ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR

Membre de HONORIS UNITED UNIVERSITIES

Module :

Développement mobile native :

Chapitre 3:
Interface utilisateur sous Android

Pr. Zakia EL UAHHABI

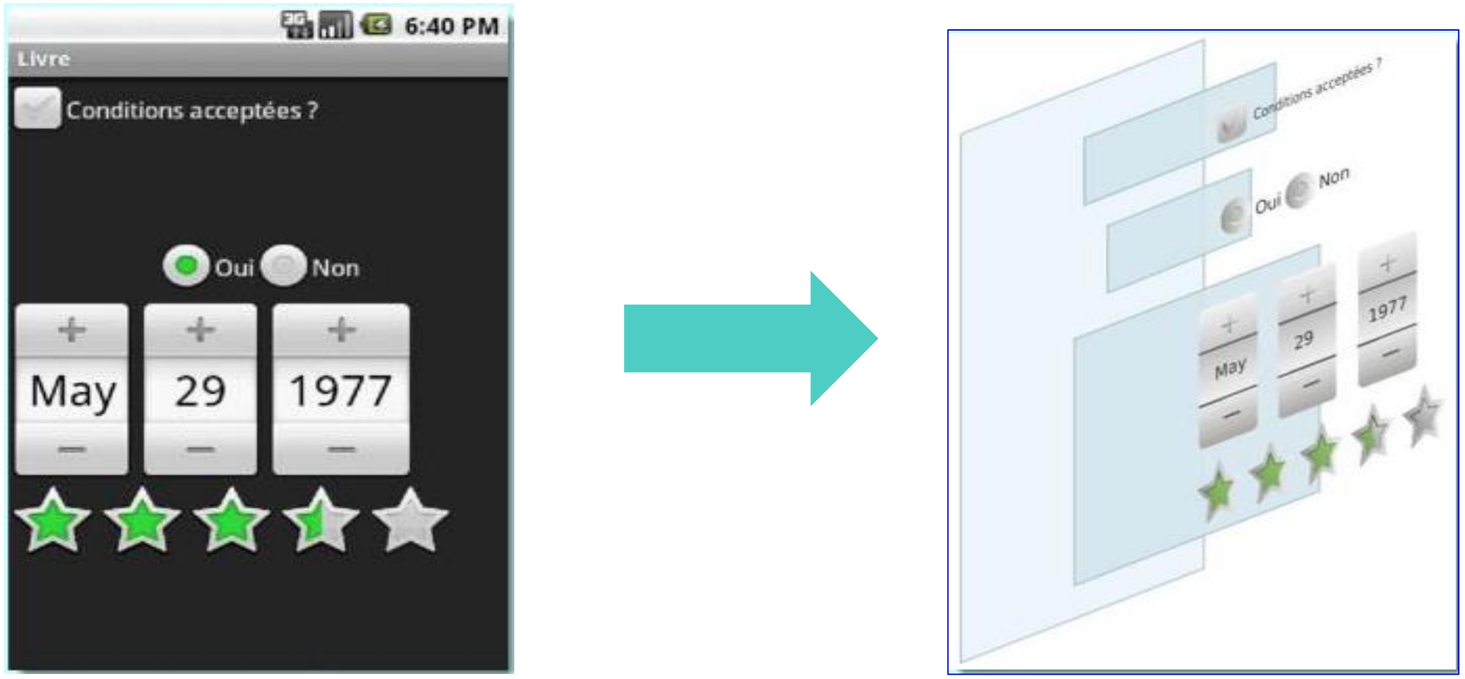
Interface Graphique

- Les interfaces (Layout) permettent de dessiner la vue tel qu'elle doit s'afficher à l'utilisateur.
- Android recommande l'utilisation des fichiers XML pour définir les interfaces.
- Avec Android Studio, vous avez la possibilité d'utiliser l'outil de conception graphique ou d'éditer et de modifier directement le fichier XML.
- Une interface XML peut être associée à une activité (vue) grâce à la méthode setContentView (identifiant)

