DEVELOPPEMENT MOBILE ANDROID
INTRODUCTION

INSA BADJI Doctorant à l'Université de Thiès / Tuteur à

Séquence 2: Plateforme Android

- Objectif

- Plan
  - Les composants : Activity, Service, Content Provider, Broadcast Receiver
  - Anatomie d'un projet Android : Manifest, Resources et qualifications, R.java, Internationalisation string.xml, Assets, Libraries
  - Cycle de vie d’une application et persistance
  - Un premier projet
  - Exécution et adb
  - Logcat

Basé sur le cours du Pr. Ousmane SALL

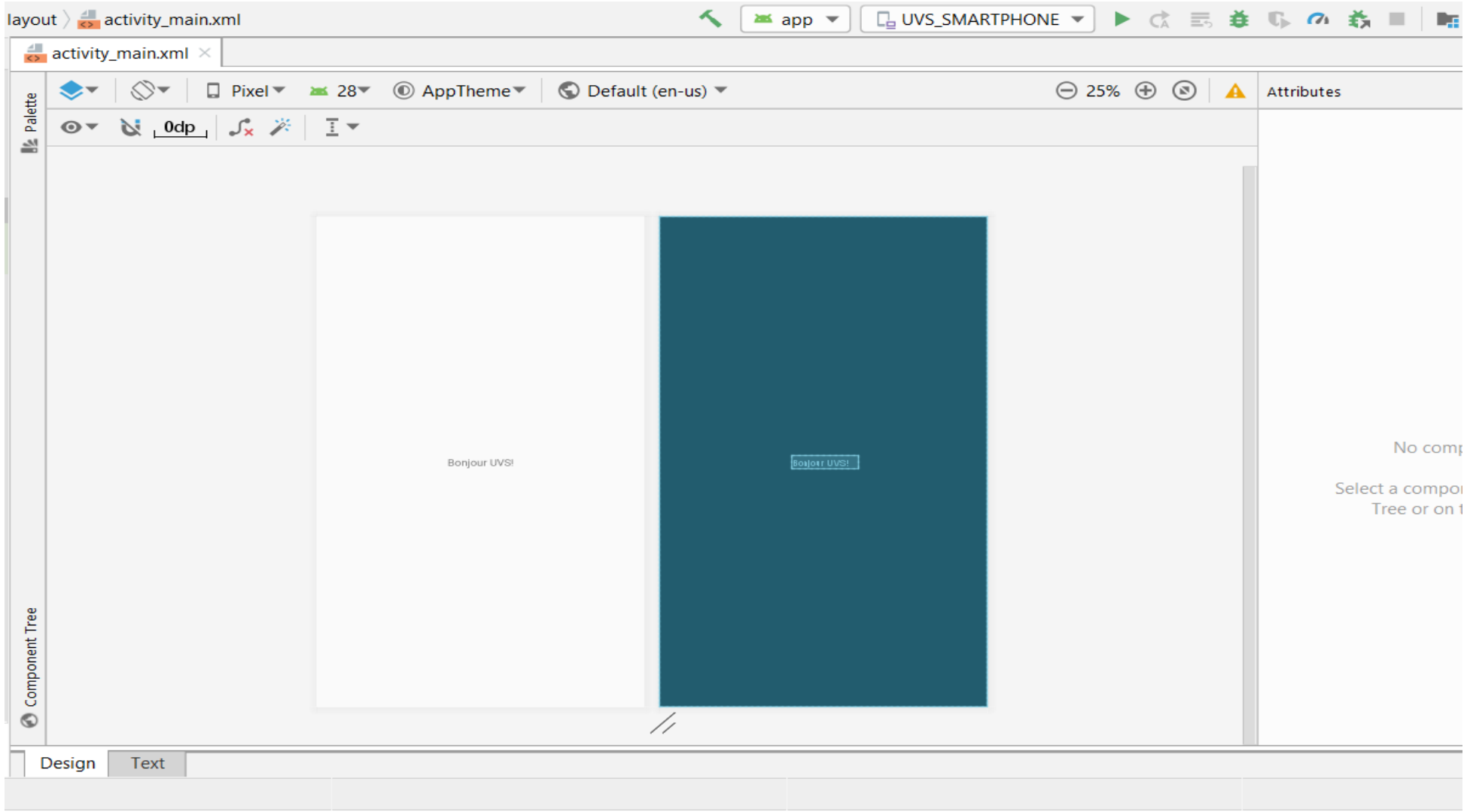
Le fichier activity_main

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=« Bonjour UVS!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

