TextView dans Android Studio

TextView est une vue utilisée pour afficher du texte à l'écran dans une interface Android.

- Attributs de base :

- `text` : contenu textuel affiché
- `textSize` : taille du texte
- `textColor` : couleur du texte
- `textStyle` : style du texte (bold, italic)

- Exemple XML :

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bonjour Yassine !"
    android:textSize="18sp"
    android:textColor="#FF5722"
    android:textStyle="bold" />
```

Méthodes Java pour TextView

Assurez-vous que le TextView est bien référencé via son ID dans le layout XML.

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

- Manipulation en Java :

Voici comment récupérer et modifier le texte et la couleur d'un TextView en Java.

- Récupérer le texte :

```
TextView tv = findViewById(R.id.monTextView);
String texte = tv.getText().toString();
```

- Modifier le texte :

```
tv.setText("Nouveau texte");
```

Modifier la couleur du texte :

```
tv.setTextColor(Color.parseColor("#FF5722"));
```

EditText dans Android Studio

- EditText est une vue permettant à l'utilisateur de saisir du texte dans une interface Android.

  Attributs XML de base :
  - `hint` : texte d'indication affiché quand le champ est vide
  - `text` : texte initial affiché
  - `textSize` : taille du texte
  - `textColor` : couleur du texte
  - `inputType` : type de saisie (texte, nombre, email, etc.)

- Exemple XML :

```
<EditText
    android:id="@+id/monEditText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Entrez votre nom"
    android:textSize="16sp"
    android:textColor="#000000"
    android:inputType="nombre" />
```

Méthodes Java pour EditText

Assurez-vous que l'EditText est bien référencé via son ID dans le layout XML.

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

- Manipulation en Java :
  Voici comment récupérer et modifier le texte et la couleur d'un EditText en Java.

- Récupérer le texte :

```
EditText et = findViewById(R.id.monEditText);
String texte = et.getText().toString();
```

- Modifier le texte :

```
et.setText("Nouveau texte");
```

- Modifier la couleur du texte :

```
et.setTextColor(Color.parseColor("#3F51B5"));
```

Button dans Android Studio

Button est une vue qui permet à l'utilisateur de déclencher une action en cliquant dessus.

- **Attributs XML de base :**

- `text` : texte affiché sur le bouton
- `textSize` : taille du texte
- `textColor` : couleur du texte
- `background` : couleur ou image de fond
- `onClick` : nom de la méthode appelée lors du clic

- **Exemple XML :**

```
<Button
    android:id="@+id/buttonValider"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Valider"
    android:textSize="16sp"
    android:textColor="#FFFFFF"
    android:background="#3F51B5"
    android:onClick="validerFormulaire" />
```

Méthodes Java de base pour Button

- Récupérer et modifier le texte du bouton :

```
Button btn = findViewById(R.id.buttonValider);
// Récupérer le texte
String texte = btn.getText().toString();
// Modifier le texte btn.setText("Envoyer");
```

- Modifier la couleur du texte :

```
btn.setTextColor(Color.parseColor("#FF5722"));
```

- Gérer le clic sur le bouton (méthode Java) :

```
btn.setOnClickListener(new View.OnClickListener() {
    @Override public void onClick(View v) {
    // Action à effectuer lors du clic
    }
});
```

ImageView dans Android Studio

ImageView est une vue qui permet d'afficher une image dans une interface Android.

- **Attributs XML de base :**
  - `src` : ressource de l'image à afficher
  - `contentDescription` : description de l'image (accessibilité)
  - `layout_width / layout_height` : taille de la vue
  - `scaleType` : mode de redimensionnement de l'image
  - `background` : couleur ou image de fond
- **Exemple XML :**

```
<ImageView
    android:id="@+id/imageLogo"
    android:layout_width="120dp"
    android:layout_height="120dp"
    android:src="@drawable/logo_inwi"
    android:contentDescription="Logo Inwi"
    android:scaleType="centerCrop"
    android:background="#EEEEEE" />
```

Méthodes Java relatives à **ImageView**

- Récupérer et modifier le texte du bouton :

```
ImageView img = findViewById(R.id.imageLogo);
// Changer l'image affichée
img.setImageResource(R.drawable.logo_inwi);
// Changer la couleur de fond
img.setBackgroundColor(Color.parseColor("#EEEEEE"));
```

- Modifier la couleur du texte :

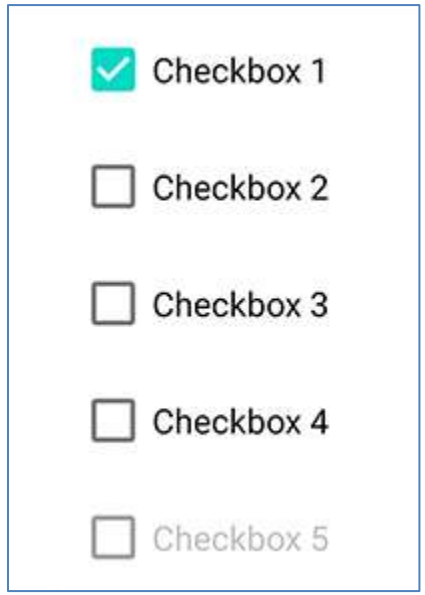
```
btn.setTextColor(Color.parseColor("#FF5722"));
```

- Gérer le clic sur le bouton (méthode Java) :

```
btn.setOnClickListener(new View.OnClickListener() {
    @Override public void onClick(View v) {
    // Action à effectuer lors du clic
    }
});
```

CheckBox dans Android Studio

CheckBox est une vue qui permet à l'utilisateur de sélectionner ou désélectionner une option (état vrai/faux).



- **Attributs XML de base :**
  - text : texte affiché à côté de la case
  - checked : état initial (cochée ou non)
  - textSize : taille du texte
  - textColor : couleur du texte
- **Exemple XML :**

```
<CheckBox
    android:id="@+id/checkboxNewsletter"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="S'abonner à la newsletter"
    android:checked="false"
    android:textSize="16sp"
    android:textColor="#333333" />
```

Méthodes Java relatives à CheckBox

- Récupérer et modifier l'état de la CheckBox :

```
CheckBox cb = findViewById(R.id.checkboxNewsletter);
// Vérifier si la case est cochée
boolean estCochee = cb.isChecked();
// Cocher ou décocher la case
cb.setChecked(true);
```

- Gérer le changement d'état (écouteur) :

```
cb.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView,boolean isChecked) {
        // Action à effectuer lors du changement d'état
    }
});
```

RadioButton dans Android Studio

RadioButton est une vue qui permet à l'utilisateur de sélectionner une seule option parmi un groupe (utilisé avec RadioGroup).

- **Attributs XML de base :**

– text : texte affiché à côté du bouton

– checked : état initial (sélectionné ou non)

– textSize : taille du texte

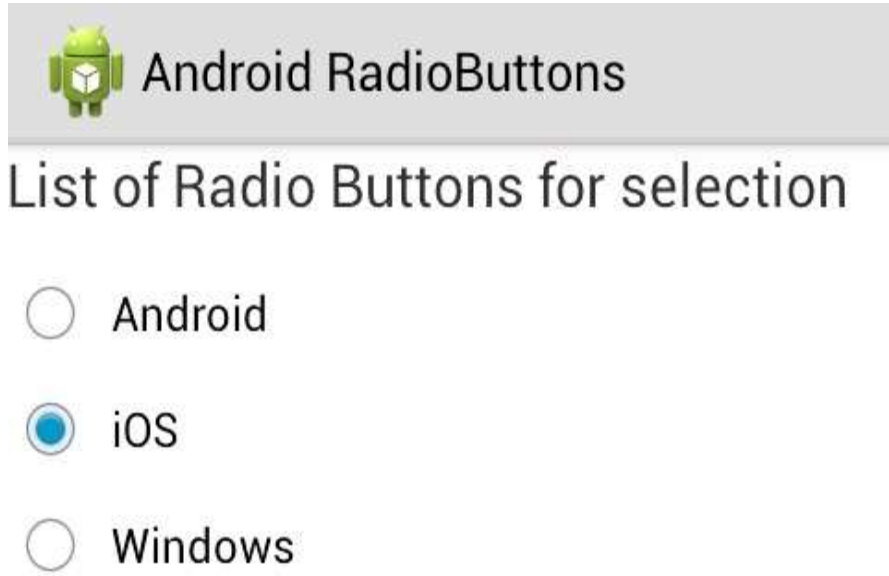
– textColor : couleur du texte