Exemple : setContentView(R.layout.login);

Interface Graphique

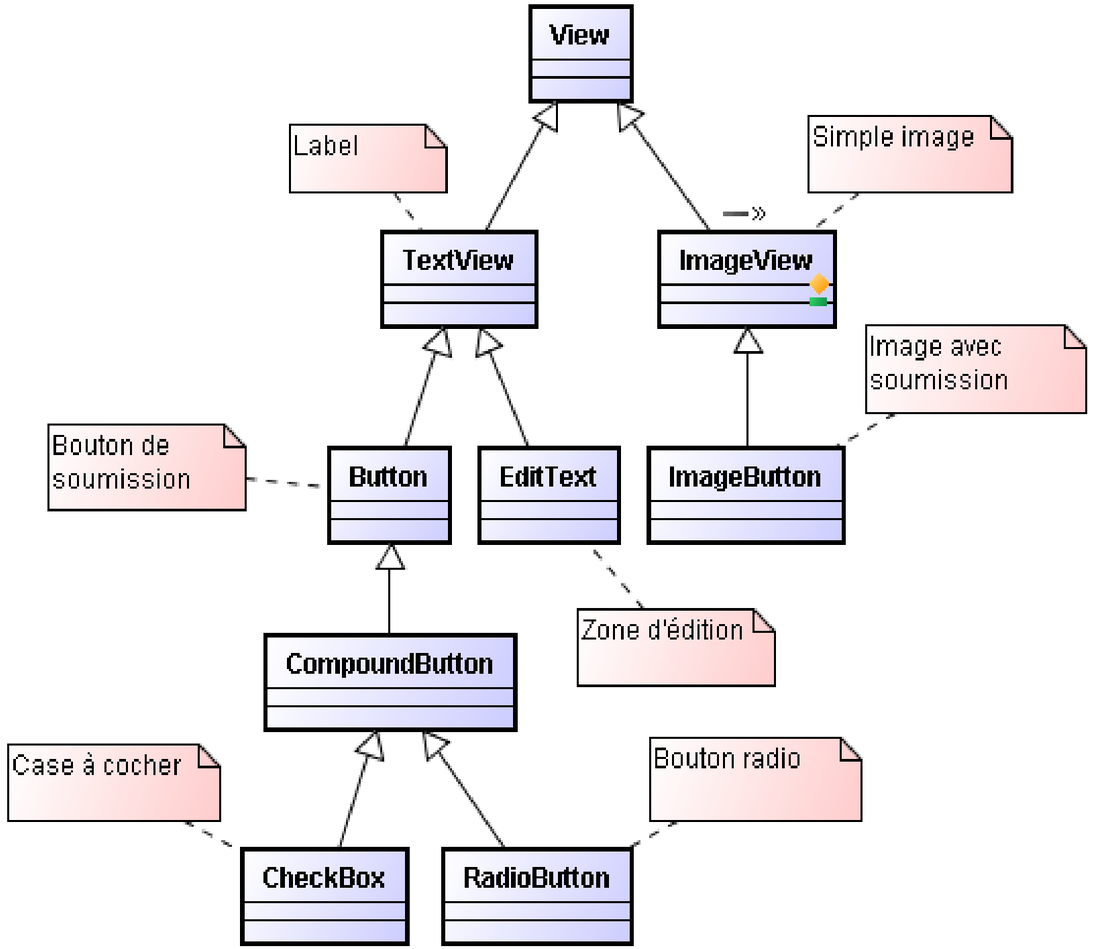
- Construire une interface, c'est mettre des composants graphiques les uns à l’intérieur des autres.



Interface Graphique

- Tous les composants graphiques (boutons, images, cases à cocher, etc.) d'Android héritent de la classe View.

- Un composant graphique est une instance d'une classe particulière.



Interface Graphique

Widget

- Un widget: c'est un terme résultant de la contraction des mots Windows et gadget.
- Concrètement c'est un composant d'interface graphique.