Le fichier activity_main



Le fichier AndroidManifest

Généré par Android studio, contient la description de l'application

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.premierprojetandroiduvs">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="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>
```

- On modifiera ce fichier pour déclarer les éléments de l'application, les permissions, etc.

Rubriques du fichier AndroidManifest

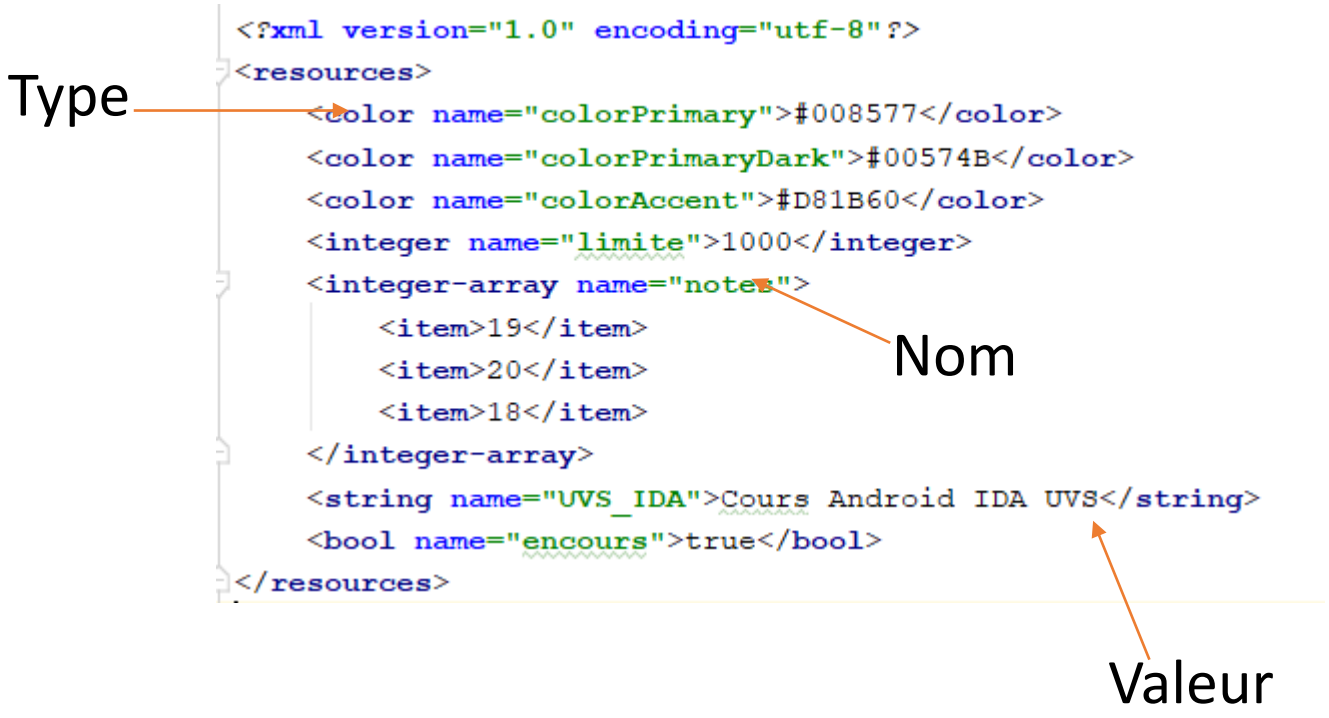
- Manifest
  - Nom du paquetage
  - Versions
  - SDK
- Application
  - Nom
  - Icône
  - Eléments constituant l'application (activités, services, …)
- Permissions
  - Accès aux capteurs
  - Accès au réseau
  - Accès à la téléphonie
  - …
- Instrumentation (pour les tests)

Les ressources

- Application embarquée tout doit être dans le fichier .apk téléchargé
- Le répertoire **res** contient toutes les ressources qui seront mises dans le apk :
  - drawable-hdpi (images en haute définition)
  - drawable-ldpi (images en basse définition)
  - drawable-mdpi (images en moyenne définition)
  - layout (description en XML des interfaces)
  - values (définitions en XML de constantes : chaînes, tableaux, valeurs numériques …)
  - anim (description en XML d'animations)
  - menus (description en XML de menus pour l'application)
  - xml (fichiers XML utilisés directement par l'application)
  - raw (tous les autres types de ressources : sons, vidéos, …) On peut ajouter d'autres sous répertoires

Créer des ressources valeurs

Les ressources de type valeur sont décrites dans des fichiers XML ayant la forme suivante :




```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
    <integer name="limite">1000</integer>
    <integer-array name="notes">
        <item>19</item>
        <item>20</item>
        <item>18</item>
    </integer-array>
    <string name="UVS_IDA">Cours Android IDA UVS</string>
    <bool name="encours">true</bool>
</resources>
```

Les noms (identificateurs) servents à les designer:

- Dans d'autres fichiers XML
- Dans le code

La classe R

- C’est une classe générée par Android studio qui permet à l’application d’accéder aux ressources
- Elle contient des classes internes dont les noms correspondent aux types de ressources (id, drawable, layout …)
- Elle est constituée à partir des fichiers placés dans les sous répertoires du répertoire res
- Une propriété est créée pour :
  - Chaque image placé dans drawable-xxxx
  - Chaque identificateur défini dans des fichiers XML (objets d’interface, constantes)
  - Chaque fichier placé dans les répertoires xml , raw …

Utilisation des ressources

- Référencement d'une ressource dans un fichier xml. La forme générale est : "@type/identificateur"
- Par exemple : @string/*machaine* Fait référence à une chaine contenue dans un fichier XML placé dans le répertoire **res/values et définie comme suit :**

