

Par NIDHAL JELASSI  
jelassi.nidhal@gmail.com

DÉVELOPPEMENT MOBILE

---

PROG. ANDROID



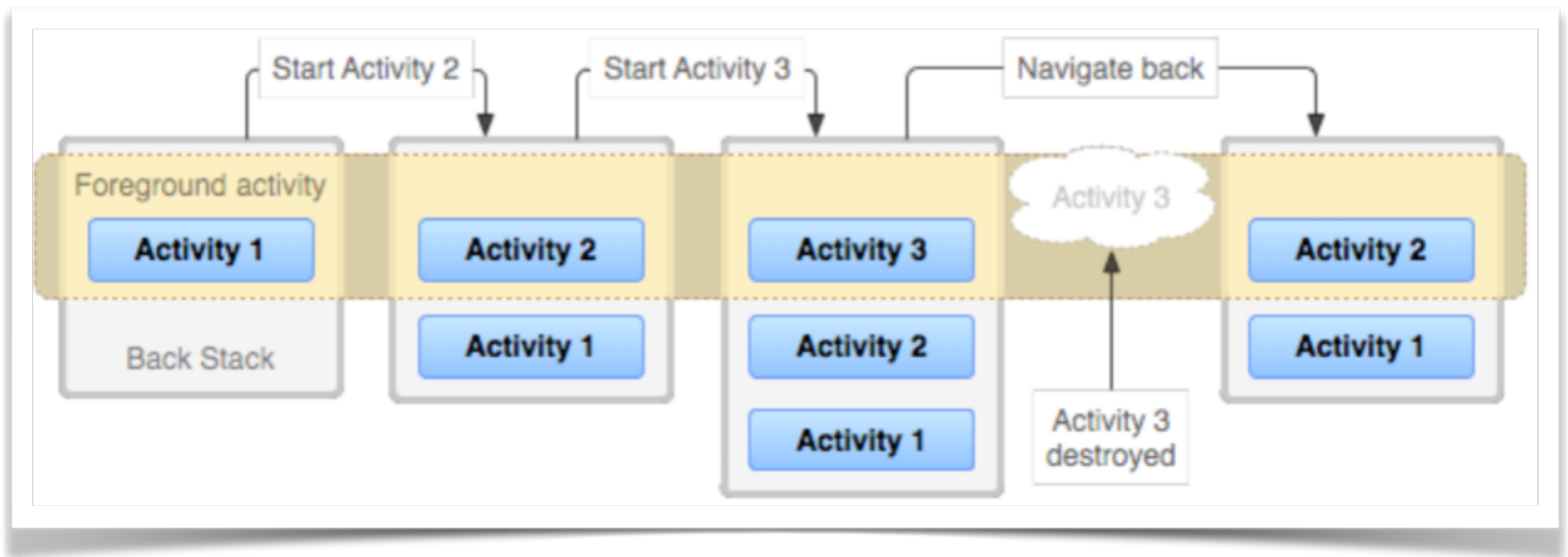
Chapitre 2 : Activités et Layouts

## CYCLE DE VIE D'UNE APPLICATION

- ▶ Les composants (vue) d'une application ont un cycle de vie :
  - ▶ *Un début* : Lors de l'instanciation en réponse aux Intents.
  - ▶ *Une fin* : Lors de la destruction des instances.
- ▶ Entre le début et la fin, les composants passent par les états suivants :
  - ▶ *Actifs ou inactifs*
  - ▶ *Visibles ou invisibles*

# CYCLE DE VIE D'UNE APPLICATION

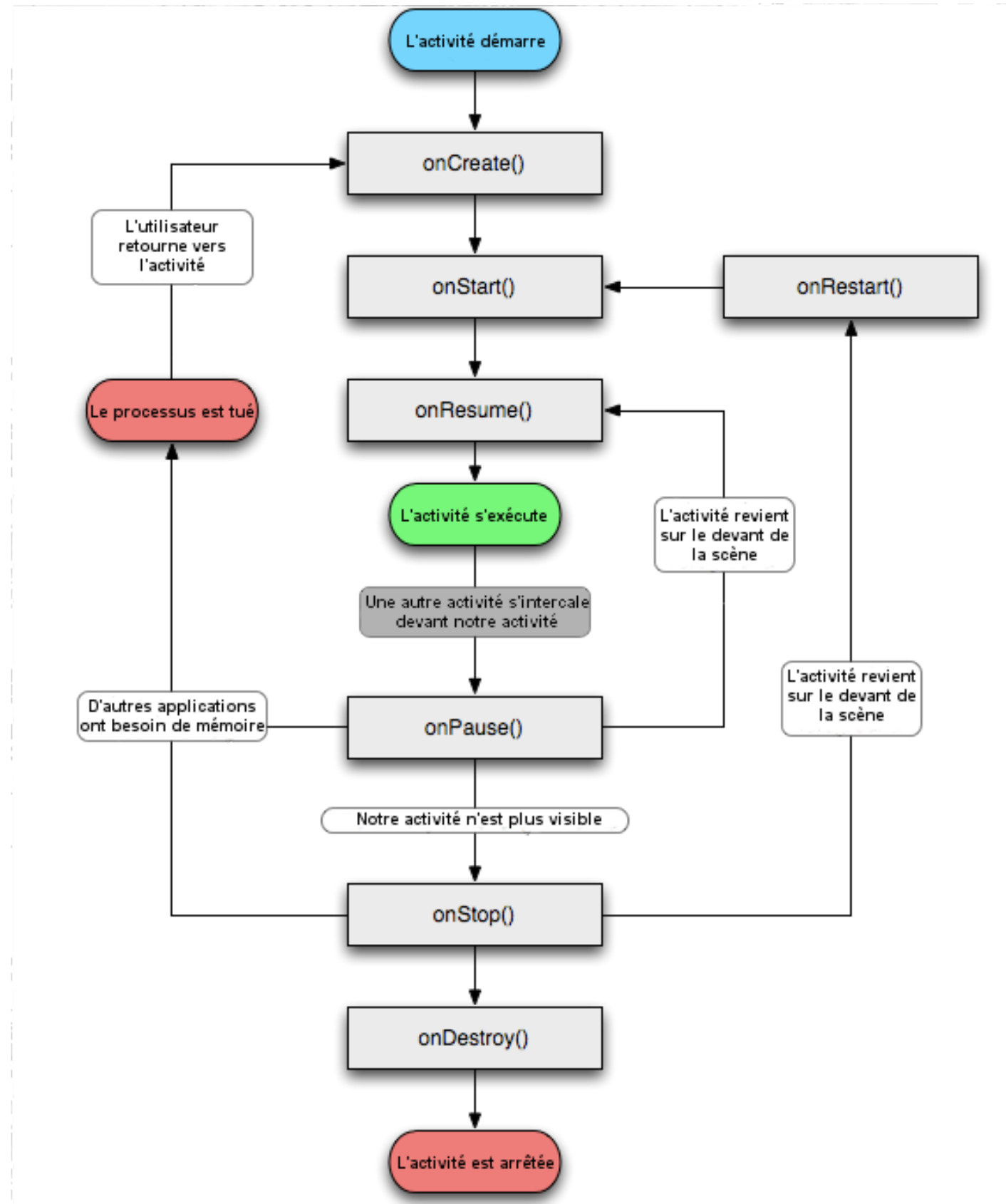
- ▶ Les activités d'une application sont gérées sous forme de **Pile**.