Exemple:

Champs de texte, Champs de saisie, boutons...

Interface Graphique

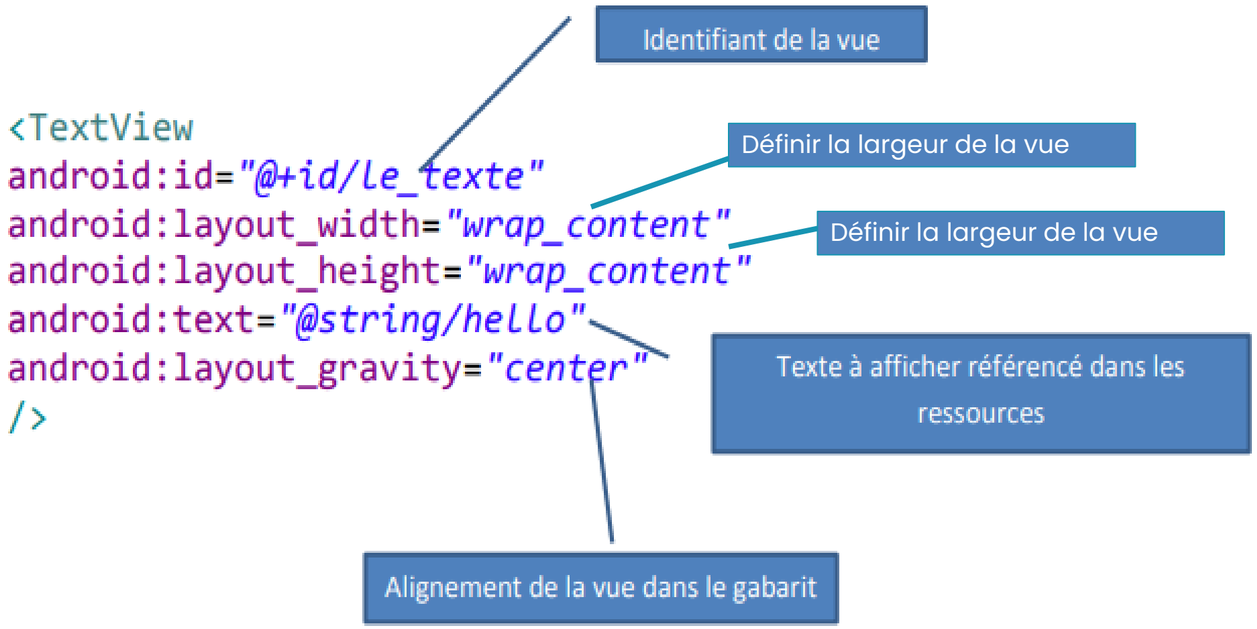
Vues de base (Widgets)

- ☞ TextView : pour les labels texte;
- ☞ EditText : Champ de saisie;
- ☞ ListView : Liste de vues horizontales;
- ☞ Button : bouton standard;
- ☞ CheckBox : case à cocher;
- ☞ Spinner : Liste déroulante
- ☞ ImageView : image
- ☞ RadioButton : radio (choix exclusif)
- ☞ TimePicker : Choix de date
- ☞ ProgressBar : Bar de progression
- ☞ …….

Interface Graphique

Widgets

- Une vue peut être déclarée dans un fichier XML



Interface Graphique

Widgets

- Une vue peut être aussi crée dynamiquement

```
public class Activity2 extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout gabarit = new LinearLayout(this);
        gabarit.setGravity(Gravity.CENTER); // centrer les éléments
graphiques
        gabarit.setOrientation(LinearLayout.VERTICAL); // disposi-
tion horizontal !
        TextView texte = new TextView(this);
        texte.setText("Programming creation of interface !");
        gabarit.addView(texte);
        setContentView(gabarit);
    }
}
```

Définition du Layout à construire

Définition de la gravité du composant

Définition du composant Text.

Affectation du texte à afficher par le composant Texte.