```
<resources>
    …
    <string name="machaine">Contenu de cette chaine</string>
    …
</resources>
```

- Référencement d'une ressource dans le code. La forme générale est : R.type.identificateur
  - Par exemple : R.string.*machaine*
  - Fait référence à la même chaine

La classe Resources

- Permet l'accès aux ressources répertoriées dans R
- On obtient une instance de cette classe par getResources() de l'activité
- Principales méthodes de la classe Resources (le paramètre est un identifiant défini dans R de la forme R.type.nom) :
  - boolean getBoolean(int)
  - int getInteger(int)
  - int[] getArray(int)
  - String getString(int)
  - String[] getStringArray(int)
  - int getColor(int)
  - float getDimension(int)
  - Drawable getDrawable(int)
- Exemple : String titre = getResources().getString(R.string.ma_chaine);

Utilisation des ressources

- Accès aux ressources dans l'application
  - Mise en place de l'interface principale
    - setContentView(R.layout.nom_du_fichier_xml);
  - Mise en place d'interfaces supplémentaires
    - Par les classes LayoutInflater ou MenuInflater
  - Accès direct à une valeur ou à une ressource :
    - String titre = getResources().getString(R.string.texte_titre);
    - Drawable monImage = getResources().getDrawable(R.drawable.nom_de_l_image)

Uri (Uniform Resource Identifiers)

- Désigne des ressources locales ou distantes (plus général que les URL car non lié à un protocole réseau)
- Création
  - Ressource locale
    - Uri.parse("android.resource://nom_du_paquetage_de_l_activité/" +R.chemin.mon_son);
  - Ressource distante
    - Uri.parse("http://domaine.sous_doamine/chemin/nom_du_fichier");
    - Uri.fromFile(File)
- Codage en chaîne de caractères
  - toString()

Les chaines

- Les chaines constantes de l'application sont situées dans **res/values/strings.xml.** Voici un exemple:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello Hello Test !</string>
    <string name="app_name">AndroidTest</string>
</resources>
```

- La récupération de la chaine se fait via le code:

```
Resources res = getResources();
String hw = res.getString(R.string.hello);
```

Structure d’une application

- **Activité (android.app.Activity):** Programme qui gère une interface graphique
- **Service (android.app.Service):** Programme qui fonctionne en tâche de fond sans interface
- **Fournisseur de contenu (android.content.ContentProvider):**Partage d’informations entre applications
- **Ecouteur d’intention diffusées (android.content.BroadcastReceiver) :** Permet à une application de récupérer des informations générales (réception d’un SMS, batterie faible, …)
- **Eléments d’interaction**
  - **Intention (android.content.Intent) :** qu’elle sait faire ou de chercher un savoir-faire
  - **Filtre d’intentions (<intent-filter>) :** permet de choisir la meilleure application pour assurer un savoir-faire

Déclaration des éléments dans AndroidManifest.xml

- **Activité**

```
<activity>
<intent-filter>
…les savoir-faire
</intent-filter>
</activity>
```

- **Service**

```
<service>
<intent-filter>
… les savoir-faire
</intent-filter>
</service>
```

- **Ecouteur d'intention diffusée**

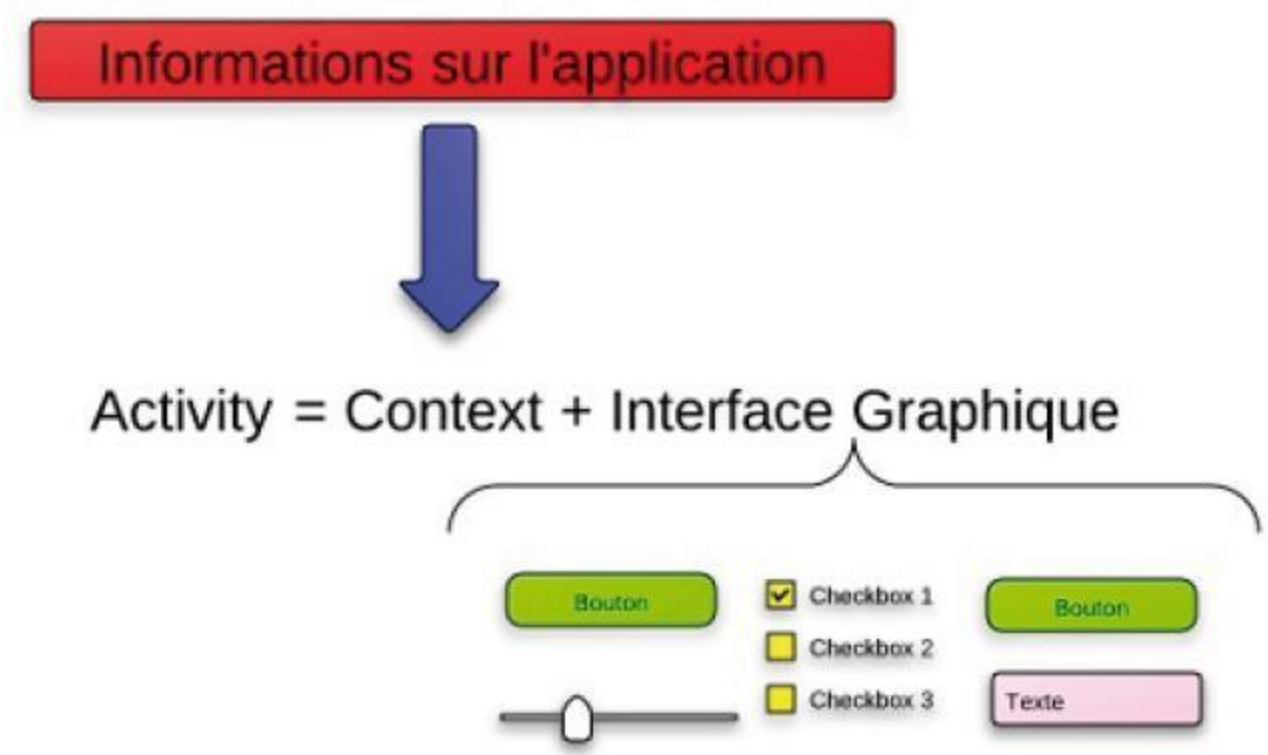
```
<receiver>
<intent-filter>
… les savoir-faire
</intent-filter>
</receiver>
```

- **Fournisseur de contenu**

```
<provider>
<grant-uri-permission …/>
<path-permission …/>
</provider>
```