```
<RadioGroup
    android:id="@+id/radioGroupSexe"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <RadioButton
        android:id="@+id/radioHomme"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Homme"
        android:checked="true"
        android:textSize="16sp"
        android:textColor="#333333" />
    <RadioButton
        android:id="@+id/radioFemme"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Femme"
        android:checked="false"
        android:textSize="16sp"
        android:textColor="#333333" />
</RadioGroup>
```

Méthodes Java relatives à **RadioButton**

- Récupérer et modifier l'état d'un RadioButton :

```
RadioButton rbHomme = findViewById(R.id.radioHomme);
RadioButton rbFemme = findViewById(R.id.radioFemme);

// Vérifier si un bouton est sélectionné
boolean hommeSelectionne = rbHomme.isChecked();

// Sélectionner un bouton par code
rbFemme.setChecked(true);
```



- Gérer la sélection dans un RadioGroup :

```
RadioGroup rg = findViewById(R.id.radioGroupSexe);
rg.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(RadioGroup group, int checkedId) {
        if (checkedId == R.id.radioHomme) {
            // Action si "Homme" est sélectionné
        } else if (checkedId == R.id.radioFemme) {
            // Action si "Femme" est sélectionné
        }
    }
});
```

Switch dans Android Studio

- Le **Switch** est un composant interactif pour activer/désactiver une option.
- **Attributs XML de base :**
  - android:id : Identifiant unique du Switch.
  - android:text : Texte affiché à côté du Switch.
  - android:checked : État initial (true = activé, false = désactivé).
  - android:enabled : Active ou désactive l'interaction.
  - android:visibility : Contrôle la visibilité (visible, invisible, gone).


```
<Switch
    android:id="@+id/switchMode"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Activer le mode"
    android:checked="false"
    android:enabled="true"
    android:visibility="visible" />
```

Méthodes Java relatives à **Switch**

```
// Récupération du Switch par son ID
        Switch switchMode = findViewById(R.id.switchMode);

        // Écouteur pour détecter le changement d'état
        switchMode.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) {
                if (isChecked) {
                    // Action(s) si le Switch est activé
                                } else {
                    // Action (s) si le Switch est désactivé
                 }
            }
        });
```

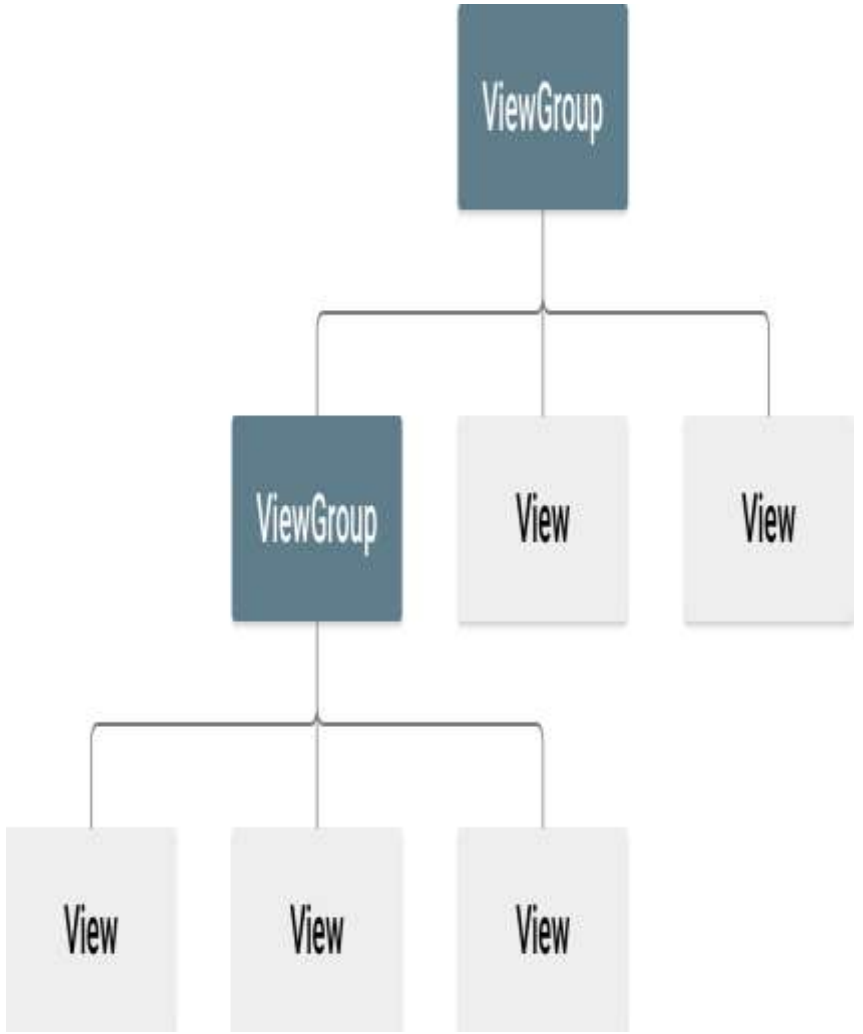
Le widget <View> en Android

- <View> est la classe de base pour tous les composants UI en Android.
- Utilisé pour dessiner des formes simples avec une taille, une couleur, une position.
- Utilisation courante :
  - Créer des rectangles colorés.
  - Ajouter des espaces vides (spacers).
- Exemple :