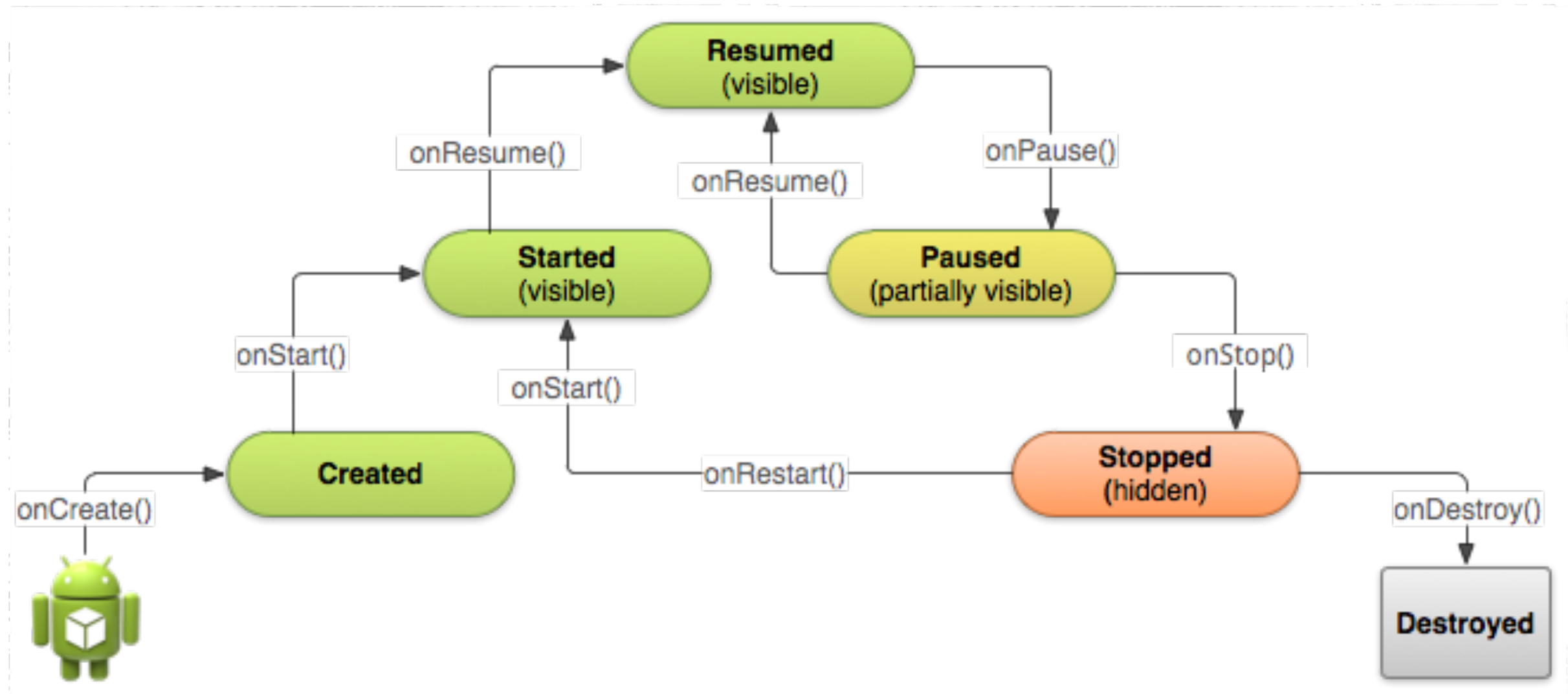
## CYCLE DE VIE D'UNE APPLICATION

- ▶ Au lancement d'une nouvelle activité, elle est placée à la tête de la pile. Elle est alors **l'activité en exécution**.
- ▶ L'activité précédente ne revient en **tête de la pile** que si la nouvelle activité est fermée.
- ▶ Sur le terminal, en cliquant sur le bouton **Retour**, l'activité suivante dans la pile devient active.
- ▶ Il n'existe pas de **méthode main** dans un programme Android
- ▶ Android exécute le code d'une activité en appelant des **callbacks** qui correspondent aux phases de la vie d'une activité
- ▶ Il n'est **pas nécessaire** d'implémenter toutes les callbacks.

## CYCLE DE VIE



# ETATS D'UNE ACTIVITÉ



## EVÉNEMENTS LIÉS

- ▶ Au passage d'une activité d'un état à un autre, Android appelle les méthodes de transition correspondantes :