Ajout du composant Text au layout.

Permet de déterminer quel fichier layout à utiliser.

Interface Graphique

Interaction entre Java et XML

- Si le visuel est défini dans un fichier XML, toute la partie interaction est définie dans des classes java.

Interface Graphique

Interaction entre Java et XML

- Pour effectuer des modifications graphiques sur un composant, il faut utiliser un identifiant XML pour récupérer une référence vers ce composant.

- En java, il faut utiliser la méthode **findViewById** pour récupérer le composant associée à un identifiant.

Interface Graphique

Interaction entre Java et XML

- Les vues déclarées dans les fichiers XML sont crées automatiquement par le système et peuvent être récupérées dans le code Java. On utilise la simple méthode finfViewById permettra de récupérer une référence vers une vue nommée par exemple myView dans le fichier xml.
- Pour modifier le texte présent sur le bouton, on utilise la méthode setText

```
View v = findViewById(R.id.myView);
```

Exemple:

```
Button b = (Button)findViewById(R.id.btnCancel);
 b.setText("texte modifie depuis java")
```

Interface Graphique

Widgets (contenus)

- Il est possible d'accéder aux propriétés des widgets en lecture et/ou en écriture

```
EditText edit = (EditText)findViewById(R.id.nom);
//On y met une valeur en dure
edit.setText("Voici ta nouvelle valeur");
//on y met une valeur dans les ressources
edit.setText(R.string.hello_world);
//On récupère le contenu
edit.getText();
```

