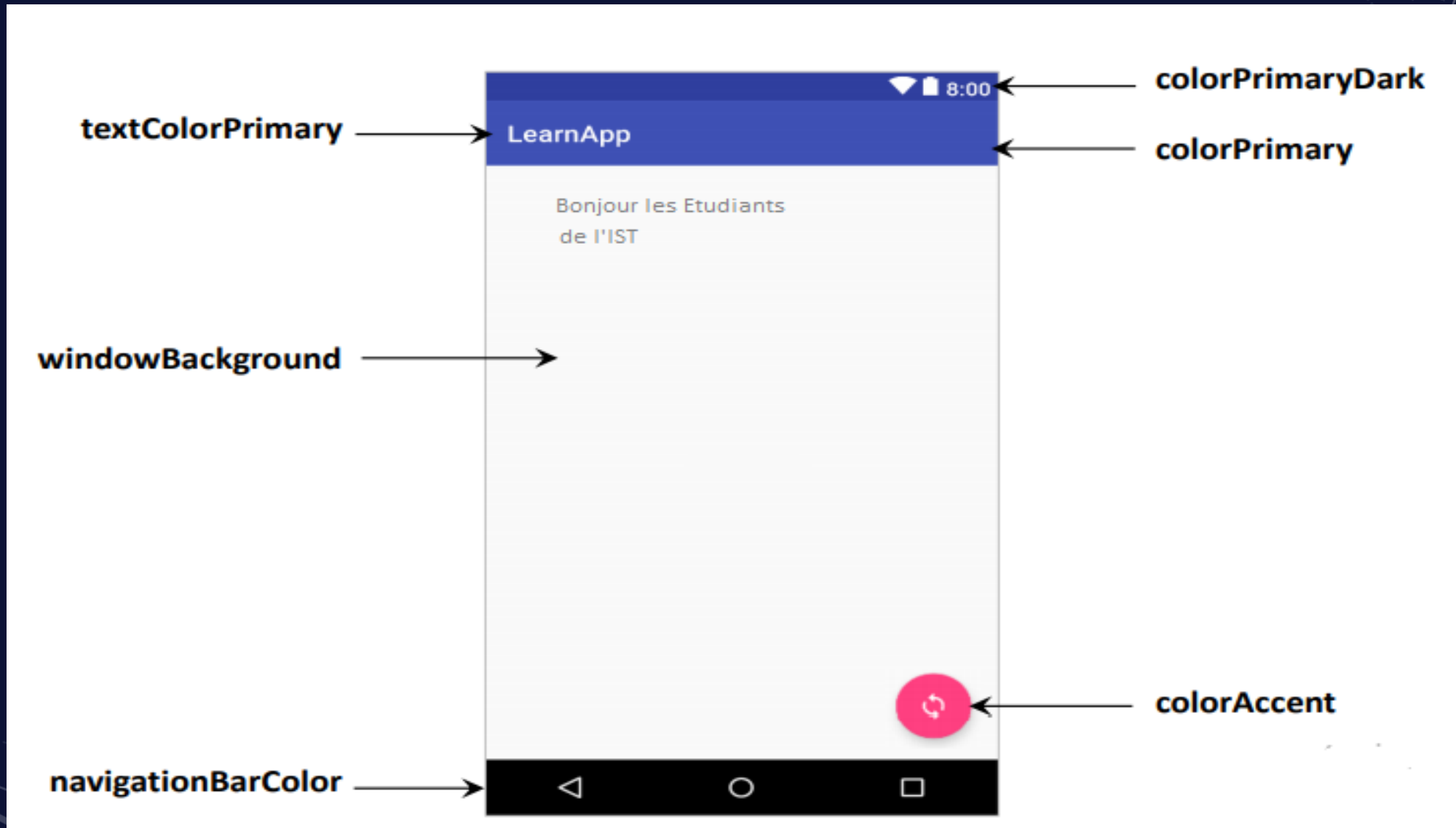
- ❖ Trouver le layout `main_activity.xml`
- ❖ Quel est son type ?
- ❖ Changer le type en `LinearLayout`
- ❖ Changer la valeur du `TextView` en:
- ❖ En « Bonjour les Etudiants »
- ❖ Ajouter un autre `TextView` qui s'alignera en bas du premier et sa valeur est « de l'IST »;