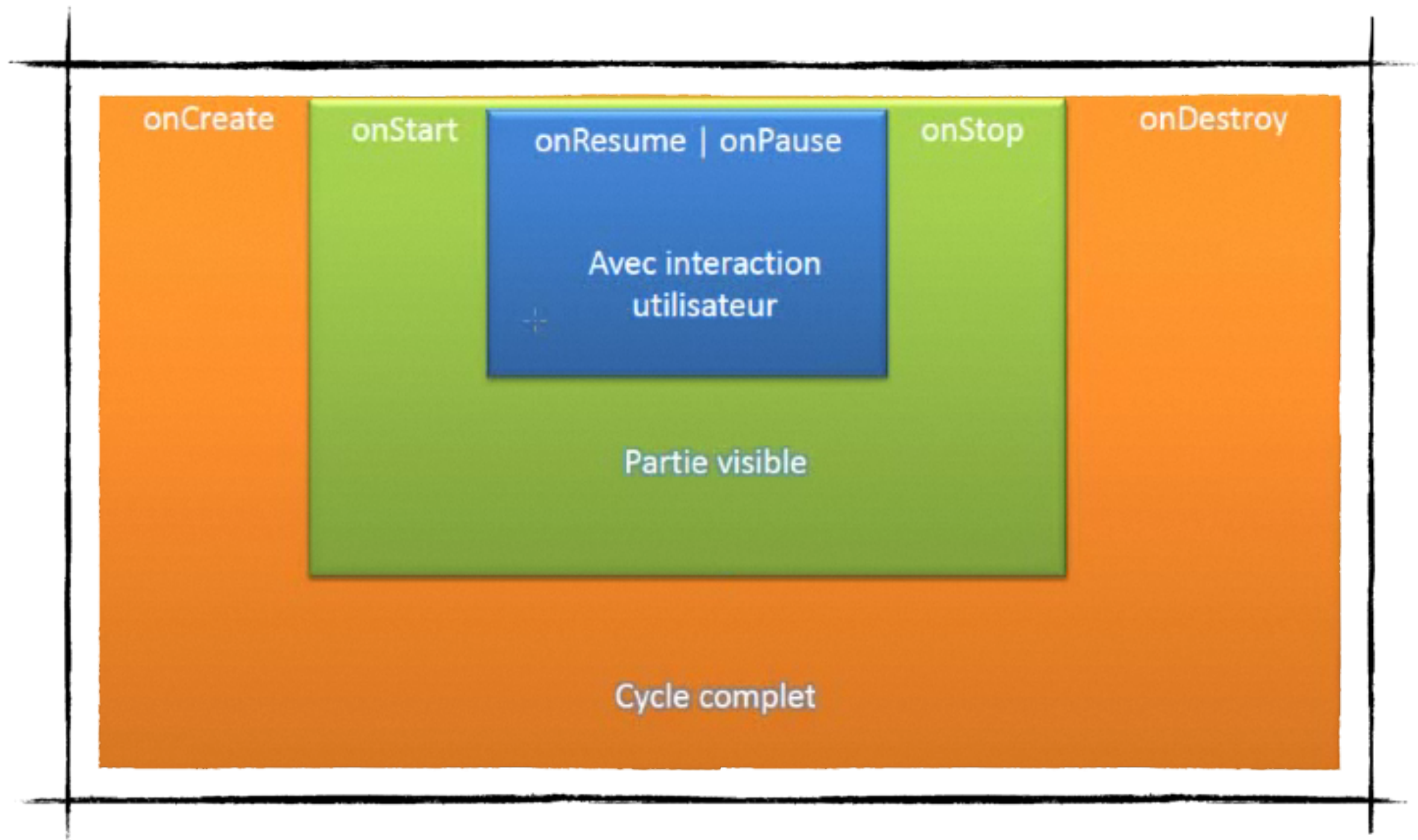
Callbacks

```
public class Main extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.acceuil); }  
    protected void onDestroy() {  
        super.onDestroy(); }  
    protected void onPause() {  
        super.onPause(); }  
    protected void onResume() {  
        super.onResume(); }  
    protected void onStart() {  
        super.onStart(); }  
    protected void onStop() {  
        super.onStop(); } }  
}
```

Obligatoire

Recommandé

# CALLBACKS





# CALLBACKS

## VOID ONCREATE(BUNDLE SAVEDINSTANCESTATE)

- ▶ Invoquée à la création d'une activité
- ▶ Initialisation de tous void onCreate(Bundle savedInstanceState) les éléments
- ▶ Le Bundle savedInstanceState contient l'état précédent de l'activité.

## VOID ONSTART()

- ▶ Invoquée juste avant que l'activité ne devienne visible.

## VOID ONRESUME()

- ▶ Invoquée quand l'activité commence à interagir avec l'utilisateur.
- ▶ L'activité est alors en tête de pile.

## CALLBACKS

### VOID ONRESTART()

- ▶ Invoquée quand l'activité est stoppée et ensuite démarré une nouvelle fois.
- ▶ C'est un état de transition.

### VOID ONPAUSE()

- ▶ Invoquée quand le système va démarrer une autre activité.
- ▶ Utilisée généralement pour sauvegarder des changements ou des données. Arrête tout ce qui consomme de la mémoire (ex: animation, connexion DB).
- ▶ Son traitement doit être léger car l'activité suivante ne démarre qu'une fois que tout est fini.

## CALLBACKS

### VOID ONSTOP()

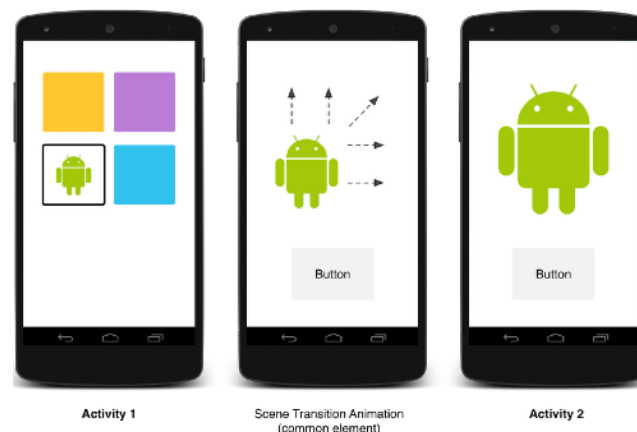
- ▶ Invoquée quand l'activité n'est plus visible. Elle est alors totalement cachée et ne peut plus exécutée de code
- ▶ N'empêche pas le système de tuer l'activité

### VOID ONDESTROY()

- ▶ Invoquée quand l'activité est détruite. C'est l'état ultime d'une activité.
- ▶ Deux scénarios possibles :
- ▶ l'activité est en cours de finition.
- ▶ le système détruit temporairement cette instance de l'activité pour économiser de l'espace. Dans ce cas, utiliser onPause() ou onStop() pour sauvegarder vos données.

# UNE ACTIVITÉ

- ▶ C'est un **composant** d'application.
- ▶ Fournit un écran avec lequel les utilisateurs peuvent **interagir** avec l'application et ses différentes fonctionnalités.
- ▶ Chaque **Activité** est associée à une fenêtre qui représente **l'interface** utilisateur.
- ▶ Une application = Enchaînement des activités



## UNE ACTIVITÉ

- ▶ Pour pouvoir être lancée, toute activité doit être préalablement déclaré dans le **AndroidManifest**.
- ▶ Une des activités de l'application doit être désigné comme **l'activité initiale** de l'application (toujours dans AndroidManifest)
- ▶ Lancer une activité simple : Méthode `startActivity(...)`
- ▶ Lancer une activité en vue d'obtenir un résultat en retour (on parle alors de "sous-activité") : Methode `startActivityForResult(...)`