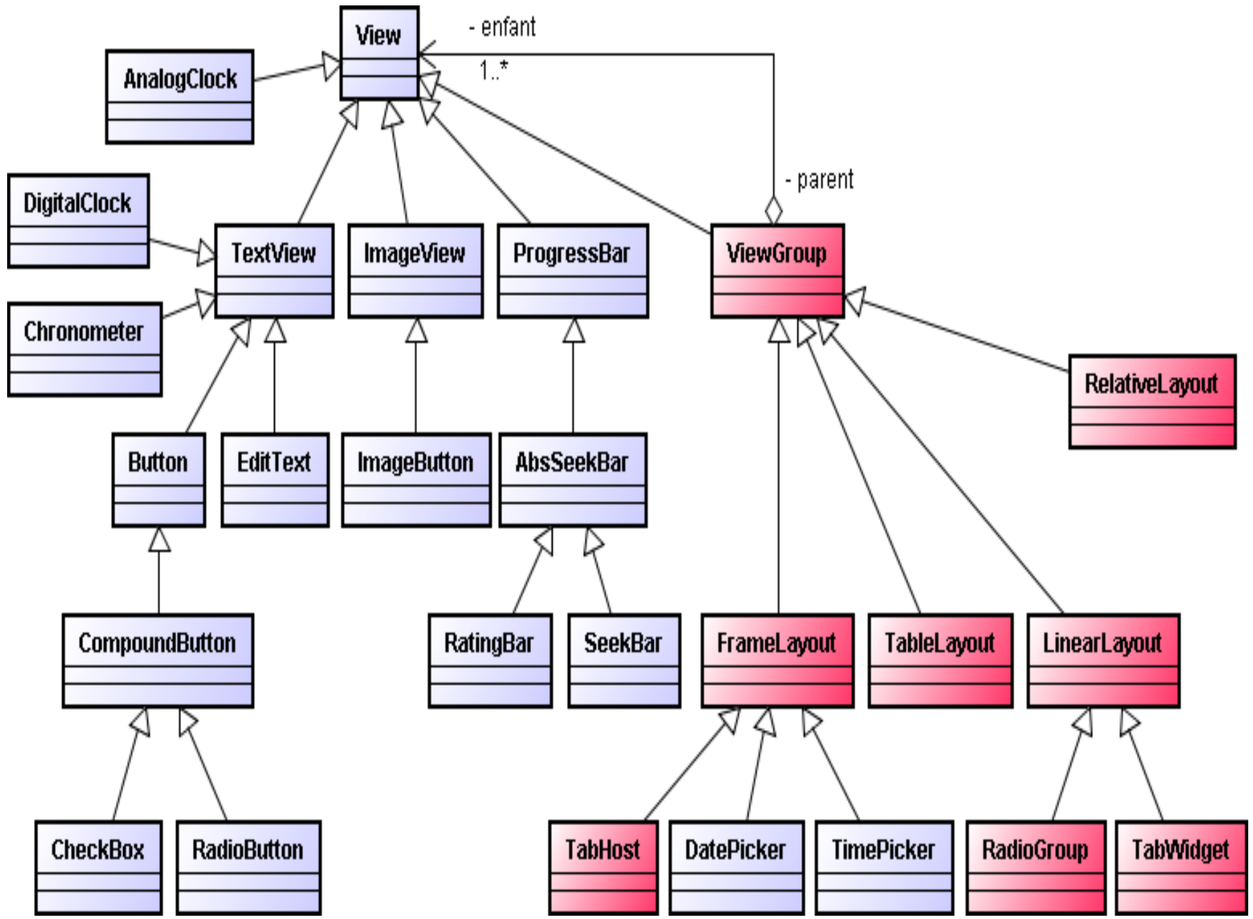
L'élément devient accessible

- **Dans un fichier JAVA**
  R.id.nom_identifiant
- **Dans un fichier XML**
  @id/nom_identifiant

Interface Graphique

Positionnement des vues avec les gabarits

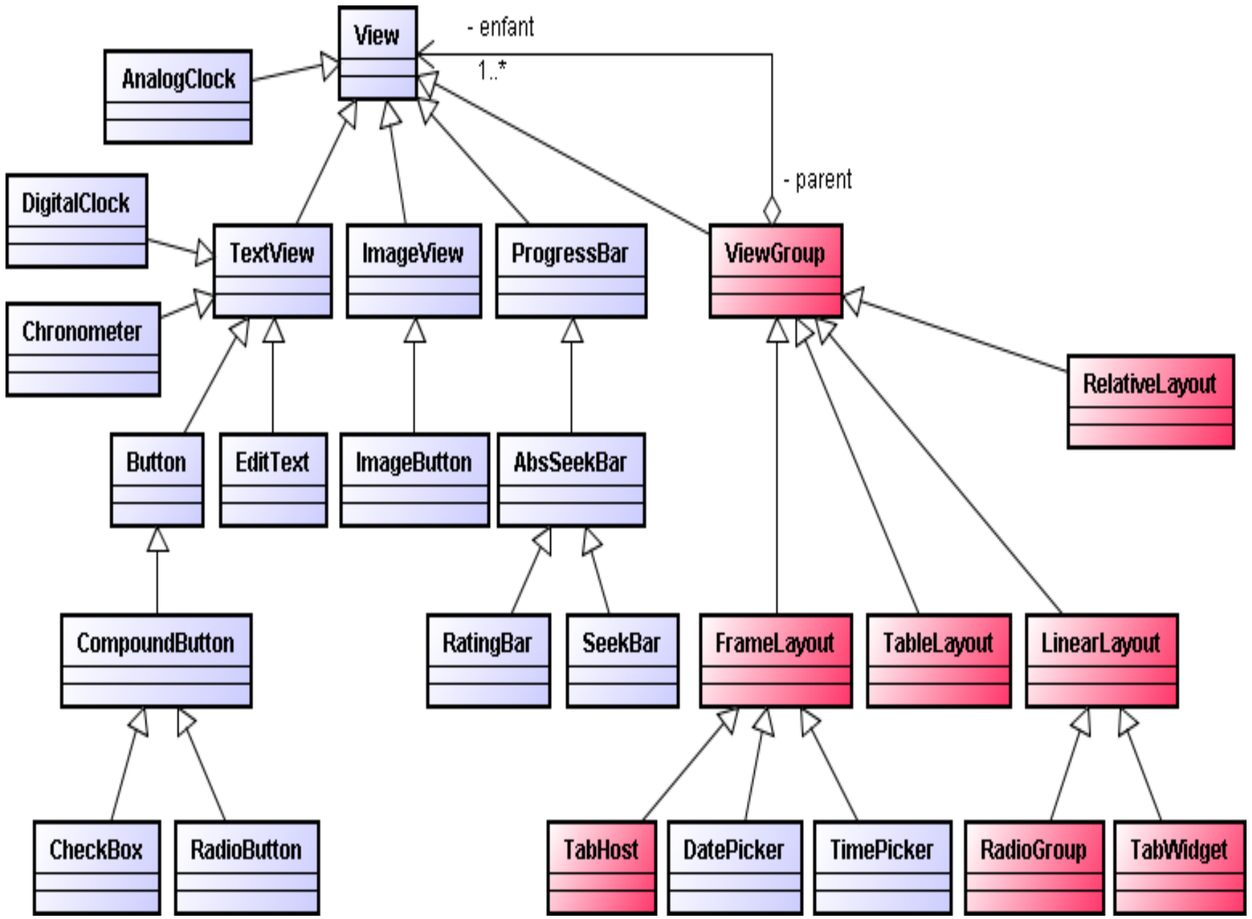
- Android permet de regrouper plusieurs vues dans une structure arborescente à l'aide de la classe ViewGroup.
- Il existe des vues particulières permettant de contenir d'autres vues et de les positionner : les gabarits (layouts).
- Android contient plusieurs dizaines de composants d'interface :
  - des vues : TextView, Button, EditText. .
  - des groupes : LinearLayout, RelativeLayout, TableLayout. . .
- On peut créer des mises en forme évoluées à l'aide de l'imbrication des gabarits.