RESSOURCES : LAYOUTS

Attributs de LinearLayout

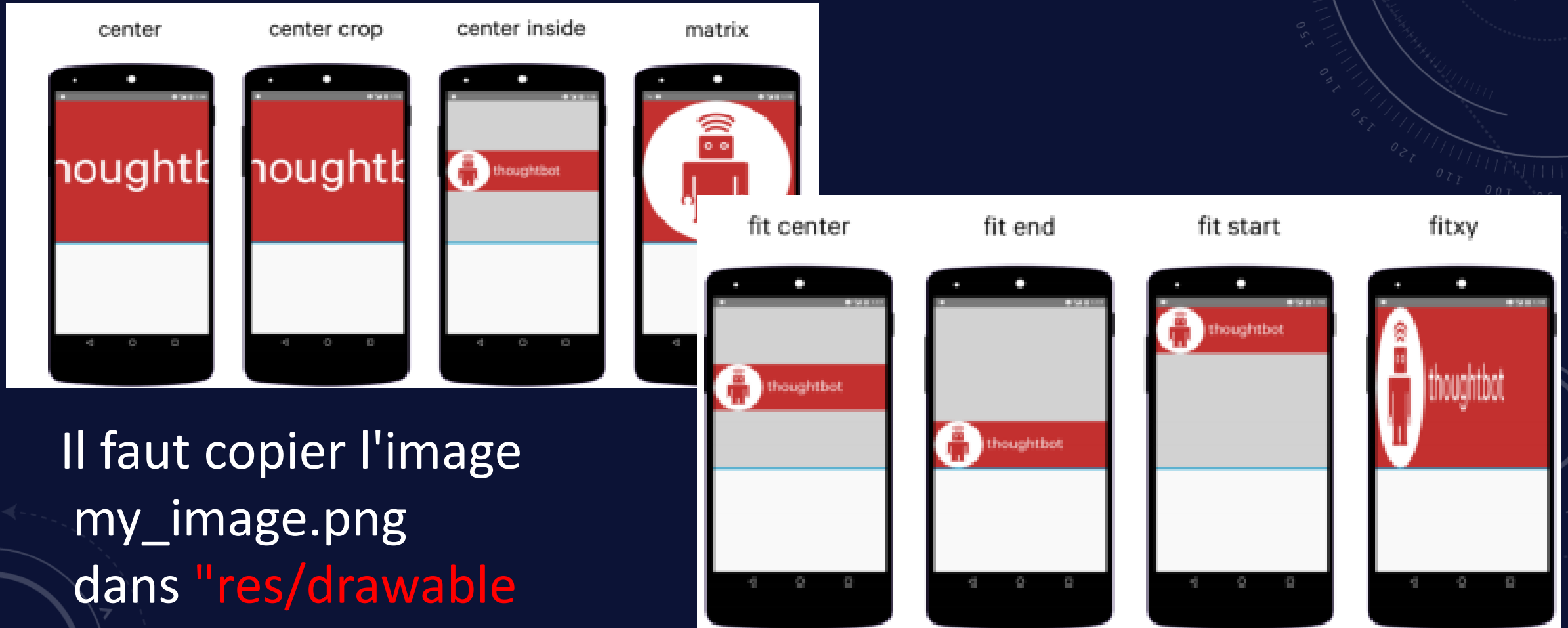
- ❖ **orientation** : **vertical** | **horizontal**
- ❖ **layout_width** et **layout_height** :
 - ✓ **Valeur constante** : **en dp ou px**
 - ✓ **wrap_content** : **place minimum**
 - ✓ **match_parent** : **taille du parent**
- ❖ **gravity** : **top, bottom, left, right, center, ...**
- ❖ **padding** et **margin** : **en dp ou px**

RESSOURCES : COULEURS



RESSOURCES : DRAWABLES

- ❖ `android:src : @drawable/my_image`
- ❖ `android:scaleType :`



Il faut copier l'image
my_image.png
dans "**res/drawable**"

RESSOURCES : DRAWABLES

Exemple:

- ❖ Ajouter le logo de l'IST au projet, et le centrer

Ajouter le logo dans "res/drawable"

Ajouter une ImageView dans le layout

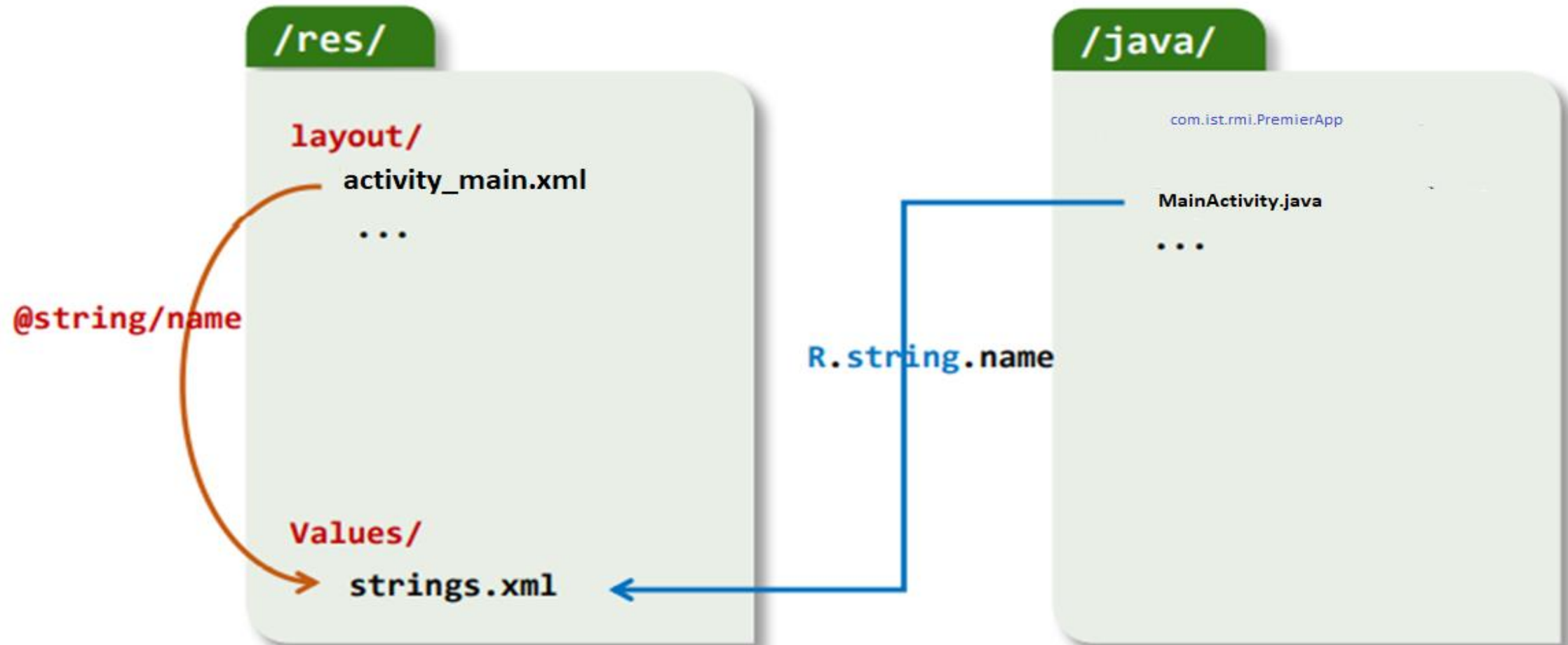
RESSOURCES : STRINGS

`/res/values/strings.xml`

```
<resources>
  <string name="app_name">PremierApp</string>
</resources>
```