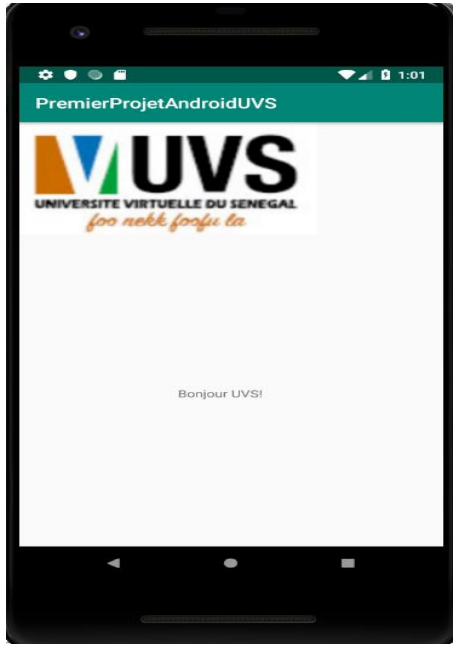
Application Android

- Une activité = un programme + une interface



Activités Android

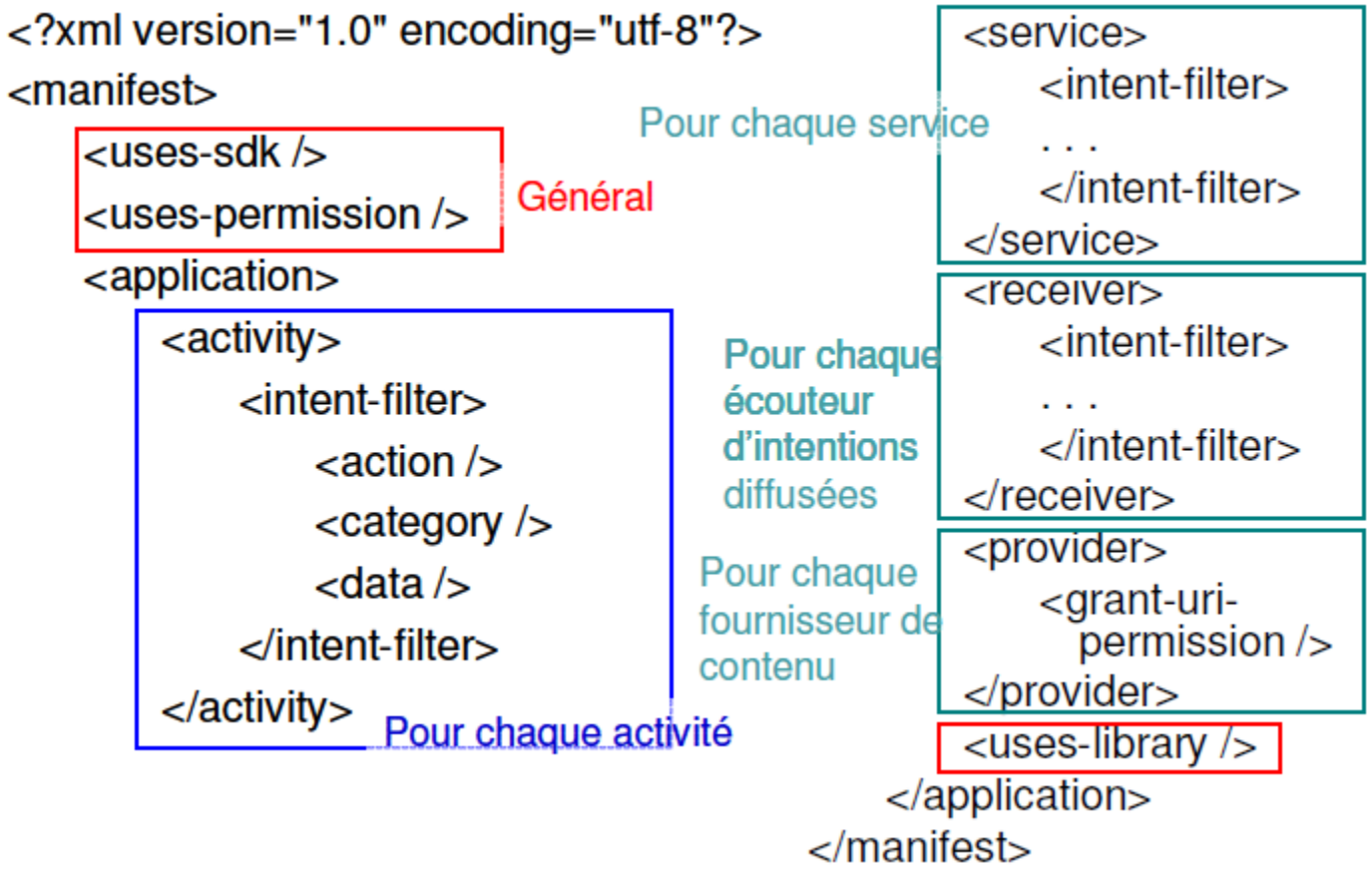
- Partie importante de l’interface graphique Android
- Peut être considéré comme une «fenêtre»



Application Android

- Un service = un programme sans interface
- Une application =
  - Une activité principale
  - Eventuellement une ou plusieurs activités secondaires
  - Eventuellement un ou plusieurs services
  - Eventuellement un ou plusieurs écouteurs d’intentions diffusées
  - Eventuellement un ou plusieurs fournisseurs de contenu