Interface Graphique

Positionnement des vues avec les gabarits

- Les gabarits (Layouts) proposent une prédispositions des objets graphiques:
  - LinearLayout
  - RelativeLayout
  - TableLayout
  - FrameLayout
  - GridLayout
  - ConstraintLayout



Interface Graphique

Positionnement des vues avec les gabarits

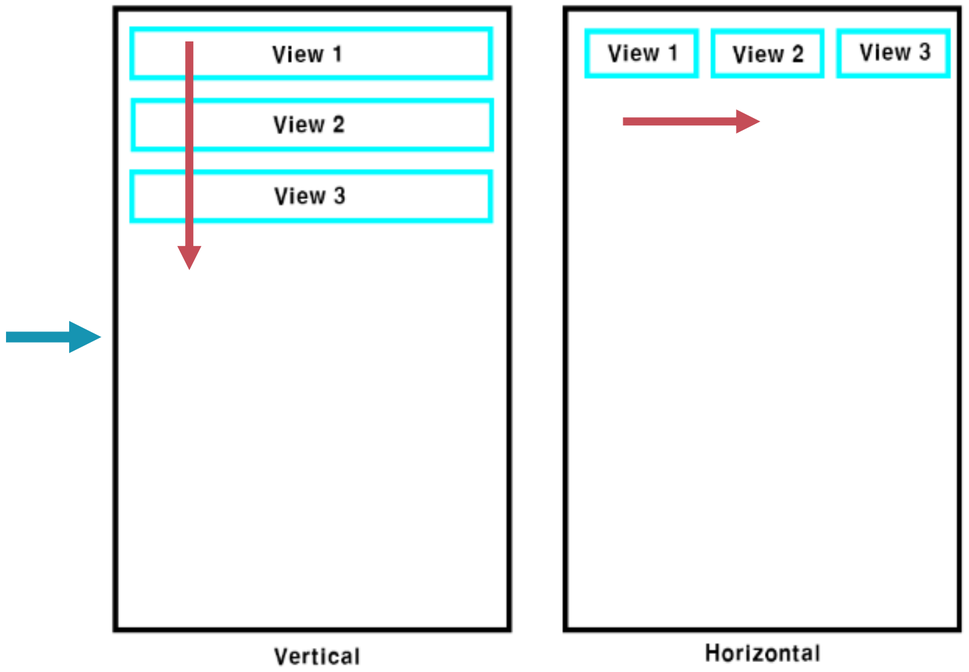
- Les principaux Layout Android sont:
  - ☞LinearLayout
  - ☞RelativeLayout
  - ☞ConstraintLayout
  - ☞TableLayout
  - ☞FrameLayout