- ❖ Exploitation dans res (fichiers XML : layout, styles, ...) : `@string/app_name`
- ❖ Exploitation dans java (code source) : `R.string.app_name`
- ❖ Accès à la ressource :
`getResources().getString(R.string.app_name);`

RESSOURCES : STRINGS



LE FICHER "R.JAVA"

chemin : "\\app\\build\\generated\\source\\r\\debug\\[package]\\R.java"

- ❖ Les ressources sont utilisées grâce à un identifiant.
- ❖ De ce fait, un fichier, appelé **R.java**, est généré par Android, permettant de référencer toutes les ressources définies dans **./res** à partir de fichiers source Java.
- ❖ Il n'est pas modifiable et il est mis à jour dynamiquement lorsqu'une ressource est créée ou modifiée.
- ❖ Les références dans R sont représentées par des constantes de type entier sur 32 bits,
- ❖ exemple : 0x1A34F678

LE FICHER "R.JAVA"

- ❖ il existe 2 types de ressources via R :
 - ✓ Ressources définies par le développeur : ce sont toutes les ressources se trouvant dans le dossier ./res, par exemple `R.color.vert` (dans `res : @color/vert`)
 - ✓ Ressources prédéfinies dans Android : Pour référencer les ressources qui sont prédéfinies dans Android, il faut utiliser `android.R.color.black` (dans `res : @android:color/black`)

IDENTIFICATION D'UNE VUE – LAYOUT

```
<TextView
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:text="Bonjour Les Etudiants"
```

```
    android:id="@+id/text1"
```

```
/>
```

@+id permet d'ajouter un nouveau nom de ressource à la classe R

IDENTIFICATION D'UNE VUE – LAYOUT

```
public class MainActivity extends AppCompatActivity {  
    TextView textView1;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        textView1 = (TextView) findViewById(R.id.text1);  
        textView1.setText(« Bonjour tout le monde !");  
    }  
}
```

findViewById(int id) renvoie un objet View référencé par id

TP: CRÉATION D'UNE VUE D'AUTHENTIFICATION

Créer une activité LoginActivity

- ❖ Une **ImageView** pour afficher une icone
- ❖ Un **EditText** pour introduire l'identifiant
 - ✓ Avec son **TextView**
- ❖ Un **EditText** pour introduire le mot de passe
 - ✓ Avec son **TextView**
- ❖ Un **Button** pour vérifier l'authentification
 - ✓ Vérifier si les champs sont valides

TAILLE ET DENSITÉ DES ÉCRANS



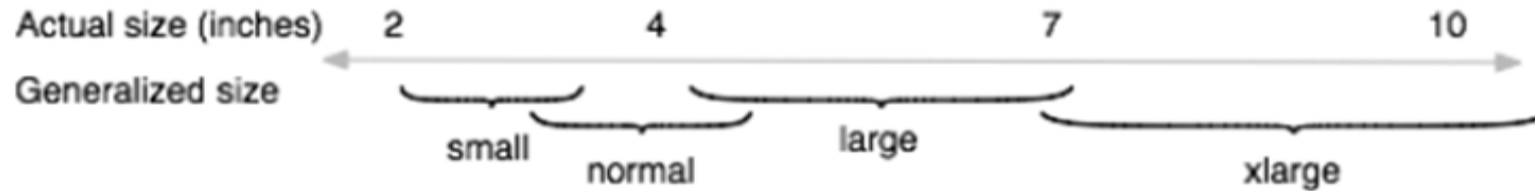
TAILLE ET DENSITÉ DES ÉCRANS

- ❖ Avec la diversité des machines sous lesquelles fonctionne Android, il faut exploiter toutes les opportunités offertes par les ressources pour développer des applications qui fonctionnent sur la majorité des terminaux.
- ❖ Une application Android polyvalente possède un fichier XML pour chaque type d'écran, de façon à pouvoir s'adapter.
- ❖ En effet, si vous développez une application uniquement à destination des petits écrans, les utilisateurs de tablettes trouveront votre application illisible et ne l'utiliseront pas.

TAILLE ET DENSITÉ DES ÉCRANS

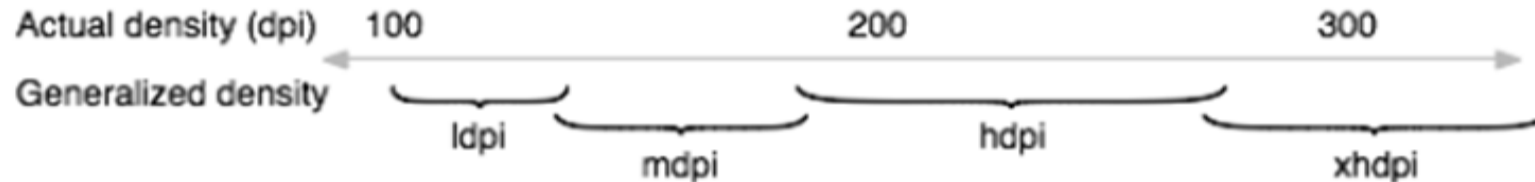
TAILLES DES ÉCRANS : 1 INCH = 2,54 CM


- small, normal, large, xlarge

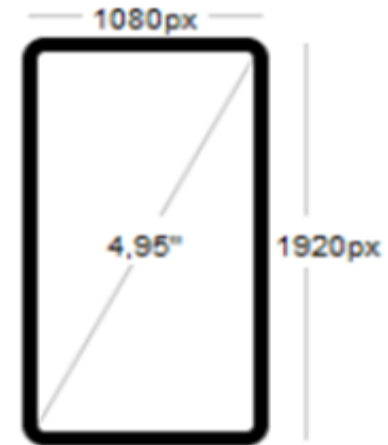


DENSITÉ : DPI (DOT PER INCH)