Contenu du fichier AndroidManifest



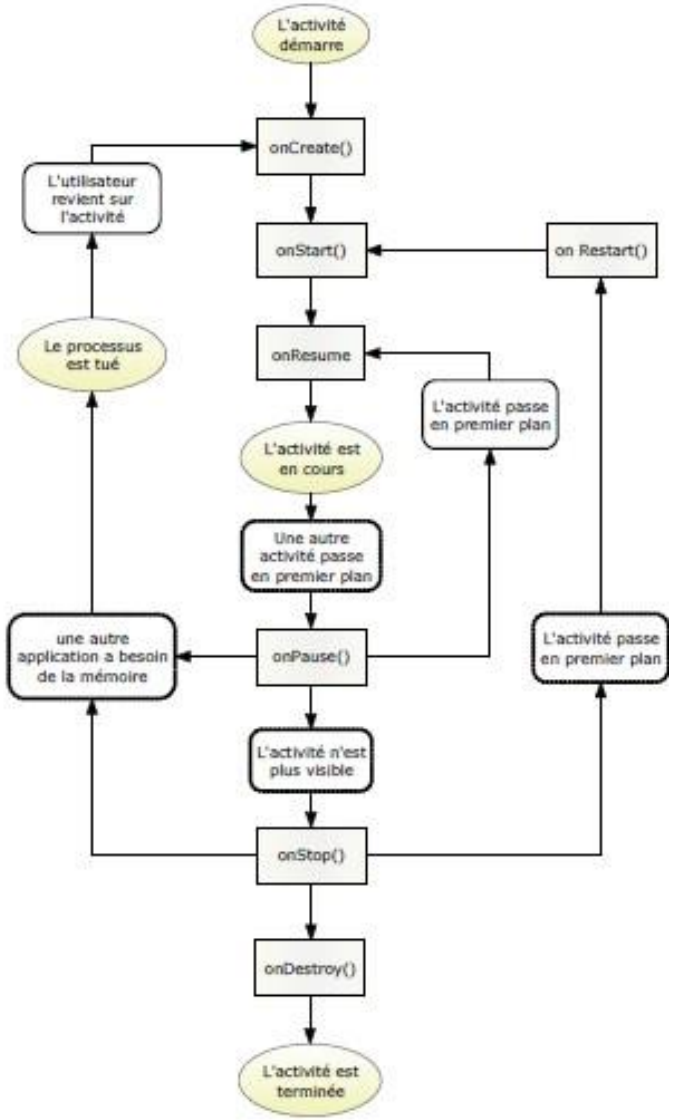
Activité Android

- Classe qui hérite de Activity ou d'une classe dérivée de Activity (par exemple de MapActivity pour utiliser Google maps, ListActivity ou TabActivity pour des interfaces particulières)
- On surcharge certaines méthodes qui sont appelées par Android pour définir le comportement (même principe que les applets) :
  - onCreate lors de la création
  - onDestroy lorsque l'activité se termine
  - onStart lorsque l'activité démarre ou redémarre
  - onPause lorsque l'activité n'est plus en premier plan
  - onResume lorsque l'activité revient en premier plan
  - onStop lorsque l'activité n'est plus visible
  - onRestart lorsque l'activité redevient visible

Cycle de vie d'une activité

- Android se réserve le droit de tuer le processus unix d'une activité s'il n'y a plus assez de ressources (mémoire). Les règles sont les suivantes :
  - Une activité en premier plan n'est tuée que si c'est elle qui consomme trop de ressources.
  - Une activité en arrière plan ou non visible peut être tuée.
- Lorsqu'une activité a été tuée, si on revient dessus elle est relancée (onCreate)
  - On peut sauvegarder l'état (c'est-à-dire les propriétés) d'une activité (dans onPause) pour le retrouver lorsqu'elle elle est recréée par le paramètre transmis à onCreate

Cycle de vie d'une activité



Une application Android étant hébergée sur un système embarqué, le cycle de vie d'une application ressemble à celle d'une application Java ME. L'activité peut passer des états:

- démarrage -> actif: détient le focus et est démarré (**onStart invoqué)**
- actif -> suspendue: ne détient plus le focus (**onPause invoqué)**
- suspendue -> actif: **onResume invoqué**
- suspendue -> détruit: **onDestroy invoqué**

Pensez vos interfaces pour un smartphone

- Ecran tactile de petite taille
  - Eviter les interfaces trop touffues (on ne peut pas agrandir l'écran comme on agrandit une fenêtre)
  - Eviter les éléments cliquables trop petits (il faut pouvoir cliquer avec le doigt même si on a des gros doigts)
  - Eviter les élément cliquables trop tassés (il faut pouvoir cliquer sur le bon élément même si on vise mal)
- Le défilement se fait par touché/glissé
  - Pas trop d'ascenseurs (on ne peut pas faire défiler un conteneur entier ET des éléments de ce conteneur dans le même sens)
  - Pas d'ascenseurs mal placés (si tous les éléments sont cliquables comment faire défiler sans cliquer ?)
- L'écran peut être tourné
- Tous les smartphones n'ont pas la même définition d'écran

Création d’interfaces

- Par programme (comparable à java swing) mais avec des classes propres à Android
  - Définition de conteneurs (un conteneur = un conteneur + un mode de placement = JPanel + Layout)
  - Définition d’éléments d’interaction (widgets) + placement et ajout dans les conteneurs
- Par description dans des fichiers xml (forme déclarative statique)
- Une interface est un arbre dont la racine est l’écran et les feuilles les éléments de l’interface (boutons, textes, cases à cocher, …)