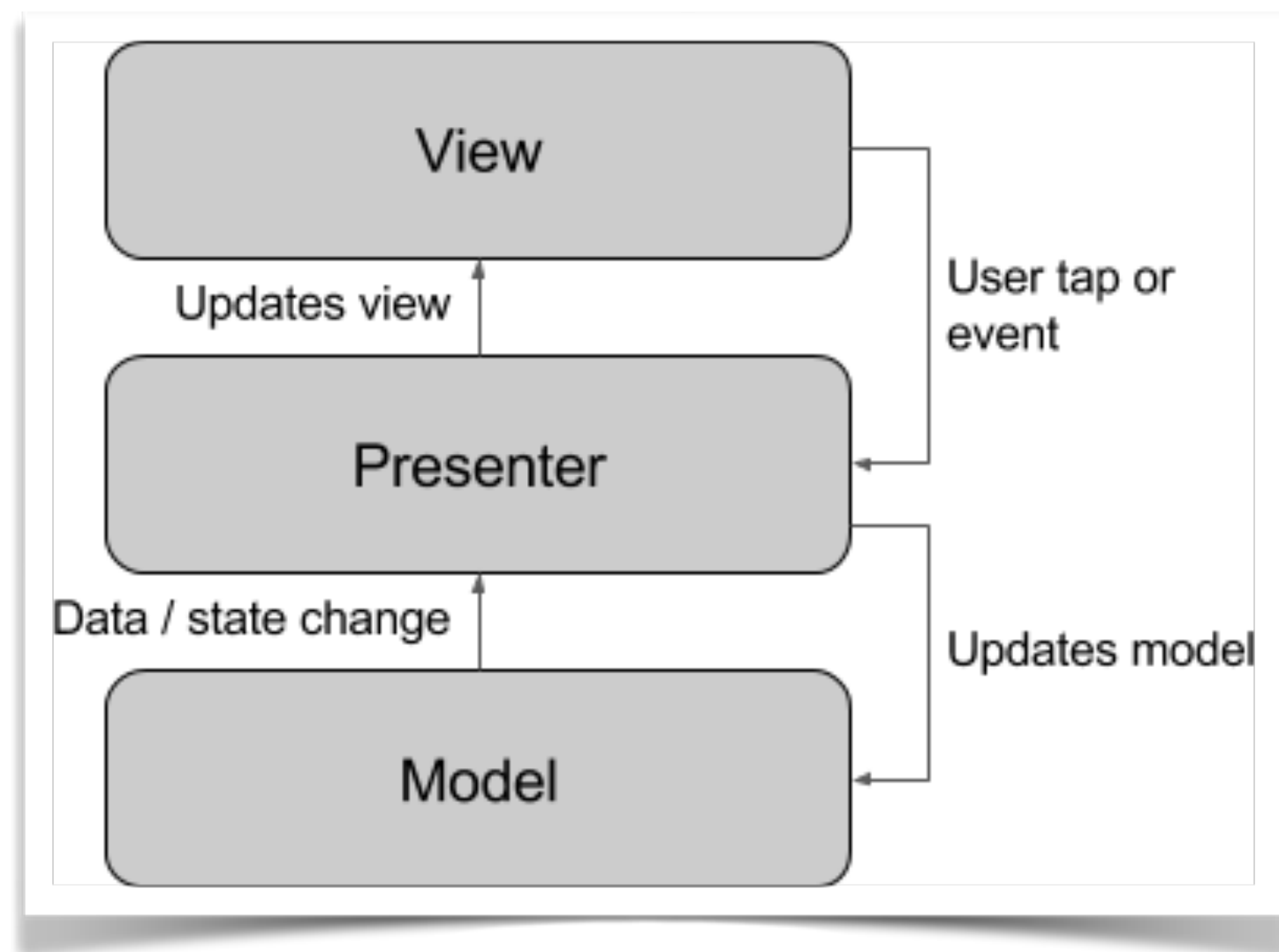
# UNE ACTIVITÉ

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        TextView tx = (TextView) this.findViewById(R.id.txtv);  
        tx.setText("Cours Prog Mobile - ISIE 1 - Nidhal");  
    }  
}  
  
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.example.nidhal.firstapp">  
  
    <application  
        android:allowBackup="true"  
        android:icon="@mipmap/ic_launcher"  
        android:label="FirstApp"  
        android:supportRtl="true"  
        android:theme="@style/AppTheme">  
        <activity android:name=".MainActivity">  
            <intent-filter>  
                <action android:name="android.intent.action.MAIN" />  
  
                <category android:name="android.intent.category.LAUNCHER" />  
            </intent-filter>  
        </activity>  
    </application>  
</manifest>
```



## LE MODÈLE MVP

- ▶ La relation entre l'**activité** et son **layout** se fait selon le modèle MVP (Model-View-Presenter).



## LE MODÈLE MVP

- ▶ Les **Views** sont des éléments d'interface utilisateur qui affichent des données et répondent aux actions de l'utilisateur. Chaque élément de l'écran est une vue.
- ▶ Les **Presenters** connectent les vues de l'application au modèle. Ils fournissent les vues avec les données spécifiées par le modèle, ainsi que le modèle avec les entrées utilisateur à partir de la vue.
- ▶ Le **Model** spécifie la structure des données de l'application et le code permettant d'accéder aux données et de les manipuler.



## VIEW ET VIEWGROUP

- ▶ Une **vue** est une classe qui étend **View**.
- ▶ Un groupe de vue étend **ViewGroup** (et donc View) et peut contenir d'autres vues.
- ▶ Un groupe de vues organise l'affichage des vues qu'il contient
- ▶ La méthode `setContentView` (de Activity) sert à préciser la vue à afficher.

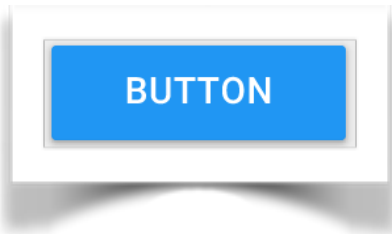
# VIEW ET VIEWGROUP

Types de vues :

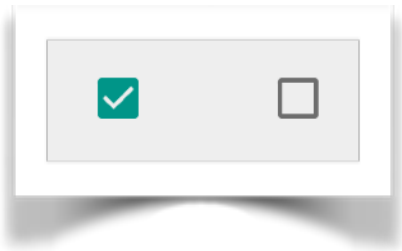
- ▶ *TextView, EditText.*
- ▶ *Buttons (Button class), menus, etc...*
- ▶ *Scrollable (ScrollView, RecyclerView)*
- ▶ *ImageView*
- ▶ *Group views (ConstraintLayout and LinearLayout)*

# EXAMPLES

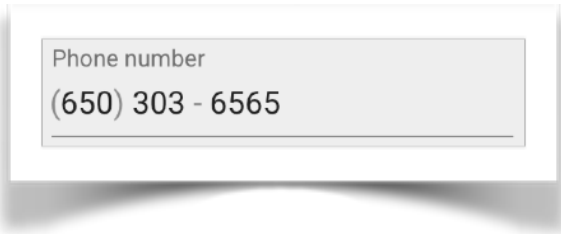
Button



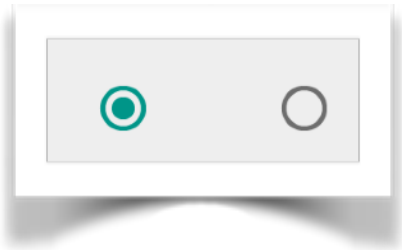
CheckBox



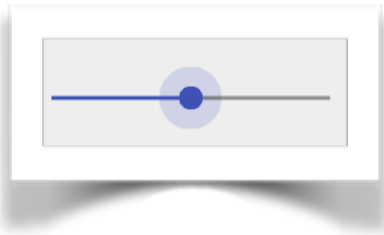
EditText



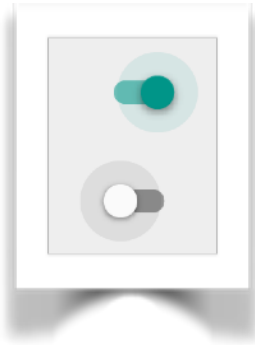
RadioButton



Slider



Switch



# ATTRIBUTS

- ▶ Couleur
- ▶ Dimensions,
- ▶ Positionnement
- ▶ Avoir (ou non) le focus (quand on a besoin de l'intervention de l'utilisateur)
- ▶ L'interactivité (répondre aux clicks des users par exp)
- ▶ La visibilité
- ▶ Eventuellement la relation avec les autres views

## ATTRIBUTS : EXEMPLE