```
<!-- rectangle (arrière-plan rouge) -->
<View
   android:layout_width="100dp"
   android:layout_height="100dp"
   android:background="#FF0000"/>
```

Mises en page des vues android (Layout)

- Dans Android, les layouts (ViewGroups) sont essentiels pour organiser et structurer les éléments d'interface (Views) afin de créer des applications ergonomiques et adaptables
  - Views : Ce sont les éléments d'interface (widgets) comme Button, TextView, etc.
  - ViewGroups : Ce sont des conteneurs appelés layouts, qui organisent les Views. Exemples : LinearLayout, ConstraintLayout.



Déclaration des Layouts

- Deux méthodes principales :
- **Via XML**
  - Android propose un vocabulaire XML simple pour définir les Views et ViewGroups.
  - Vous pouvez utiliser l'**éditeur de mise en page** (drag & drop) pour générer le fichier XML.
- **Par code (runtime)**
  - Instanciez et configurez les Views et ViewGroups directement dans votre application.
  - Permet une manipulation dynamique et automatisée

Charger un layout XML dans une Activity

1. Android appelle onCreate() automatiquement au démarrage de l'Activity (cycle de vie).
2. Lors de l'exécution (pas la compilation), Android lit le fichier XML du layout et le transforme en une hiérarchie d'objets View (LinearLayout, TextView, Button, etc.).
3. Chaque fichier XML de mise en page est converti en objets View utilisables dans le code Java/Kotlin.
4. Ce processus s'appelle inflation : Android crée les objets et les attache à l'écran.
5. La méthode setContentView() charge le layout XML et l'associe à la fenêtre de l'Activity pour l'affichage.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

Attributs XML dans Android

- Chaque **View** et **ViewGroup** possède ses propres attributs XML.
- **Types d'attributs :**
  - **Spécifiques à une classe (**ex. : textSize pour TextView).
  - **Communs** à toutes les Views (ex. : android:id).
- **Paramètres de mise en page** (définis par le parent, ex. : layout_width, layout_height).
  - Les attributs permettent de définir :
  - Taille, position, texte, couleur, etc.
  - Organisation dans le conteneur (ViewGroup).

L’attribut ID et son utilisation

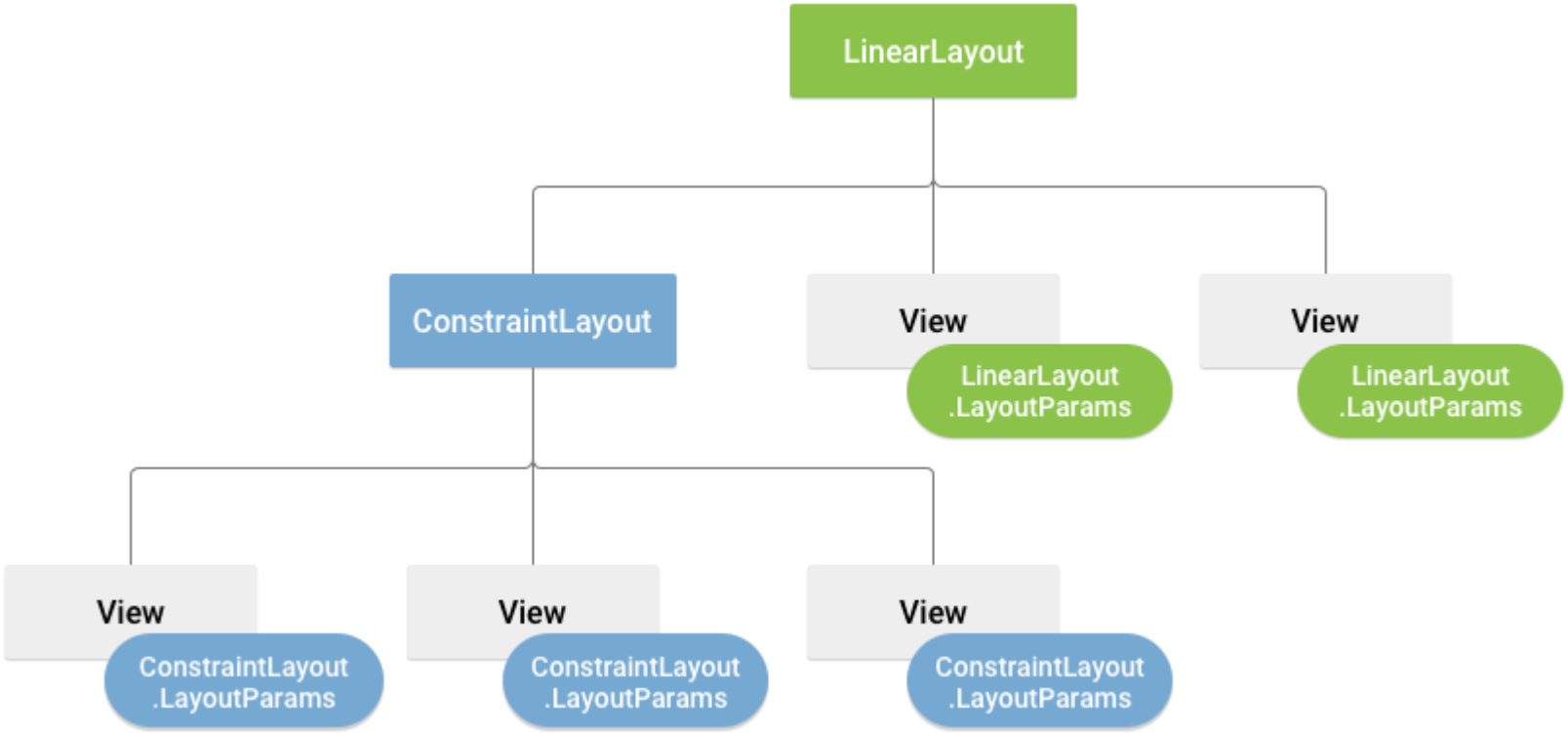
- ID : Attribut commun à toutes les Views pour les identifier.
  - **Déclaration dans XML** :

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

    - @ : indique une ressource.
    - + : crée un nouvel ID dans R.java
  - **Référence en Java :**

```
Button myButton = findViewById(R.id.my_button);
```

Paramètres de layput(LayoutParams)



- **Chaque ViewGroup définit des paramètres de mise en page (LayoutParams) pour ses vues enfants**, en précisant leur taille et leur position dans la hiérarchie.
- **Les attributs XML comme layout_width et layout_height sont obligatoires pour chaque vue**, tandis que d'autres paramètres (marges, bordures) sont facultatifs et dépendent du parent.

Paramètres de layput(LayoutParams)

- Les attributs layout définissent **taille et position** des vues dans leur **ViewGroup parent**.
- Chaque **ViewGroup** possède sa propre sous-classe LayoutParams pour gérer ces paramètres.
- **Attributs obligatoires :**
  - layout_width
  - layout_height
- **Valeurs courantes :**
  - wrap_content → Ajuste à son contenu.
  - match_parent → Occupe tout l'espace du parent.
- **Bonnes pratiques :**
  - Éviter les tailles fixes en pixels.
  - Utiliser des unités relatives (dp) ou constantes (wrap_content, match_parent) pour compatibilité multi-écrans.
- **Options supplémentaires :** marges, bordures.

Types des Layout Andoird

- Un Layout est un conteneur qui organise la disposition des vues (boutons, textes, images) dans une interface Android. Il définit comment les éléments sont positionnés et dimensionnés à l'écran.
- Si aucun type de layout n'est spécifié, tous les éléments seront placés en haut à gauche de l'écran.
- Chaque types de Layout répond à des besoins spécifiques :
  - Organisation simple en ligne ou colonne.
  - Positionnement relatif entre les vues.
  - Interfaces complexes et adaptatives.
- **Principaux types de Layouts :**
  - **LinearLayout** : organisation **linéaire** (verticale ou horizontale).
  - **RelativeLayout** : positionnement **par rapport aux autres vues ou au parent**.
  - **ConstraintLayout** : disposition **flexible et optimisée**, adaptée aux interfaces modernes.

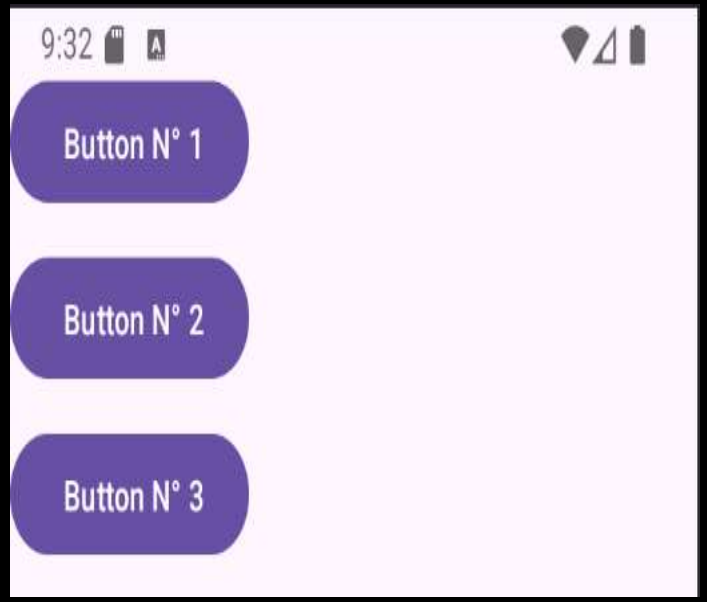
Introduction au LinearLayout

- LinearLayout est un conteneur (ViewGroup) qui organise ses éléments enfants d'une manière linéaire:
  - Verticale : les vues sont empilées de haut en bas.
  - Horizontale : les vues sont placées côte à côte.
- Caractéristique principale :
  - L'attribut android:orientation détermine la direction (verticale ou horizontale).
- Utilisation :
  - Idéal pour des interfaces simples où les éléments doivent être alignés dans une seule direction.

Exemple (Verticale)

<LinearLayout android:orientation="vertical"

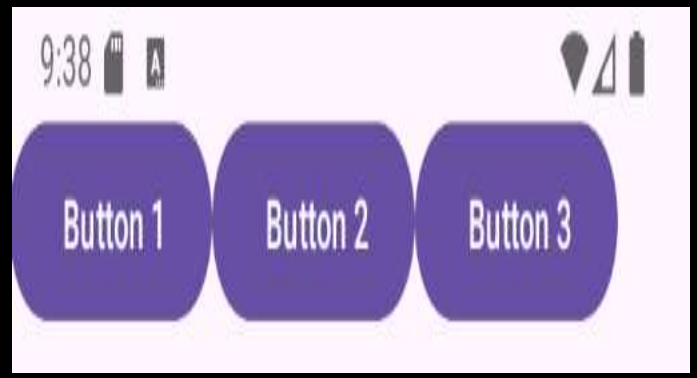
```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text=" Button N° 1"/>
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text=" Button N° 2"/>
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text=" Button N° 3"/>
</LinearLayout>
```



Exemple (horizontal)

<LinearLayout android:orientation="horizontal"

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal">

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text=" Button N° 1"/>
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text=" Button N° 2"/>
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text=" Button N° 3"/>
</LinearLayout>
```