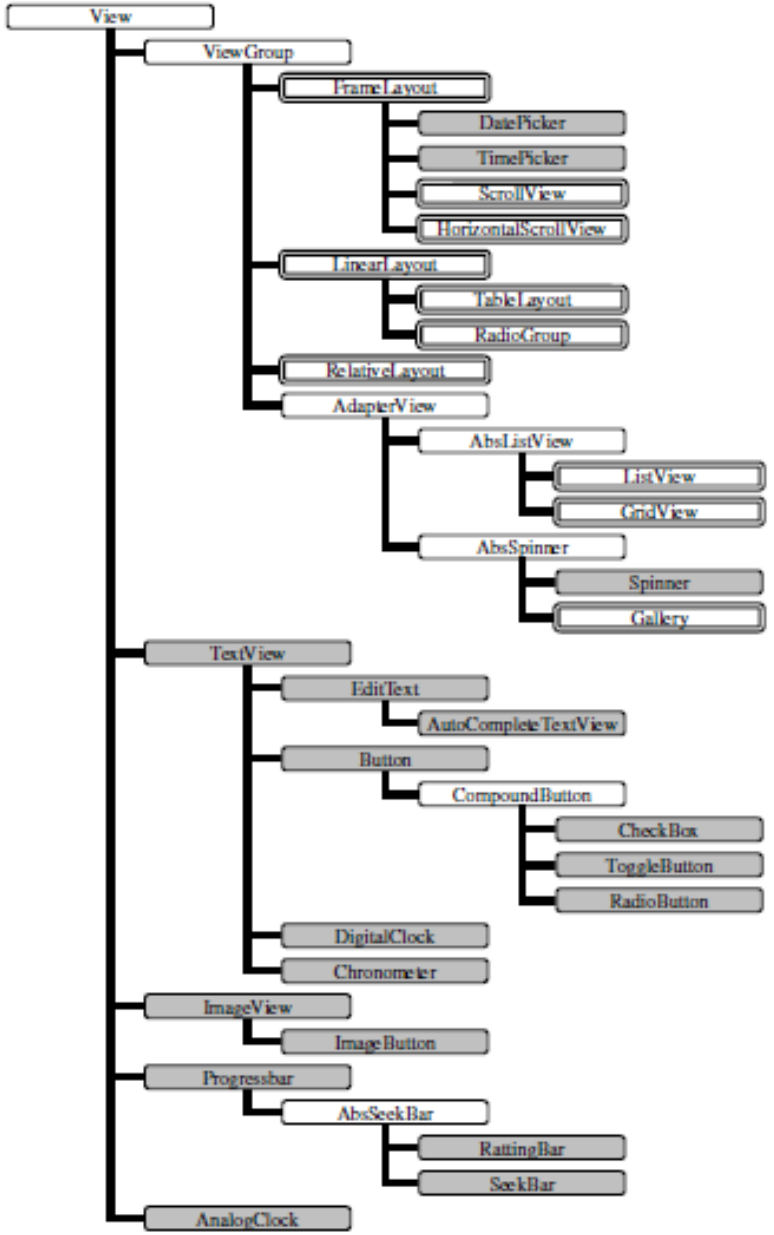
Hiérarchie partielle de classes pour les interfaces

- View
- ViewGroup
- TextView

Légende

Trait double = conteneurs ou groupes

Grisé = éléments d'interaction (wigets)



Définir une interface en XML

Définition de l'interface

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Commentaire
-->
<Classe_du_conteneur_principal
    xmlns:android="http://schemas.android.com/apk/res/android"
    propriétés du conteneur principal
>
   <Classe de conteneur ou d'élément d'interface
    propriétés  du conteneur ou de l'élément d'interface
    />
…
   <Classe de conteneur ou d'élément d'interface
    propriétés  du conteneur ou de l'élément d'interface
    />

</Classe_du_conteneur_principal>
```

Espace de nommage d'Android (imposé)

Lorsqu'il s'agit d'un conteneur il doit être décrit avec son contenu

Définir une interface en XML

Description d'un conteneur de l'interface

```
<Classe_de_conteneur
propriétés du conteneur
>
   <Classe de conteneur ou d'élément d'interface
    propriétés  du conteneur ou de l'élément d'interface
    />

...

   <Classe de conteneur ou d'élément d'interface
    propriétés  du conteneur ou de l'élément d'interface
    />