```
<TextView  
    android:id="@+id/show_count"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:background="@color/myBackgroundColor"  
    android:text="@string/count_initial_value"  
    android:textColor="@color/colorPrimary"  
    android:textSize="@dimen/count_text_size"  
    android:textStyle="bold"  
  
/>
```

# SYNTAXE

**android:<property\_name>="@+id/view\_id"**

**Example:** android:id=« @+id/show\_count"

**android:<property\_name>="<property\_value>"**

**Example:** android:layout\_width="match\_parent"

**android:<property\_name>="@<resource\_type>/resource\_id"**

**Example:** android:text="@string/button\_label\_next"

# LAYOUTS

# LAYOUTS

- ▶ Les **layouts** sont des **Views** (LinearLayout, RelativeLayout, etc.) qui héritent de **ViewGroup**.
- ▶ Ils permettent de gérer le placement de Views filles à l'intérieur du ViewGroup.
- ▶ Chaque layout permet d'associer à une View (fille du Layout) un ensemble de contraintes de placement.
- ▶ Un layout peut également être défini comme un **arbre de vues statique** doit être définie en XML dans le **répertoire ressource layout**.



## TYPES DE LAYOUTS

- ▶ **LinearLayout**: dispose les éléments de gauche à droite ou du haut vers le bas