LinearLayout : Attributs supplémentaires

- **android:layout_width** : Définit la largeur du widget.
  - Valeurs courantes :
    - match_parent → occupe toute la largeur disponible
    - wrap_content → s'adapte au contenu
- **android:layout_height** : Définit la hauteur du widget.
  - Valeurs courantes :
    - match_parent
    - wrap_content
- **android:layout_weight** (optionnel mais très utilisé)Permet de répartir l'espace entre les widgets dans un LinearLayout.
  - Exemple : android:layout_weight="1" pour donner un poids égal à plusieurs vues.
- **android:gravity** (pour le contenu interne)Aligne le texte ou le contenu à l'intérieur du widget (ex. center, left, right).
- **android:layout_gravity** (pour la position dans le LinearLayout)Aligne le widget dans le conteneur (ex. center, start, end).

Introduction au RelativeLayout

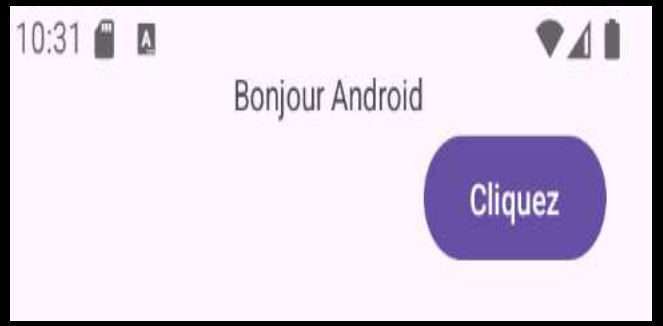
- **RelativeLayout** est un conteneur qui positionne ses vues **par rapport aux autres vues ou par rapport au parent**.
- Plus flexible que LinearLayout. Permet des interfaces complexes sans imbriquer plusieurs layouts.
- Attributs importants dans RelativeLayout
  - Par rapport au parent :
    - android:layout_alignParentTop="true"
      - aligner le bord supérieur d'une vue enfant avec le bord supérieur de son conteneur parent.
    - android:layout_alignParentBottom="true"
      - Aligne le bord inférieur de la vue sur le bord inférieur de son parent.
    - android:layout_centerInParent="true"
      - utilisé pour centrer une vue à la fois horizontalement et verticalement par rapport à son parent
  - Par rapport aux autres vues :
    - android:layout_below="@id/element"
      - pour positionner le bord supérieur d'une vue directement sous le bord inférieur d'une autre vue spécifiée par son identifiant (ID).
    - android:layout_toRightOf="@id/element"
      - pour placer le bord gauche d'une vue à la droite d'une autre vue spécifiée par son ID.
    - android:layout_alignStart="@id/element"
      - pour aligner le bord de début d'une vue sur le bord de début d'une autre vue spécifiée par son ID.
    - android:layout_toEndOf="@id/element"
      - placer une vue immédiatement après une autre vue spécifiée par son ID

Exemple1 du RelativeLayout

```
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <TextView
      android:id="@+id/titre"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Bonjour Android"
      android:layout_centerHorizontal="true" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Cliquez"
      android:layout_below="@id/titre"
      android:layout_alignStart="@+id/titre"/>
</RelativeLayout>
```

Exemple2 du RelativeLayout

```
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <TextView
      android:id="@+id/titre"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Bonjour Android"
      android:layout_centerHorizontal="true" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Cliquez"
      android:layout_below="@id/titre"
      android:layout_toEndOf="@+id/titre"/>
</RelativeLayout>
```



Introduction au ConstraintLayout

- **ConstraintLayout** permet de positionner les vues en utilisant des contraintes (relations) entre elles ou par rapport au parent.
- Introduit pour remplacer les layouts imbriqués et améliorer la performance.
- Caractéristiques principales :
  - Flexibilité : Positionnement horizontal, vertical, centré, aligné.
  - Performance : Réduit la profondeur de la hiérarchie des vues.
  - Compatibilité : Fonctionne avec tous les widgets Android.