</Classe_du_conteneur>
```

Pour chaque conteneur

Créer une interface à partir d'un fichier XML

- Dans l'activité principale

  setContentView(R.layout.*nom_du_fichier_xml)*

- Ailleurs

  LayoutInflater decodeur = LayoutInflater.*from(contexte);*

  View vue=decodeur.inflate(R.layout.*nom_du_fichier_xml, parent, false);*

  - *contexte* est l'activité qui gère cette interface
  - *parent* est le contenant dans lequel doit se placer la vue constituée à partir du fichier XML

- Il ne reste plus qu'à ajouter cette vue dans le conteneur.

Unités de mesure dans les fichiers XML

- Dans les fichiers XML, les dimensions des éléments d'interface (taille, marges, …) peuvent être exprimées en diverses unités :
  - Pixels (px)
  - Pouces (in)
  - Millimètres (mm)
  - Points (pt) = 1/72 pouce
  - Pixel à densité indépendante (dp) 1 dp = 1 pixel pour un écran de 160 dpi
  - Pixel à taille indépendante (sp) relatif à la taille des polices de caractères
- Dans les fichiers XML les unités sont exprimées sous la forme : "24.5mm" ou "65px" …

Couleurs dans les fichiers XML

- Dans les fichiers XML, les couleurs sont exprimées sous la forme d'une chaîne de caractères codant les composantes en hexadécimal : "#AARRVVBB"
  - AA est l'opacité (00 totalement transparent, FF opaque)
  - RR est la composante rouge (00 à FF)
  - VV est la composante verte (00 à FF)
  - BB est la composante bleue (00 à FF)
- Si AA est omis la couleur est opaque

Les conteneurs

- FrameLayout (un seul élément)
- AbsoluteLayout (plusieurs éléments placés par leur coordonnées)
- LinearLayout (plusieurs éléments placés horizontalement ou verticalement sans ascenseurs)
- TableLayout (plusieurs éléments en tableau sans ascenseurs)
- RelativeLayout (plusieurs éléments placés relativement les uns aux autres)
- ScrollView (un seul élément avec ascenseur vertical)
- HorizontalScrollView (un seul élément avec ascenseur horizontal)

FrameLayout

- Ne contient qu’un seul élément (si on en met plusieurs ils se superposent)

- Propriétés supplémentaires :
- **Contenu**
  - *android:foreground Pour définir une couleur ou une* image.
  - *android:foregroundGravity Pour positionner l’image*

FrameLayout

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:layout_editor_absoluteX="46dp"
        tools:layout_editor_absoluteY="143dp">

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

FrameLayout

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <FrameLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        tools:layout_editor_absoluteX="46dp"
        tools:layout_editor_absoluteY="143dp">

        <ImageView
            android:id="@+id/imageView2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:adjustViewBounds="false"
            android:cropToPadding="false"
            app:srcCompat="@mipmap/capture" />

        <Button
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button" />
    </FrameLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

LinearLayout

- Pour placer plusieurs éléments en ligne ou en colonne sans ascenseur (utiliser ScrollView et/ou HorizontalScrollView).
- Propriétés supplémentaires :
  - android:orientation Pour en définir le sens du LinearLayout (vertical ou horizontal)
  - android:layout_weightSum Un paramètre de type : android:layout_weight peut être associé à chacun des éléments placés dans le LinearLayout pour indiquer leur poids de redimensionnement relatif à la valeur de layout_weightSum.
  - Par exemple : android:layout_weightSum= "100" permettra de placer 2 éléments ayant android:layout_weight = "60"
    et android:layout_weight = "40"

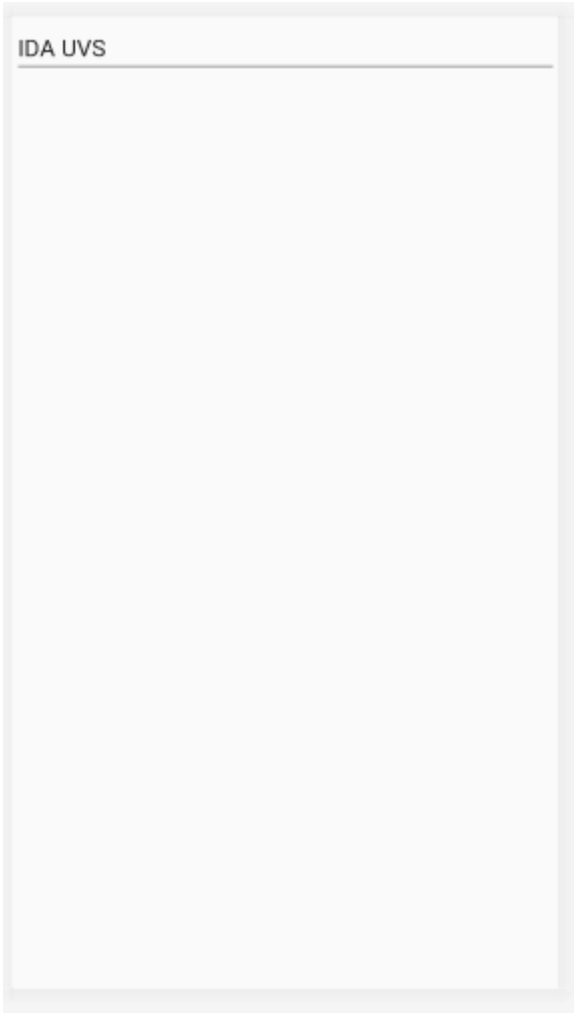
Exemple avec LinearLayout

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        tools:layout_editor_absoluteX="98dp"
        tools:layout_editor_absoluteY="128dp">

        <EditText
            android:id="@+id/editText"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:inputType="textPersonName"
            android:text="IDA UVS " />
    </LinearLayout>

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



AbsoluteLayout

- Pour placer les éléments par positions fixes selon des coordonnées.

- Chaque élément ajouté dans un AbsoluteLayout indiquera sa position en mettant dans ses propriétés :
  - android:layout_x="unité"
  - android:layout_y="unité"