- ▶ **RelativeLayout**: les éléments sont placés relativement les uns par rapport aux autres
- ▶ **TableLayout**: disposition matricielle
- ▶ **FrameLayout**: disposition en haut à gauche en empilant les éléments (un seul visible à la fois). Les vues sont stockées dans une pile, La taille de FrameLayout est la taille de sa plus grande vue enfant.

## TYPES DE LAYOUTS

- ▶ **GridLayout** (deprecated) : disposition des composants sur une grille.
- ▶ **ConstraintLayout**: groupe de vues utilisant des points d'ancrage, des arêtes et des instructions pour contrôler le positionnement des vues par rapport aux autres éléments de la présentation.
- ▶ **AbsoluteLayout**: un groupe qui vous permet de spécifier les emplacements exacts (coordonnées x / y) de ses vues enfants. Le moins souple.
- ▶ Les déclarations se font principalement en XML, ce qui évite de passer par les instantiations Java.

# EXEMPLES



LinearLayout



RelativeLayout



ConstraintLayout



TableLayout

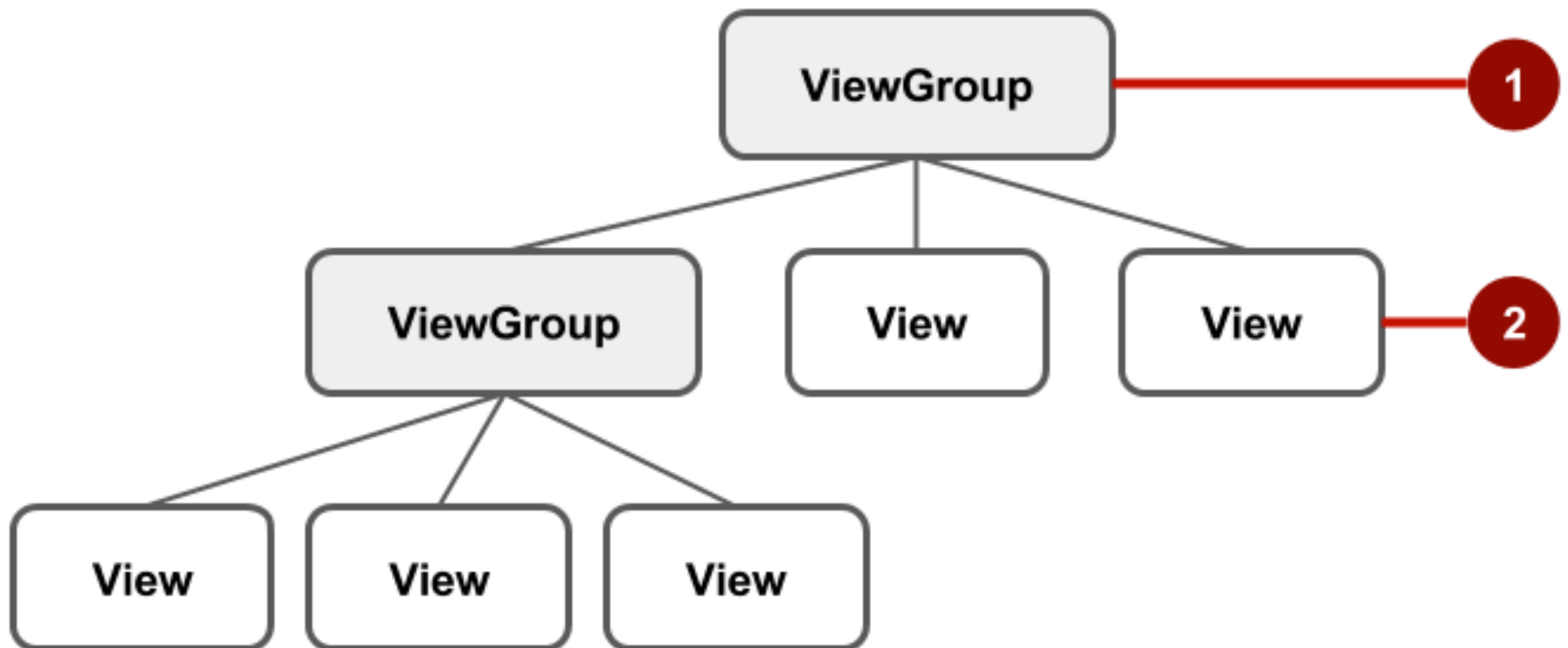


GridLayout

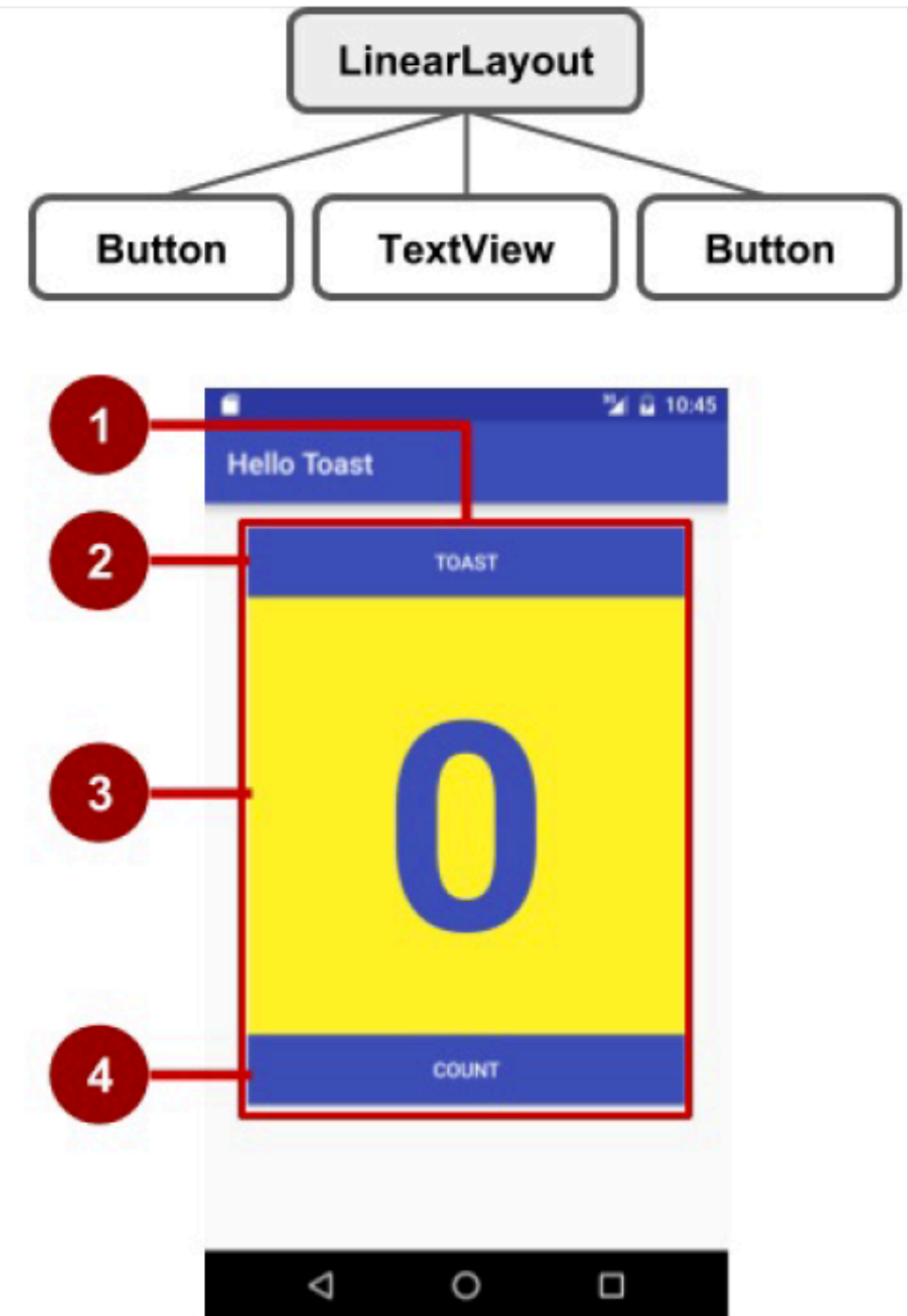
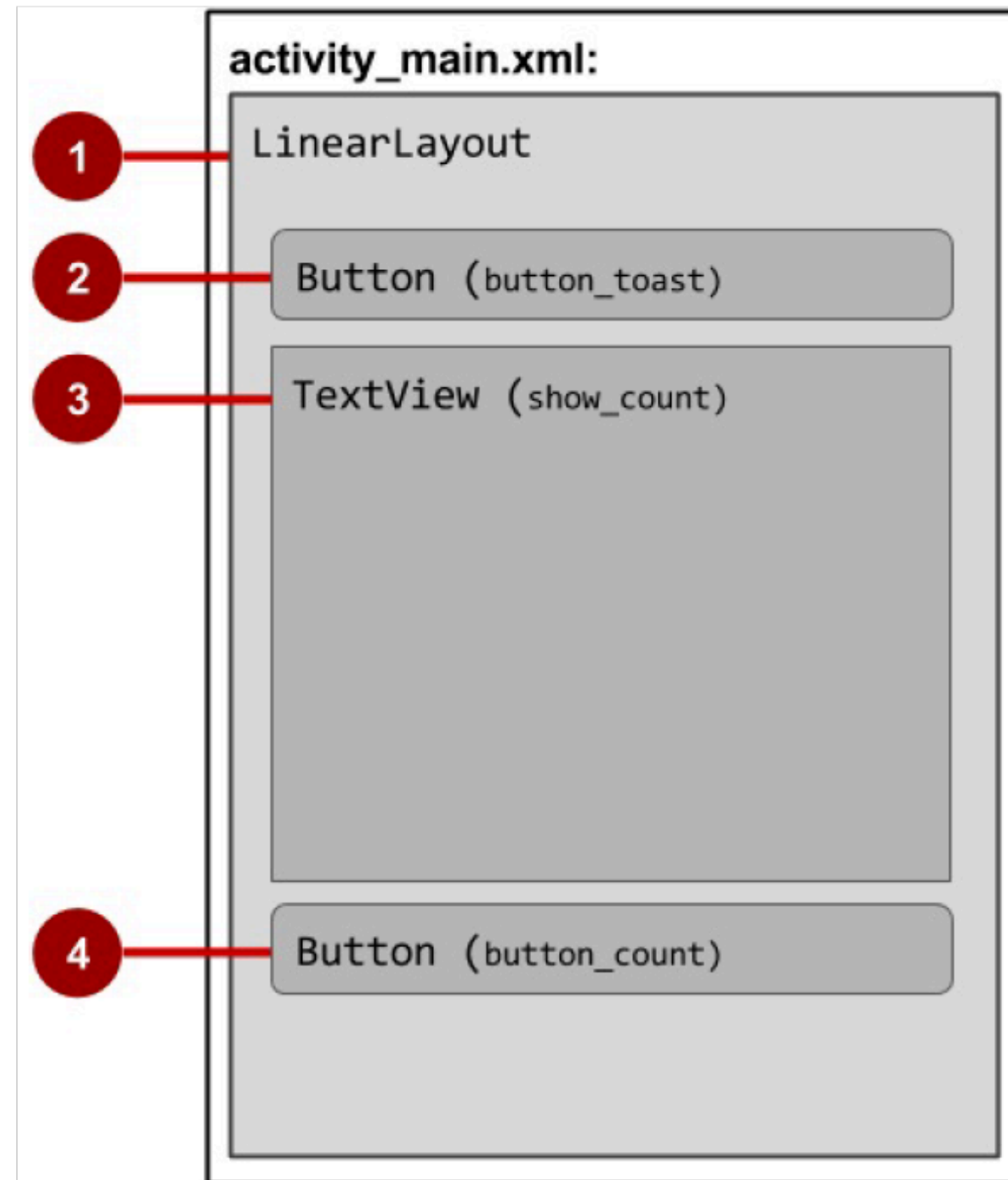