- ldpi, mdpi, hdpi, xhdpi, xxxhdpi



 Nexus 5



Size: normal
Ratio: notlong
Density: xhdpi

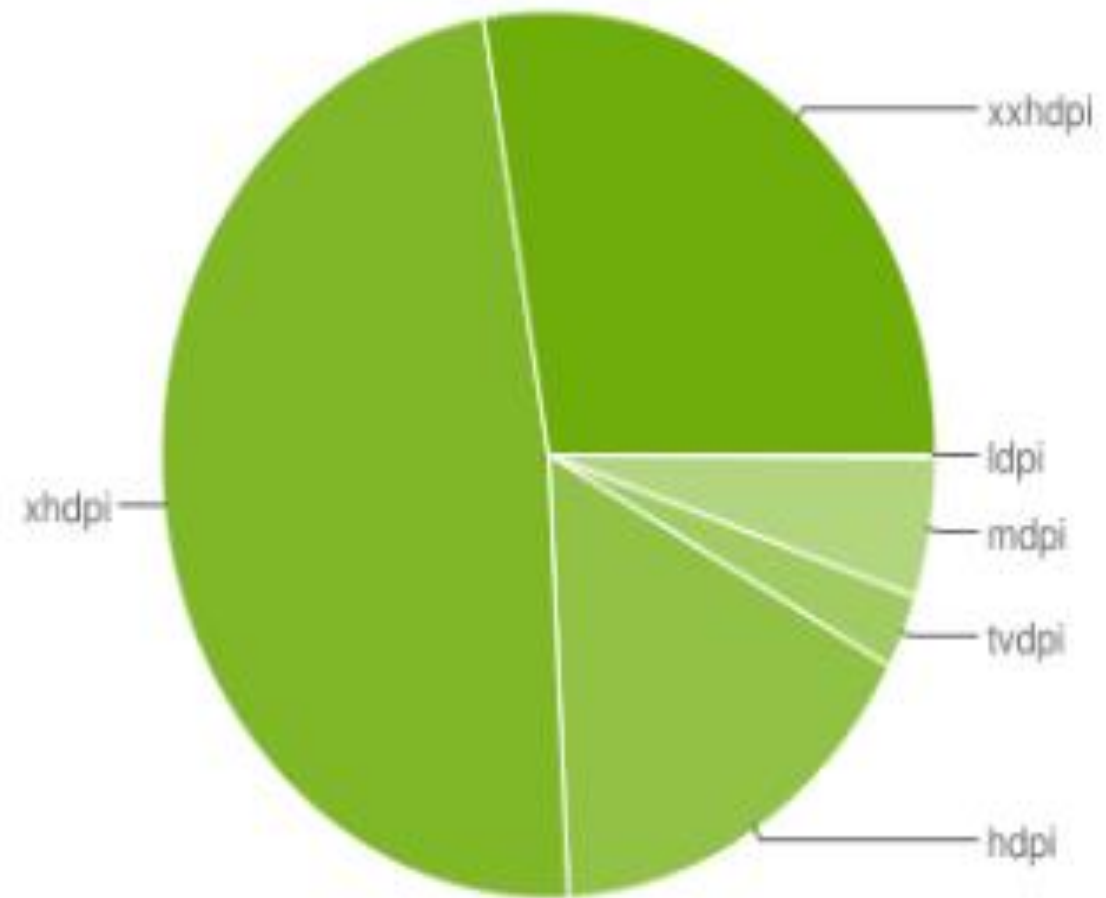
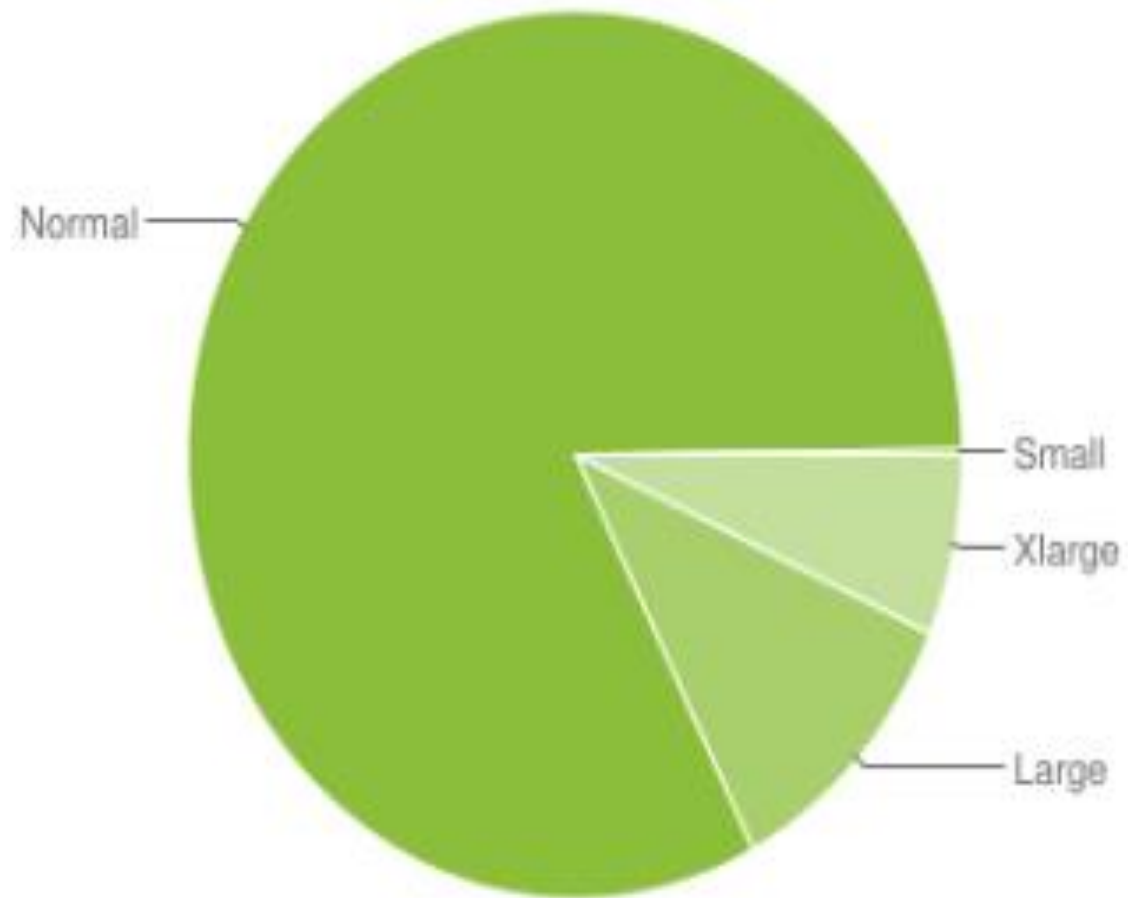
TAILLE ET DENSITÉ DES ÉCRANS