Interface Graphique

LinearLayout

- LinearLayout = Placer les éléments sur une ligne;
- On distingue deux types de ce conteneur selon son orientation.

Vertical
affichage des éléments de **haut en bas**



Horizontal
affichage des éléments de gauche à droite

Interface Graphique

LinearLayout

- Principale propriété d'un LinearLayout : l'orientation.

- L'orientation indique si le LinearLayout présente ces contenus sur une ligne (horizontalement) ou sur une colonne (verticalement). On utilise l'attribut suivant:

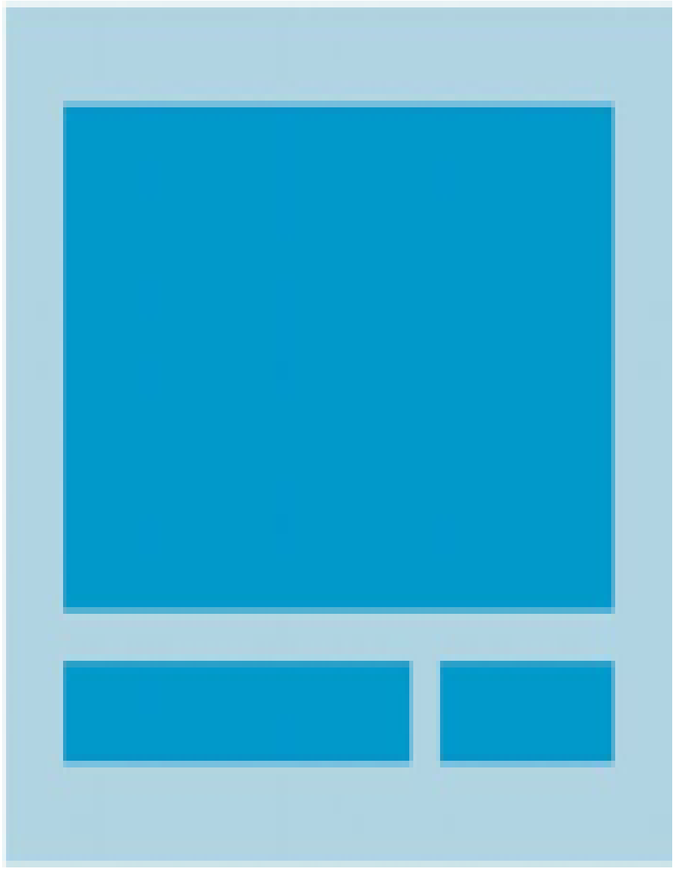
  android:orientation

  (Les valeurs possibles pour cette propriété sont **vertical** et **horizontal**)