## RELATIVE LAYOUT

- ▶ L'un des layouts les plus utilisés est `RelativeLayout`. Il permet de positionner les éléments graphiques (views) relativement l'un à l'autre.
- ▶ Pour y faire, le SDK d'Android offre plusieurs propriétés permettant de positionner ces Views telle que :
- ▶ *`android:layout_toLeftOf`*: Positionne le bord droit de cette vue à gauche d'une autre vue.
- ▶ *`android:layout_toRightOf`*: Positionne le bord gauche de cette vue à droite d'une autre vue.
- ▶ *`android:layout_centerHorizontal`*: Centre cette vue horizontalement dans son parent.
- ▶ *`android:layout_centerVertical`*: Centre cette vue verticalement dans son parent..
- ▶ *`android:layout_alignParentTop`*: Positionne le bord supérieur de cette vue pour correspondre au bord supérieur du parent.
- ▶ *`android:layout_alignParentBottom`*: Positionne le bord inférieur de cette vue pour correspondre au bord inférieur du parent.

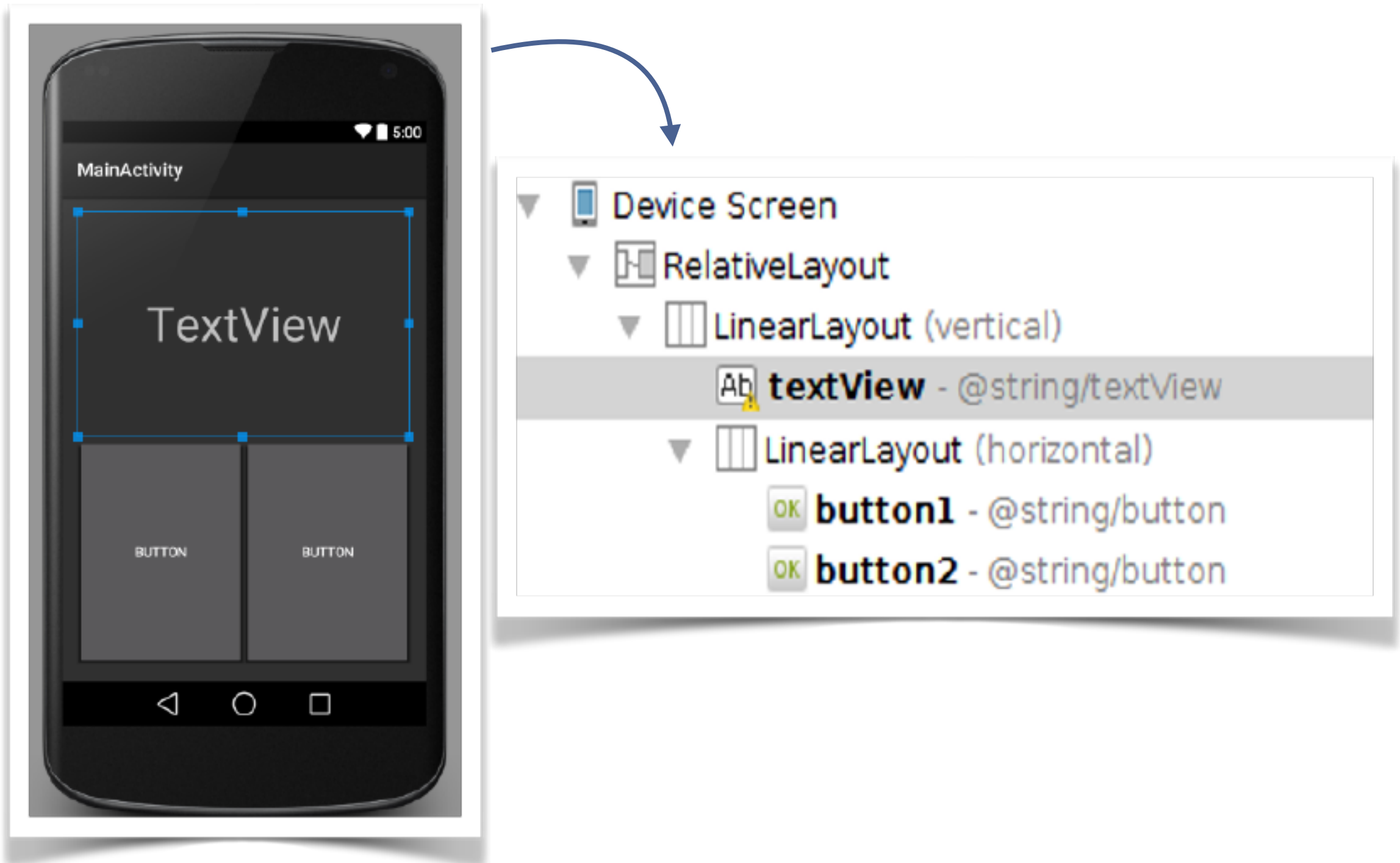
# COMBINER DES LAYOUTS



## EXAMPLE

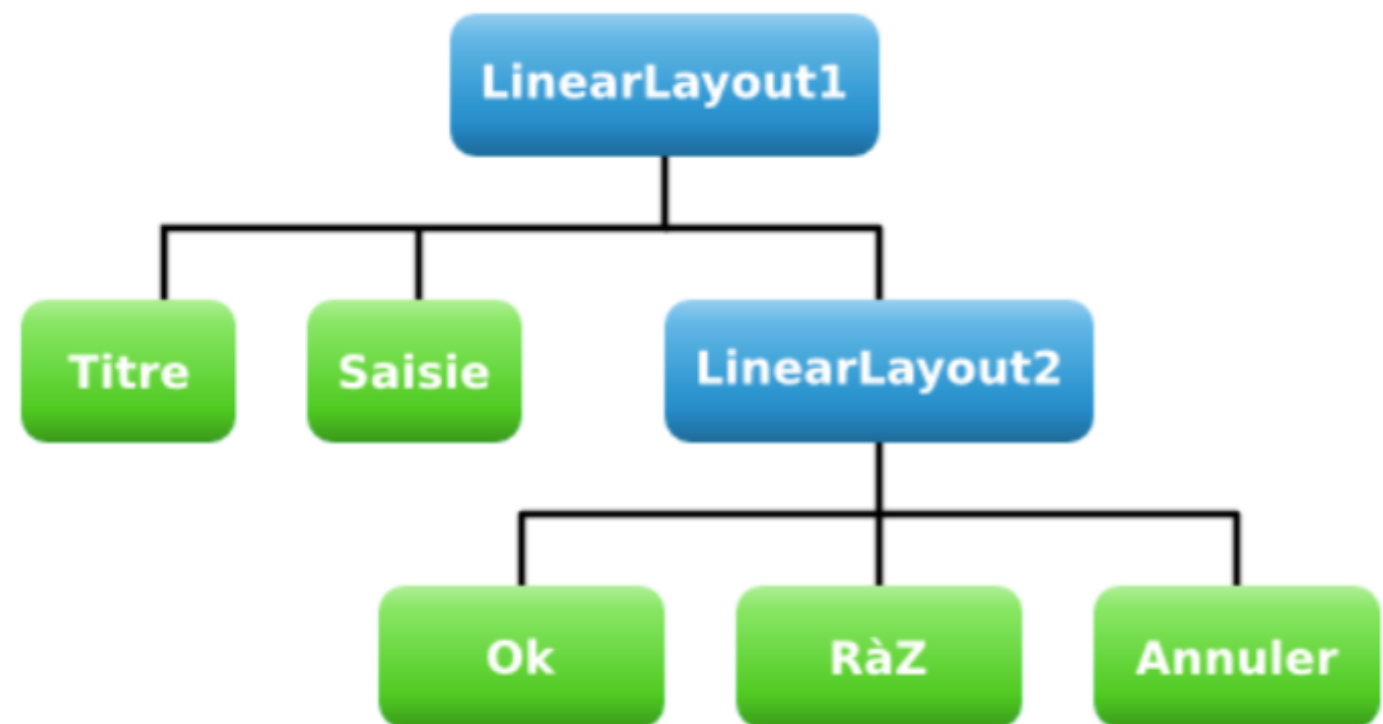
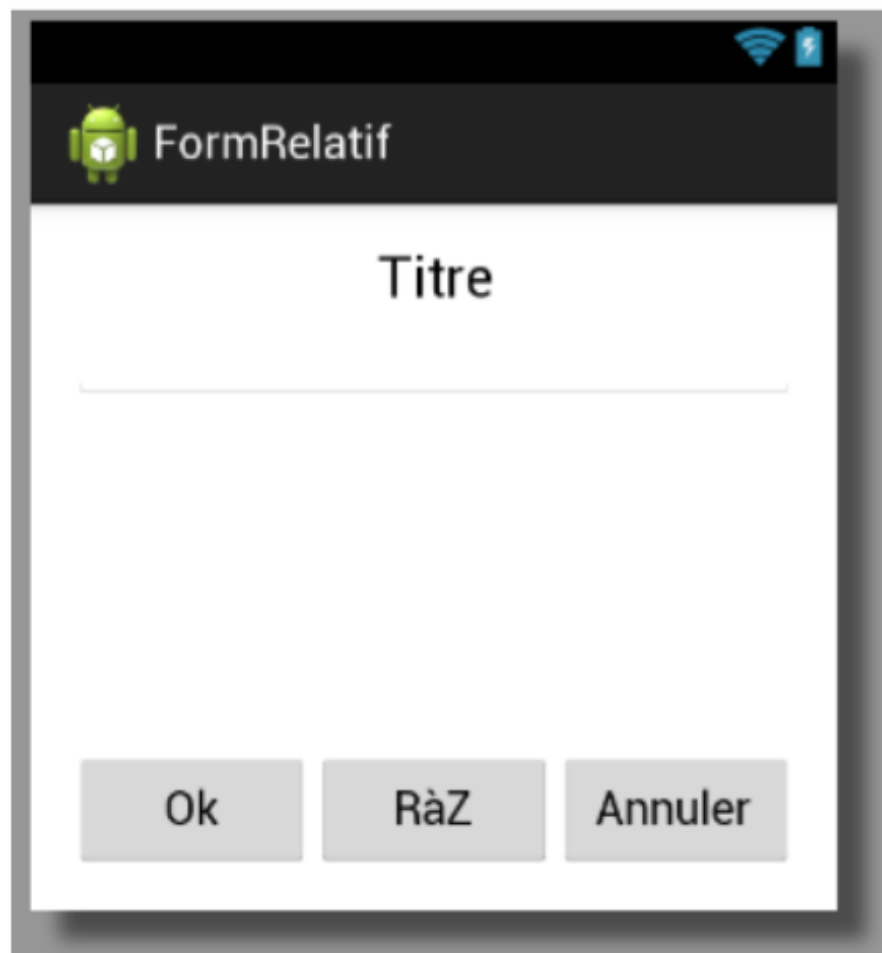


# HIERARCHIE DES VUES



# HIERARCHIE DES VUES

- ▶ Les groupes et vues forment un arbre.





## HIERARCHIE DES VUES

- ▶ En XML, cet arbre donne ça :