TableLayout

- Pour placer des éléments en tableau sans ascenseurs (pour en avoir le mettre dans un ScrollView et/ou un HorizontalScrollView).

- Propriétés supplémentaires :
  - android:collapseColumns Pour définir les numéros de colonnes à cacher
  - android:shrinkColumns Pour définir les numéros de colonnes qui peuvent être rétrécies en fonction de la place disponible
  - android:stretchColumns Pour définir les numéros de colonnes qui peuvent être agrandies en fonction de leur contenu
  - Chaque élément ajouté dans un TableLayout indiquera le nombre de colonnes qu'il occupe en mettant dans ses propriétés : android:layout_span (par défaut 1)

RelativeLayout

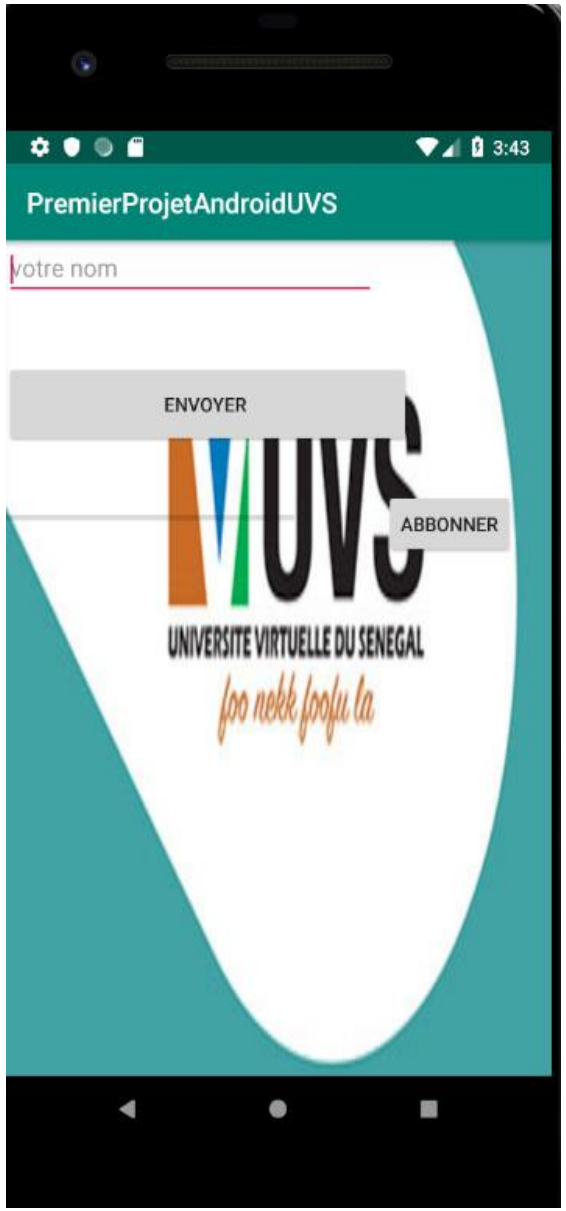
- Permet de placer des éléments les uns relativement aux autres
  - Placement par rapport au conteneur
    - android:layout_alignParentBottom="b" (où b vaut true ou false)
    - android:layout_alignParentLeft="b" (où b vaut true ou false)
    - android:layout_alignParentRight="b" (où b vaut true ou false)
    - android:layout_alignParentTop="b" (où b vaut true ou false)
    - android:layout_centerHorizontal="b" (où b vaut true ou false)
    - android:layout_centerInParent="b" (où b vaut true ou false)
    - android:layout_centerVertical="b" (où b vaut true ou false)
  - Placement par rapport aux autres éléments
    - android:layout_above="@+id/ident"/
    - android:layout_below="@+id/ident"/
    - android:layout_toLeftOf="@+id/ident"/
    - android:layout_toRightOf="@+id/ident"/
    - android:layout_alignLeft="@+id/ident"/
    - android:layout_alignRight="@+id/ident"/
    - android:layout_alignTop="@+id/ident"/
    - android:layout_alignBottom="@+id/ident"/

Exemple avec RelativeLayout

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto" xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent" android:layout_height="match_parent" tools:context=".MainActivity">

    <RelativeLayout android:layout_width="fill_parent«  android:layout_height="fill_parent"
        android:background="@mipmap/logo_uvs" tools:layout_editor_absoluteX="292dp"
        tools:layout_editor_absoluteY="109dp">
        <com.google.android.material.textfield.TextInputLayout
            android:layout_width="235dp" android:layout_height="wrap_content">
        </com.google.android.material.textfield.TextInputLayout>
        <Button android:id="@+id/button5" android:layout_width="305dp"
            android:layout_height="60dp" android:layout_alignParentTop="true"
            android:layout_marginTop="88dp«  android:text="Envoyer " />
        <com.google.android.material.textfield.TextInputEditText android:layout_width="279dp"
            android:layout_height="42dp" android:layout_alignParentTop="true"
            android:layout_alignParentRight="false" android:hint="votre nom" />
        <ProgressBar android:id="@+id/progressBar" style="?android:attr/progressBarStyleHorizontal"
            android:layout_width="218dp" android:layout_height="49dp"
            android:layout_alignParentBottom="true" android:layout_marginBottom="379dp" />
        <Button android:id="@+id/button6" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:layout_alignParentLeft="true"
            android:layout_alignParentBottom="true" android:layout_marginLeft="286dp"
            android:layout_marginBottom="375dp" android:text="Abbonner" />

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



ScrollView et HorizontalScrollView

- En général utilisés pour ajouter des ascenseurs à un conteneur.
- Ne peuvent contenir qu'un seul élément (le plus souvent un conteneur).
- Propriétés supplémentaires :
  - android:fillViewport="b" (où b vaut true ou false) indique si le contenu doit être étiré pour occuper la place disponible ou pas
- ATTENTION : En raison de l'écran tactile le défilement porte sur l'élément le plus externe (le plus haut dans l'arbre de l'interface)

Exemple scrollview

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <RelativeLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        tools:layout_editor_absoluteX="292dp"
        tools:layout_editor_absoluteY="109dp">


        <ScrollView
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical" />
        </ScrollView>
    </RelativeLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Fin de la séquence 2