Attributs essentiels de ConstraintLayout

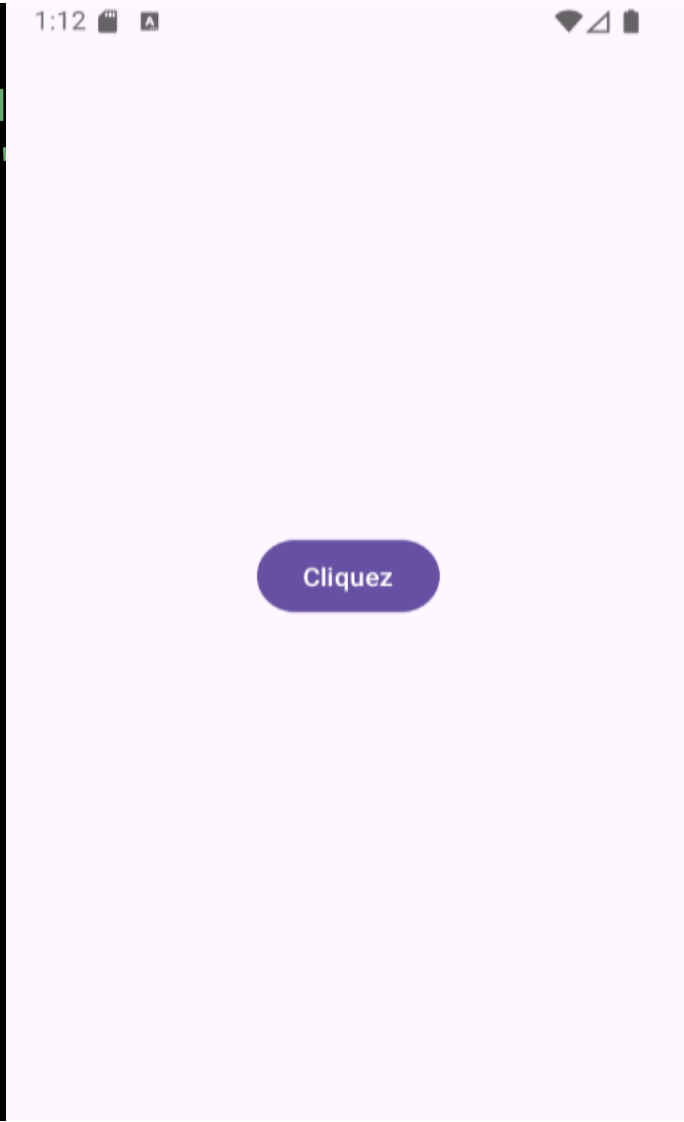
- Contraintes de positionnement:
  - **app:layout_constraintTop_toTopOf="parent"** → Aligne le haut de la vue avec le haut du parent.
  - **app:layout_constraintBottom_toBottomOf="parent"** → Aligne le bas de la vue avec le bas du parent.
  - **app:layout_constraintStart_toStartOf="parent"** → Aligne à gauche (ou début) du parent.
  - **app:layout_constraintEnd_toEndOf="parent"** → Aligne à droite (ou fin) du parent.
- Contraintes entre vues:
  - **app:layout_constraintStart_toEndOf="@id/vue"** → Positionne à droite d'une autre vue.
  - **app:layout_constraintTop_toBottomOf="@id/vue"** → Positionne en dessous d'une autre vue.

ConstraintLayout

Exemple 1 : Centrer un bouton

Le bouton est **centré horizontalement et verticalement** dans le parent.

```
<androidx.constraintlayout.widget.ConstraintLayout
   xmlns:android="http://schemas.android.com/apk/res/android
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

<Button
      android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Cliquez"
      app:layout_constraintTop_toTopOf="parent"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintEnd_toEndOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

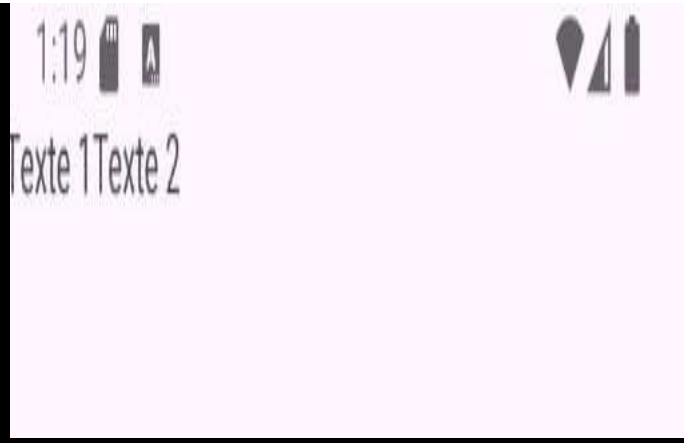


ConstraintLayout

Exemple 2 : Deux vues alignées en chaîne (Chain)

Les deux **TextView** sont alignés côte à côte grâce aux contraintes.

```
<TextView
   android:id="@+id/text1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Texte 1"
   app:layout_constraintStart_toStartOf="parent"
   app:layout_constraintTop_toTopOf="parent"/>

<TextView
   android:id="@+id/text2"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Texte 2"
   app:layout_constraintStart_toEndOf="@id/text1"
   app:layout_constraintTop_toTopOf="@id/text1"/>
```



ConstraintLayout

Exemple 3 : Utilisation d'une Guideline

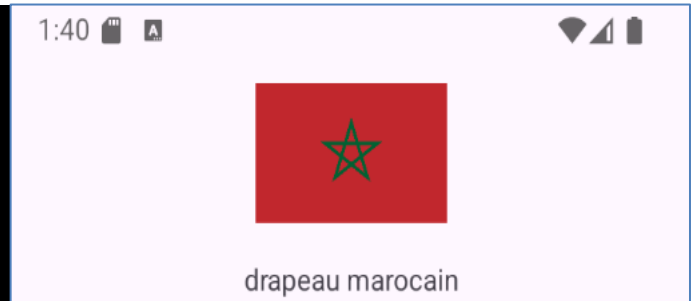
Le bouton est positionné à droite d'une ligne virtuelle (Guideline).

```
<androidx.constraintlayout.widget.ConstraintLayout
   xmlns:android=....     xmlns:app="...
  android:layout_width="match_parent"
   android:layout_height="match_parent">
   <!-- Guideline verticale -->
   <androidx.constraintlayout.widget.Guideline
      android:id="@+id/guideline"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:orientation="vertical"
      app:layout_constraintGuide_begin="100dp" />

   <!-- Bouton contraint à la Guideline -->
   <Button
      android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="À droite"
      app:layout_constraintStart_toStartOf="@id/guideline"
      app:layout_constraintTop_toTopOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