La répartition des différents appareils Android dans le monde selon leur taille et leur densité :

	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	0.1%				0.1%		0.2%
Normal		0.2%	0.3%	12.9%	43.6%	25.4%	82.4%
Large		1.4%	2.4%	0.7%	4.0%	2.3%	10.8%
Xlarge		3.5%		2.7%	0.4%		6.6%
Total	0.1%	5.1%	2.7%	16.3%	48.1%	27.7%	

Source: <https://developer.android.com/about/dashboards/> 31/07/ 2021

TAILLE ET DENSITÉ DES ÉCRANS



TAILLE ET DENSITÉ DES ÉCRANS

DENSITÉ INDÉPENDANTE/ DÉPENDANTE

L'indépendance de la densité, permet de préserver la taille physique des éléments graphiques lorsqu'ils sont affichés sur des écrans avec des densités différentes.

En effet, un élément graphique (tel qu'un bouton) apparaîtrait plus grand sur un écran à faible densité et plus petit sur un autre à haute densité.

TAILLE ET DENSITÉ DES ÉCRANS

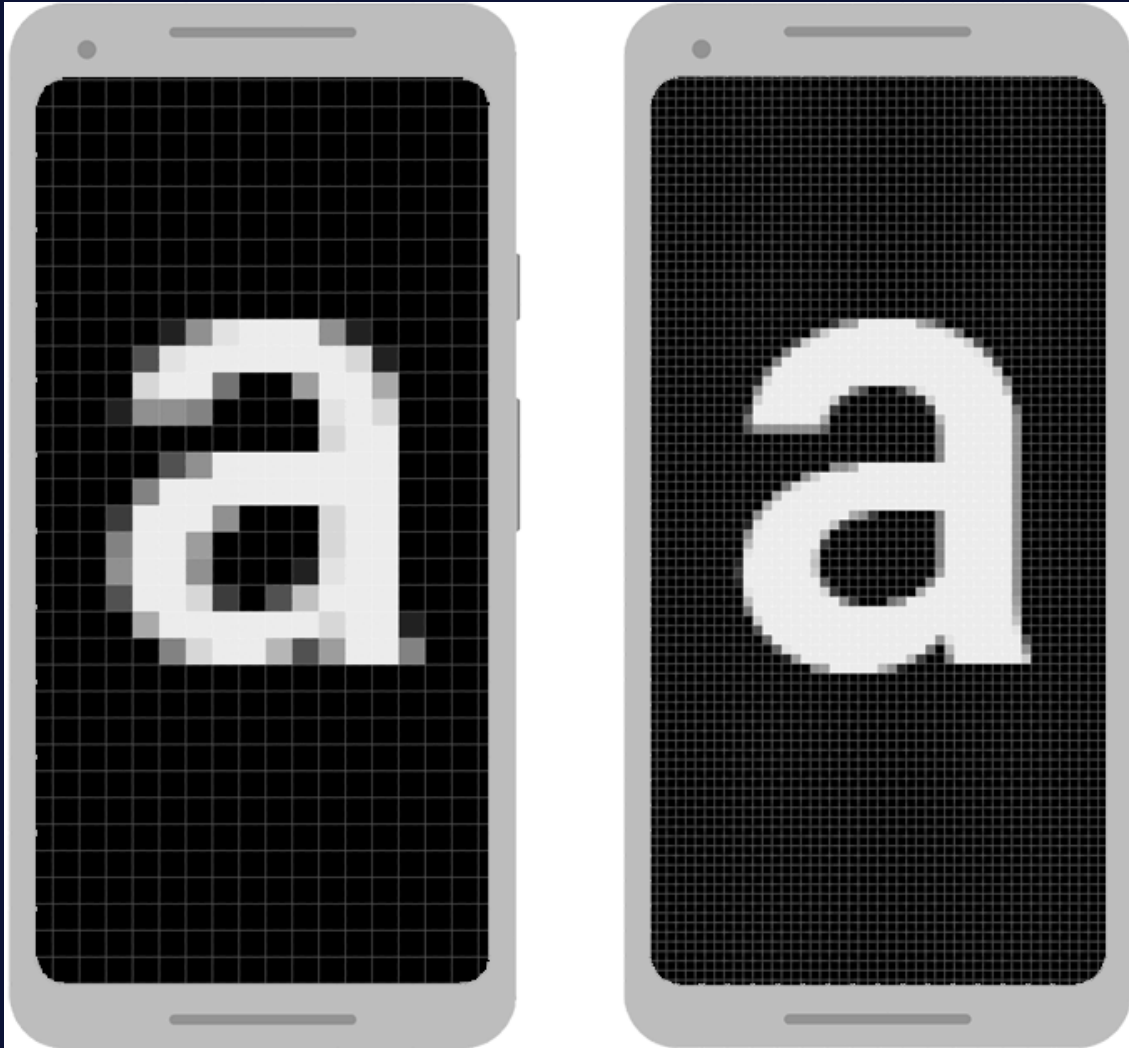
DENSITÉ INDÉPENDANTE/ DÉPENDANTE

Les pixels indépendants de la densité (dp : Density independent Pixel) sont un moyen de spécifier des dimensions d'écran qui s'adaptent à des matériels différents en fonction de leur densité en pixels.

Un dp est calculé en fonction de la densité de l'écran (dpi) : $dp = px / (dpi / 160)$.

TAILLE ET DENSITÉ DES ÉCRANS

DENSITÉ INDÉPENDANTE/ DÉPENDANTE

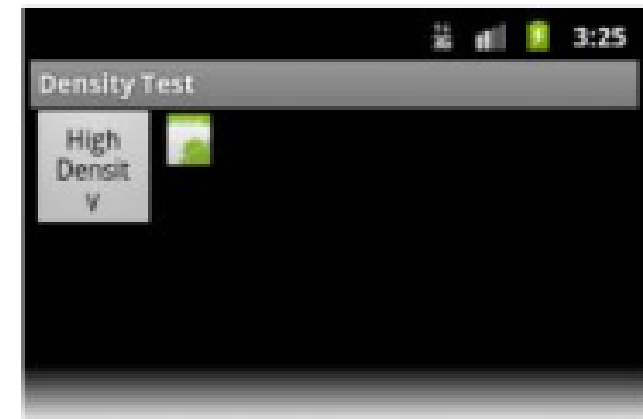
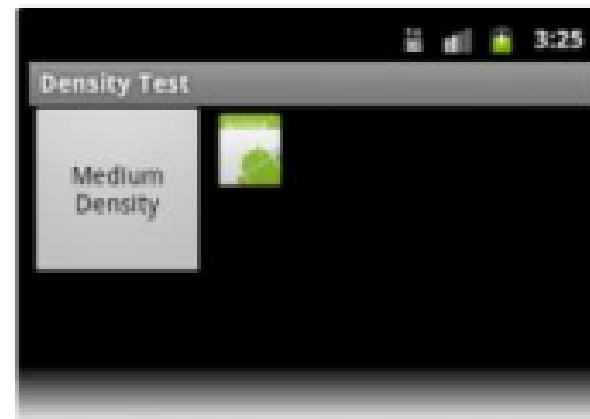
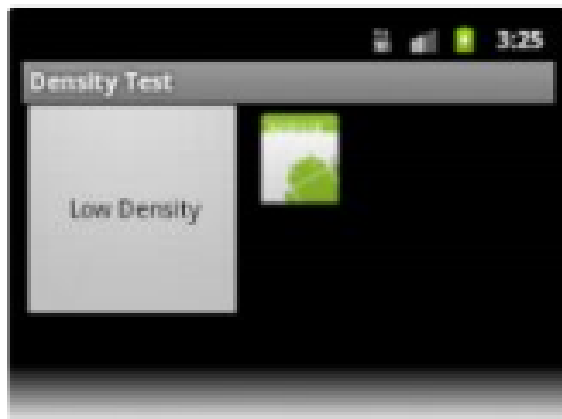


TAILLE ET DENSITÉ DES ÉCRANS

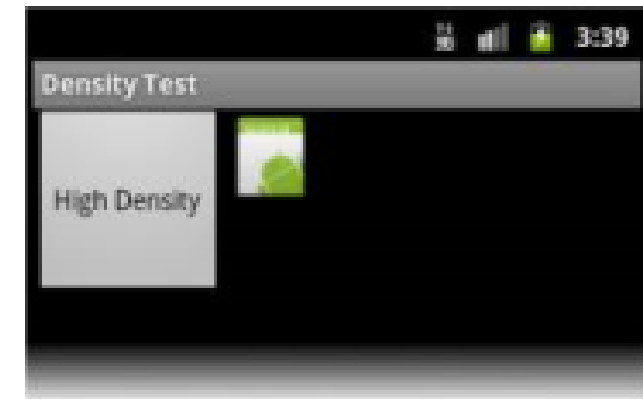
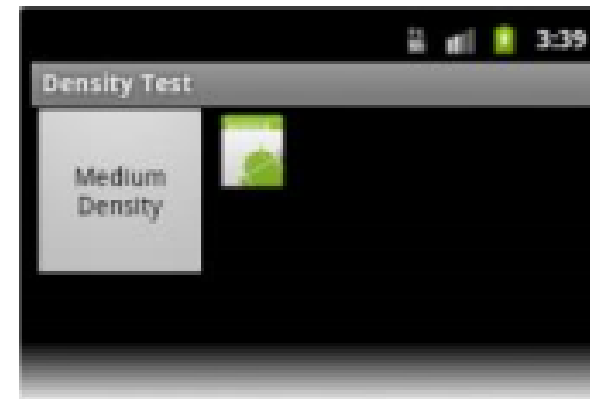
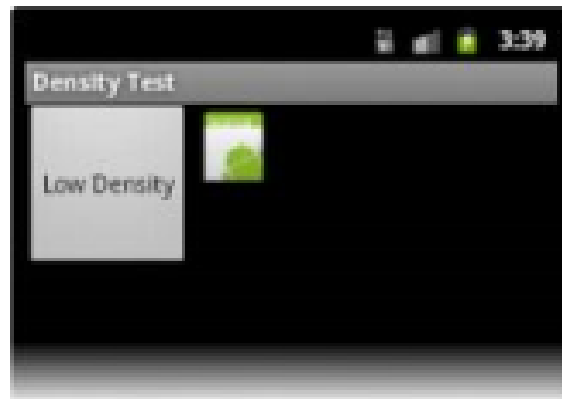
DENSITÉ INDÉPENDANTE/ DÉPENDANTE