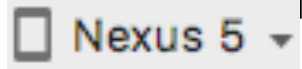
```
<LinearLayout android:id="@+id/groupe1" ...>
  <TextView android:id="@+id/titre" .../>
  <EditText android:id="@+id/saisie" .../>
  <LinearLayout android:id="@+id/groupe2" ...>
    <Button android:id="@+id/ok" .../>
    <Button android:id="@+id/raz" .../>
    <Button android:id="@+id/annuler" .../>
  </LinearLayout>
</LinearLayout>
```

# LAYOUT

- ▶ Toutes les vues doivent spécifier ces deux attributs :
- ▶ `android:layout_width` : représente la largeur de la vue
- ▶ `android:layout_height` : représente la hauteur de la vue
- ▶ Ils peuvent valoir :
- ▶ `wrap_content` : la vue est la plus petite possible
- ▶ `match_parent` (anciennement `fill_parent`) : la vue est la plus grande possible
- ▶ `valeurdp` : une taille fixe, ex : "100dp"

## APERÇU DESIGN



- ▶ Pour éditer un design particulier en mode paysage (landscape) :
- ▶ Cliquer sur « Orientation »
- ▶ Cliquer sur » Switch to Landscape or Switch to Portrait »  

- ▶ Pour prévisualiser votre design sur différents terminaux :
- ▶ Cliquer sur « Device »
- ▶ Choisir le terminal désiré



[JELASSI.NIDHAL@GMAIL.COM](mailto:JELASSI.NIDHAL@GMAIL.COM)

---

**NIDHAL JELASSI**