ConstraintLayout

Exemple 4 : Image centrée avec texte en dessous

```
...
    <ImageView
        android:id="@+id/image"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:src="@drawable/morocco_flag"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>

    <TextView
        android:id="@+id/label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="drapeau marocain"
        app:layout_constraintTop_toBottomOf="@id/image"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```



Le principe du Stack en Android

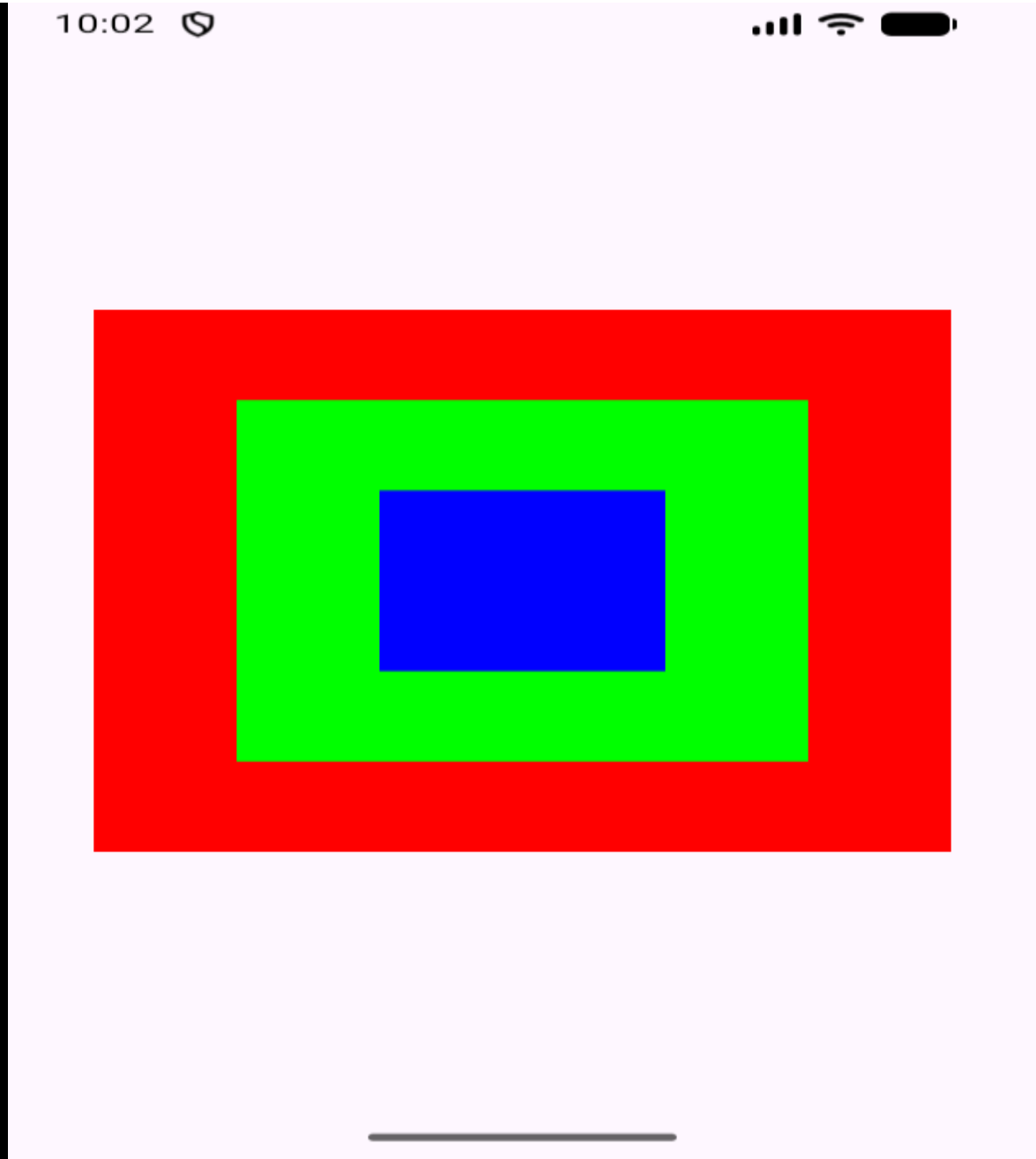
- Le mot Stak signifie Empiler plusieurs vues dans le même conteneur pour les superposer.
- Ordre d’affichage : La première vue est en arrière-plan, les suivantes au-dessus.
- Layout utilisé : FrameLayout (simule le concept de Stack).
- Utilité :
  - Créer des interfaces avec image de fond + texte + bouton.
  - Afficher des éléments superposés (ex. icône sur une image).
- Alternative avancée : ConstraintLayout peut aussi gérer le stacking avec plus de flexibilité.

Le principe du Stack – exemple1

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <ImageView
      android:src="@drawable/morocco_flag"
      android:scaleType="centerCrop"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />

   <TextView
      android:text="Bienvenue"
      android:textSize="24sp"
      android:textColor="#FFFFFF"
      android:layout_gravity="center"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />

   <Button
      android:text="Continuer"
      android:layout_gravity="bottom|center_horizontal"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />
</FrameLayout>
```

Le principe du Stack – exemple2

```xml
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- Grand rectangle (arrière-plan) -->
    <View
        android:layout_width="300dp"
        android:layout_height="300dp"
        android:background="#FF0000"
        android:layout_gravity="center" />

    <!-- Rectangle moyen -->
    <View
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:background="#00FF00"
        android:layout_gravity="center" />

    <!-- Petit rectangle (premier plan) -->
    <View
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:background="#0000FF"
        android:layout_gravity="center" />
</FrameLayout>
```

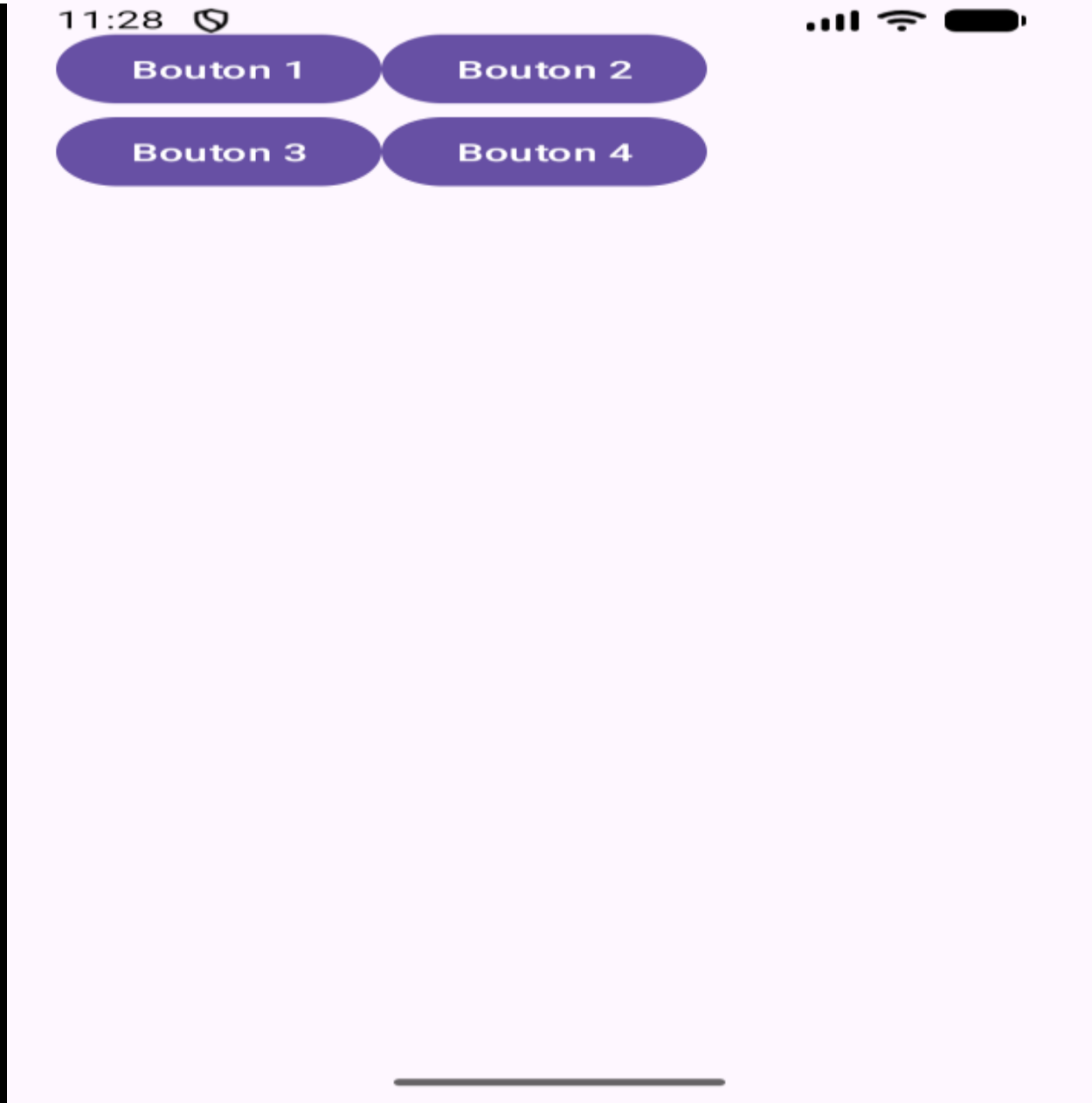


Introduction au GridLayout

- GridLayout est un conteneur qui organise les vues en grille (lignes et colonnes).
- Permet un positionnement précis des éléments et il est flexible pour créer des interfaces complexes.
- Utilisation typique :Formulaires, tableaux, écrans avec alignement structuré.
- Avantages :
  - Alignement précis des composants.
  - Adapté aux écrans variés.
- Bonnes pratiques :
  - Définir des marges pour éviter la surcharge visuelle.
  - Utiliser android:layout_rowSpan et android:layout_columnSpan pour fusionner des cellules.
- Alternatives :
  - ConstraintLayout (plus flexible pour interfaces modernes).