Les deux figures suivantes montrent l'intérêt de l'indépendance de la densité :

Dépendance de la densité (px)

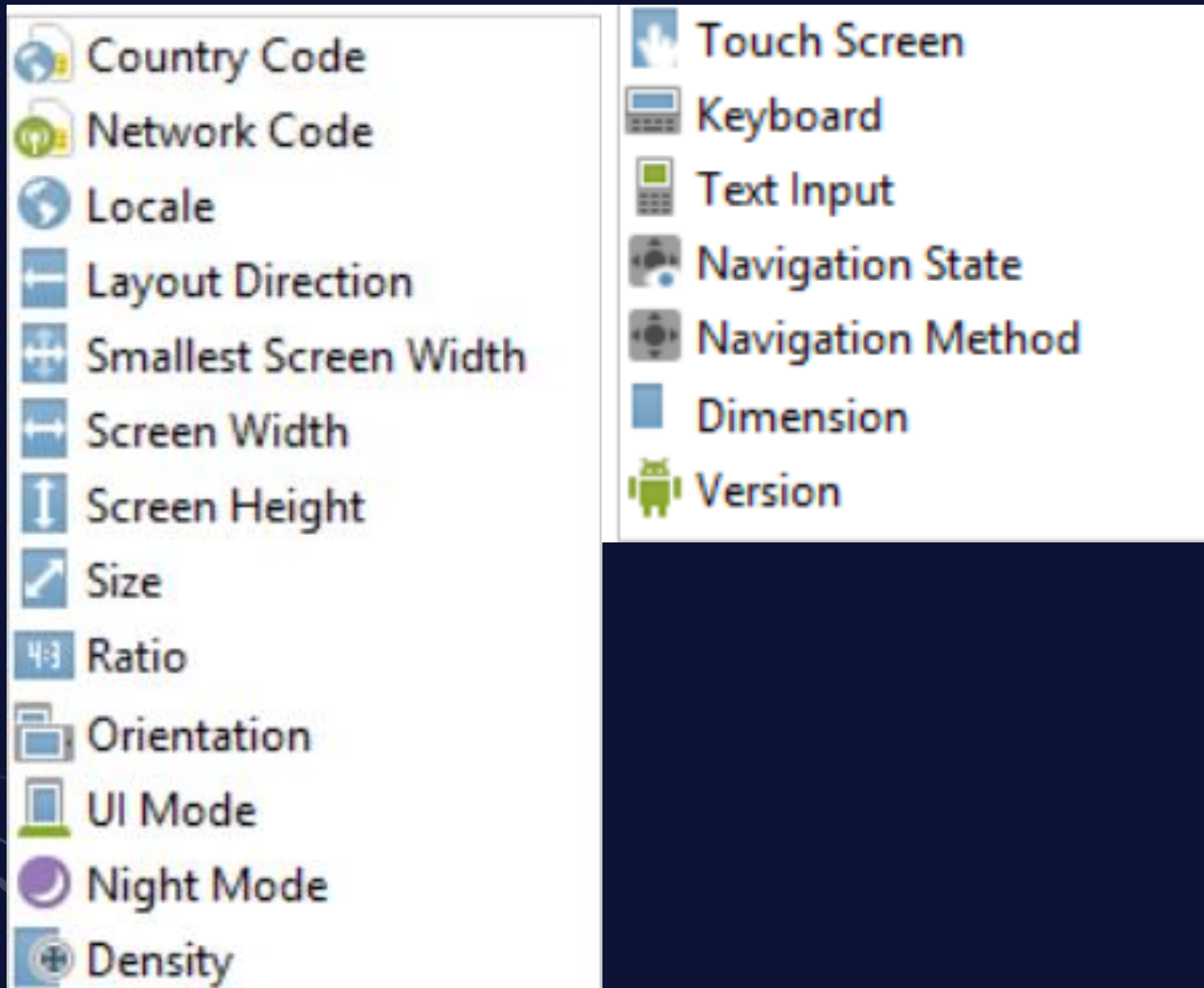


Indépendance de la densité (dp)



PRISE EN CHARGE DES DIFFÉRENTES CONFIGURATIONS

Il est possible de personnaliser les ressources, en fonction de :



PRISE EN CHARGE DES DIFFÉRENTES CONFIGURATIONS

En utilisant des qualificateurs : par exemple :

- Par défaut : **layout** : layouts par défaut
- Orientation : **layout-land** : versions paysage des layouts
- Densité : **mipmap-xxhdpi** : icônes pour des écrans de densité xxhdpi
- Taille : **layout-large** : layouts pour des écrans de taille large
- Langue : **values-fr/strings.xml** : chaînes de caractères en français

Chacun des qualificateurs est indépendant et peut être utilisé seul ou combiné à d'autres.

PRISE EN CHARGE DES DIFFÉRENTES CONFIGURATIONS

Pratiques



DRAWABLES ET DENSITÉ DE L'ÉCRAN

- Les images matricielles sont toujours stockées dans `res/drawable`,
- Une image doit être affichée sur tous les différents écrans.
- Il faut donc, créer plusieurs variantes (plusieurs tailles) de l'image selon la densité de l'écran

DRAWABLES ET DENSITÉ DE L'ÉCRAN

Images matricielles:

- **drawable-ldpi/logo.png 36 x 36 (0.75x)**
- **drawable-mdpi/logo.png 48 x 48 (1.0x baseline)**
- **drawable-hdpi/logo.png 72 x 72 (1.5x)**
- **drawable-xhdpi/logo.png 96 x 96 (2.0x)**
- **drawable-xxhdpi/logo.png 144 x 144 (3.0x)**
- **drawable-xxxhdpi/logo.png 192 x 192 (4.0x)**

DRAWABLES ET DENSITÉ DE L'ÉCRAN

- Icônes vectorielles : (**Material** ou **SVG**)
 - drawable-anydpi/graphic.xml
- Utilisation : res/drawable > New > Vector Asset

LAYOUTS : RELATIVELAYOUT

Aligner par rapport une autre vue ("`@+[package:]type:name`")

- android: `layout_above`, android: `layout_below`
- android: `layout_alignStart`, android: `layout_alignEnd`
- android: `layout_alignTop`, android: `layout_alignBottom`
- android: `layout_alignLeft`, android: `layout_alignRight`

LAYOUTS : RELATIVELAYOUT

Aligner par rapport au parent ("true" ou "false")

- android: `layout_alignParentStart`, android: `layout_alignParentEnd`
- android: `layout_alignParentTop`, android: `layout_alignParentBottom`
- android: `layout_alignParentLeft`, android: `layout_alignParentRight`
- android: `layout_alignWithParentIfMissing`

LAYOUTS : RELATIVELAYOUT

Centrer par rapport au parent ("true" ou "false")

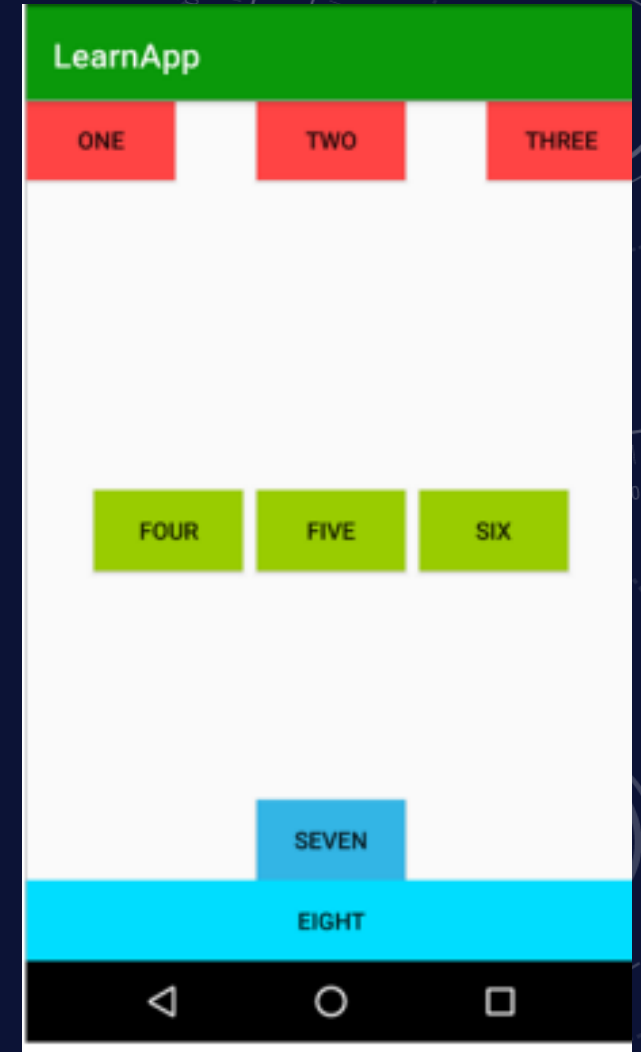
- android: **layout_centerHorizontal**
- android: **layout_centerVertical**
- android: **layout_centerInParent**

Positionner par rapport à une autre vue ("@[+][package:]type:name")

- android: **layout_toStartOf**, android: **layout_toEndOf**
- android: **layout_toLeftOf**, android: **layout_toRightOf**

TRAVAUX PRATIQUES

- Créer une activité ComplexActivity
- Créer 8 Boutons
- Aligner les boutons dans un RelativeLayout



LIENS UTILES

Les étudiants peuvent consulter ces références pour approfondir leurs connaissances :

- ✓ Cycle de vie d'une activité (1) : <https://openclassrooms.com/courses/creez-des-applications-pourandroid/preambule-quelques-concepts-avances#r-2032203>
- ✓ Cycle de vie d'une activité (2) :
<https://www.youtube.com/watch?v=UJN3AL4tiqw> Types de strings :
<http://mathias-seguy.developpez.com/tutoriels/android/utiliser-ressources#LII-A>
- ✓ Echelles d'une image : <https://robots.thoughtbot.com/android-imageview-scaletype-a-visual-guide>