Exemple 1 de **GridLayout**

```
<GridLayout
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:rowCount="2"
   android:columnCount="2"
   tools:ignore="UselessParent">

   <Button android:text="Bouton 1"
      android:layout_row="0"
      android:layout_column="0" />

   <Button android:text="Bouton 2"
      android:layout_row="0"
      android:layout_column="1" />

   <Button android:text="Bouton 3"
      android:layout_row="1"
      android:layout_column="0" />

   <Button android:text="Bouton 4"
      android:layout_row="1"
      android:layout_column="1" />

</GridLayout>
```

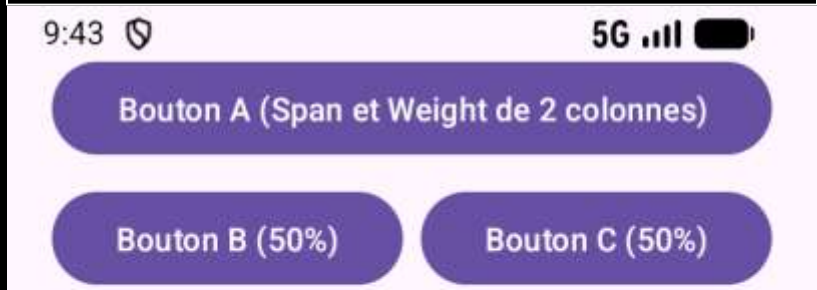


Attributs essentiels de Gridlayout

Attribut	Description du Rôle
layout_rowSpan	Indique au **GridLayout** que l'élément doit bloquer l'espace de N lignes consécutives, **fusionnant ainsi N cellules verticalement** dans la structure logique de la grille.
layout_columnSpan	Indique au **GridLayout** que l'élément doit bloquer l'espace de N colonnes consécutives**, fusionnant ainsi N cellules horizontalement** dans la structure logique de la grille.
layout_rowWeight	Indique au système d'allocation de l'espace comment étirer l'élément verticalement pour remplir l'espace restant dans sa(ses) ligne(s), en lui attribuant N parts de cet espace disponible.
layout_columnWeight	Indique au système d'allocation de l'espace comment étirer l'élément horizontalement pour remplir l'espace restant dans sa(ses) colonne(s), en lui attribuant N parts de cet espace disponible.

Exemple 2 de **GridLayout**

```xml
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"     android:layout_height="match_parent"
    android:columnCount="2" android:rowCount="2" android:padding="16dp"
    android:useDefaultMargins="true">

    <!-- Bouton A : S'étend sur 2 colonnes (columnSpan=2) -->
    <!-- Il utilise 2 parts de poids horizontal (columnWeight=2) -->
    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_columnSpan="2"
        android:layout_columnWeight="2"
        android:text="Bouton A (Span et Weight de 2 colonnes)" />

    <!-- Bouton B (Ligne 1, Colonne 0) -->
    <!-- Il utilise 1 part de poids horizontal (columnWeight=1) -->
    <Button
        android:layout_width="0dp"          android:layout_height="wrap_content"
        android:layout_columnWeight="1"                 android:text="Bouton B (50%)" />

    <!-- Bouton C (Ligne 1, Colonne 1) -->
    <!-- Il utilise 1 part de poids horizontal (columnWeight=1) -->
    <Button
        android:layout_width="0dp"          android:layout_height="wrap_content"
        android:layout_columnWeight="1"                 android:text="Bouton C (50%)" />

</GridLayout>
```



Le principe des écouteurs d'événements

- Le système Android utilise des interfaces Java spécifiques (les écouteurs) pour créer un pont entre une action physique de l'utilisateur et le code Java que vous écrivez.
  - Le View (widget) : C'est l'élément d'interface graphique (Button, View, etc.) qui détecte l'action.
  - L'Observateur (Écouteur) : C'est votre code Java qui implémente une interface spécifique.
  - La Méthode de Rappel (Callback) : La méthode unique définie dans l'interface qui est automatiquement appelée ("rappelée") lorsque l'événement se produit.
- Pour qu'un écouteur fonctionne, vous devez l'enregistrer auprès du sujet via une méthode set...Listener() (par exemple, setOnClickListener()).

Gestion des Actions Utilisateur : Le Clic (View.OnClickListener)

- L'événement le plus courant est le simple clic ou appui sur un élément interactif (un bouton, une image cliquable, etc.). Pour cela, nous utilisons l'interface Java View.OnClickListener.
- **Méthode 1 : Utilisation d'une Classe Anonyme Interne (Recommandée)**
- Cette méthode est la plus directe et se fait entièrement dans votre fichier Activity.java (généralement dans la méthode onCreate()).

```
    Button bt1 = findViewById(R.id.btn1);
    TextView tv = findViewById(R.id.tvRes);
    bt1.setOnClickListener( View v -> {
        //Ce bloc s'exécute uniquement au moment du clic
        tv.setText("Boutton Clické");
    });

}
}
```

ou

```
    Button bt1 = findViewById(R.id.btn1);
    TextView tv = findViewById(R.id.tvRes);
    bt1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            //Ce bloc s'exécute uniquement au moment du clic
            tv.setText("Boutton Clické");
        }
    });
}
}
```

Gestion des Actions Utilisateur : Le Clic (View.OnClickListener)

- **Méthode 2 : L'attribut XML android:onClick**
- Une alternative rapide consiste à spécifier le nom de la méthode de rappel directement dans le fichier de mise en page XML.
- Dans votre fichier res/layout/activity_main.xml :

```
<Button
    android:id="@+id/button_xml"
     android:text="Cliquer via XML"
    android:onClick="gererClicXML" /> <!-- Nom de la méthode ici -->
```

- Dans votre fichier MainActivity.java :

```
public class MainActivity extends AppCompatActivity {
    // ... (onCreate methods) ...
    / * Méthode de rappel appelée par l'attribut android:onClick dans le XML.
     * DOIT être publique, void, et accepter une View en paramètre. */
public void gererClicXML(View view) {
tv.setText("Le bouton a été cliqué !");
    }
}
```

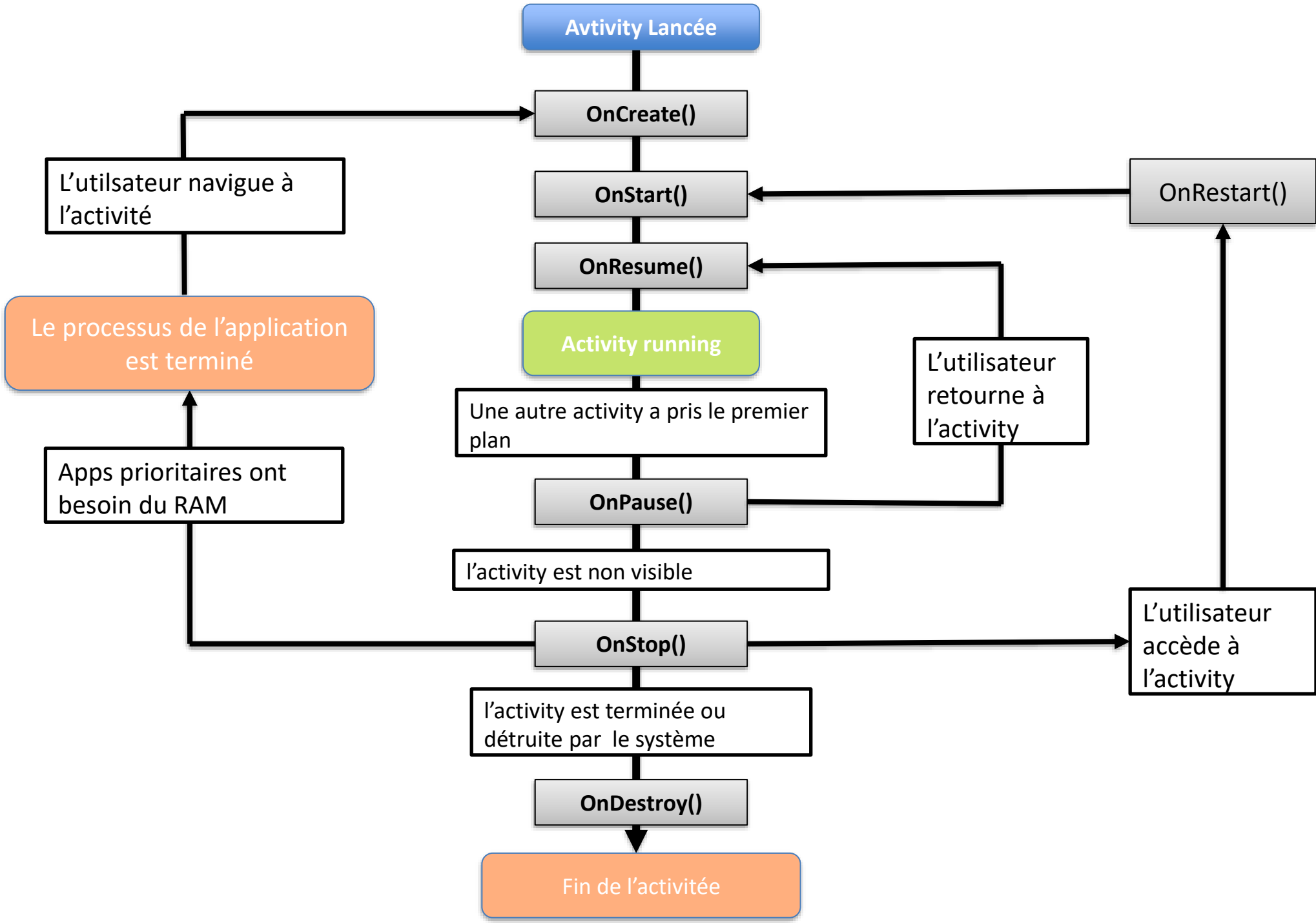
Interactions Tactiles

- Pour des interactions plus fines que le simple clic (comme détecter si l'utilisateur appuie, glisse, ou relâche son doigt), nous utilisons View.OnTouchListener.
- Cette interface utilise l'objet MotionEvent qui fournit des détails précis (coordonnées X/Y, type d'action).
- La méthode onTouch() doit retourner true si vous avez complètement géré l'événement tactile. Si vous retournez false, l'événement peut être passé à d'autres écouteurs ou vues parentes pour un traitement supplémentaire. Pour les simples clics, onClick() gère cela automatiquement

Interactions Tactiles - Exemple

```
Button bt1 = findViewById(R.id.btn1);
TextView tv = findViewById(R.id.tvRes);

bt1.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {

        // Vérifie si l'action est ACTION_DOWN (quand le doigt touche le bouton)
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            // Met à jour le TextView pour indiquer que le doigt vient de toucher le bouton
            tv.setText("ACTION_DOWN");
        }

        // Vérifie si l'action est ACTION_UP (quand le doigt est levé du bouton)
        if (event.getAction() == MotionEvent.ACTION_UP) {
            // Met à jour le TextView pour indiquer que le doigt a été relâché
            tv.setText("ACTION_UP");
        }

        // Vérifie si l'action est ACTION_MOVE (quand le doigt glisse sur le bouton)
        if (event.getAction() == MotionEvent.ACTION_MOVE) {
            // Met à jour le TextView pour indiquer que le doigt est en mouvement sur le bouton
            tv.setText("ACTION_MOVE");
        }
        return false;
    }});
```

Cycle de vie d'une activity



Introduction au Cycle de Vie d ’une activity

- Une Activity = un écran avec lequel l’utilisateur interagit.
- Android gère automatiquement son cycle de vie pour :
  - Optimiser la mémoire et les performances.
  - Adapter l’UI aux changements (rotation, multitâche).
- États principaux :

  Création → Visible → Premier plan → Pause → Arrêt → Destruction.
- Chaque état correspond à des callbacks spécifiques.
- **"callback"** :Une méthode appelée automatiquement par le système en réponse à un événement (ex. onCreate() est appelée quand l’Activity est créée).

Les callbacks essentiels du cycle de vie

- **onCreate**(): Appelée une seule fois lors de la création de l'Activity.
  - Rôle : Initialiser l'interface (via setContentView()), créer les objets, configurer les listeners.
  - Exemple : Charger le layout XML et préparer les composants.
- **onStart**(): L'Activity devient visible à l'utilisateur mais n'est pas encore interactive.
  - Rôle : Préparer les ressources nécessaires à l'affichage (ex. démarrer animations légères).
- **onResume**() : L'Activity est au premier plan et l'utilisateur peut interagir.
  - Rôle : Relancer les tâches interrompues (ex. flux vidéo, capteurs).
- **onPause**() : L'Activity perd le focus (ex. ouverture d'une boîte de dialogue ou passage à une autre Activity).
  - Rôle : Sauvegarder les données temporaires, arrêter les tâches sensibles (ex. caméra).
- **onStop**() : L'Activity n'est plus visible.Rôle : Libérer les ressources lourdes (ex. fermer connexion BD, arrêter threads).
- **onDestroy**()→ Dernière étape avant suppression de l'Activity.Rôle : Nettoyer complètement la mémoire, libérer les ressources finales.
- **onRestart**()→ Appelée après onStop() si l'Activity revient au premier plan.Rôle : Réinitialiser ce qui a été arrêté.

Différences entre onPause(), onStop(), onDestroy()

- onPause()
  - Activité partiellement visible (ex. boîte de dialogue au-dessus).
  - Toujours en mémoire, mais pas active.
  - Actions recommandées :
    - Sauvegarder l'état léger (texte saisie, position du scroll).
    - Arrêter les animations ou la lecture multimédia.
- onStop()
  - Activité invisible, mais encore en mémoire.
  - Peut être détruite si le système manque de RAM.
  - Actions recommandées :
    - Libérer les ressources lourdes (GPS, capteurs).
    - Fermer les connexions réseau inutiles.
- onDestroy()
  - Activité supprimée définitivement (finish() ou rotation).
  - Dernière étape avant la libération complète.
  - Actions recommandées :
    - Nettoyer les threads, fermer les bases de données.

Fin du processus vs Fin de l’activité

- Fin du processus (après onStop si manque de RAM)
  - Le système tue l’application entière pour libérer la mémoire.
  - Pas d’appel à onDestroy().Lors du retour → onCreate() relance tout.
- Fin de l’activité (après onDestroy)
  - Passage d’une activité à une autre activité (Accueil à Profil par exemple)
  - onDestroy() est exécuté → nettoyage final possible.
  - Le processus peut rester actif si d’autres activités existent.