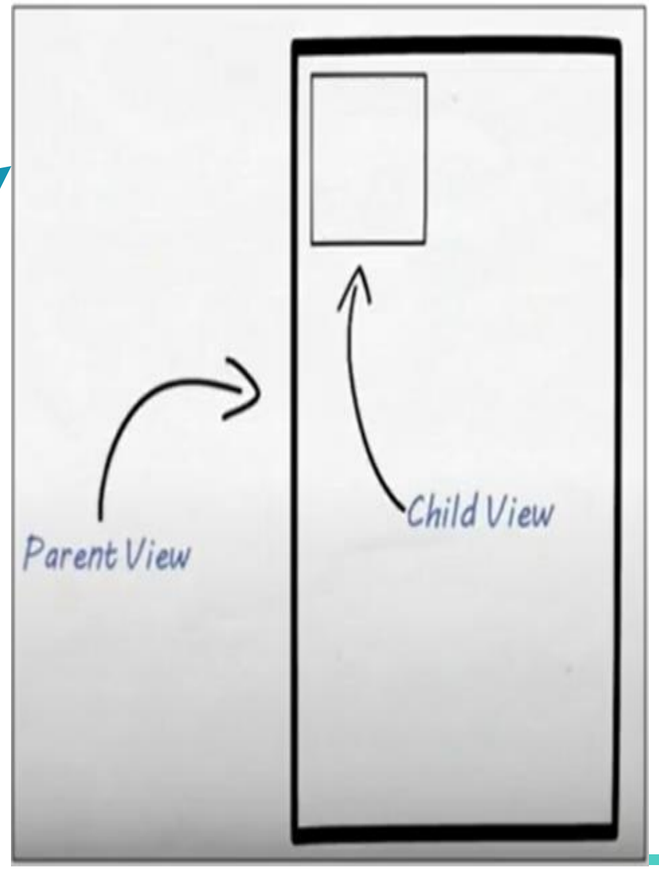
Interface Graphique

RelativeLayout

RelativeLayout
=
Placer les composants les uns par rapport aux autres.



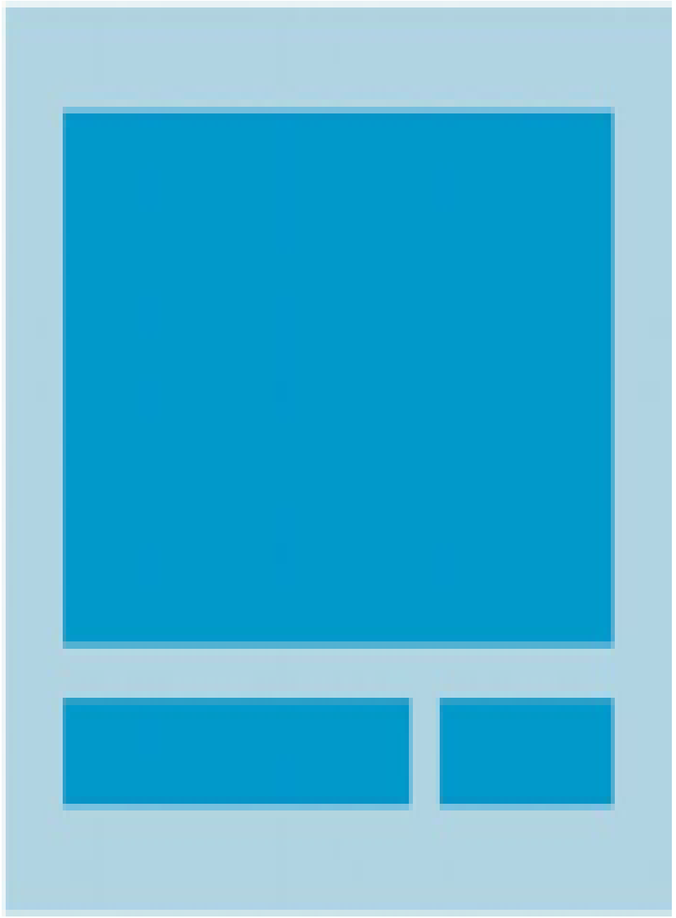
Il est même possible de placer les composants par rapport au RelativeLayout parent.



Interface Graphique

RelativeLayout

RelativeLayout
=
Placer les composants les uns par rapport aux autres.



- Positionnement par rapport à une autre vue.

La valeur à mettre dans ces contraintes est la référence à l'identifiant d'une autre vue, @id/autre_vue. Chaque vue concernée doit donc définir un identifiant.

– « collé au bord opposé » : layout_toRightOf, layout_toLeftOf, layout_above et layout_below. Par exemple, il faut coller le bord droit de cette vue au bord gauche de l'autre vue.

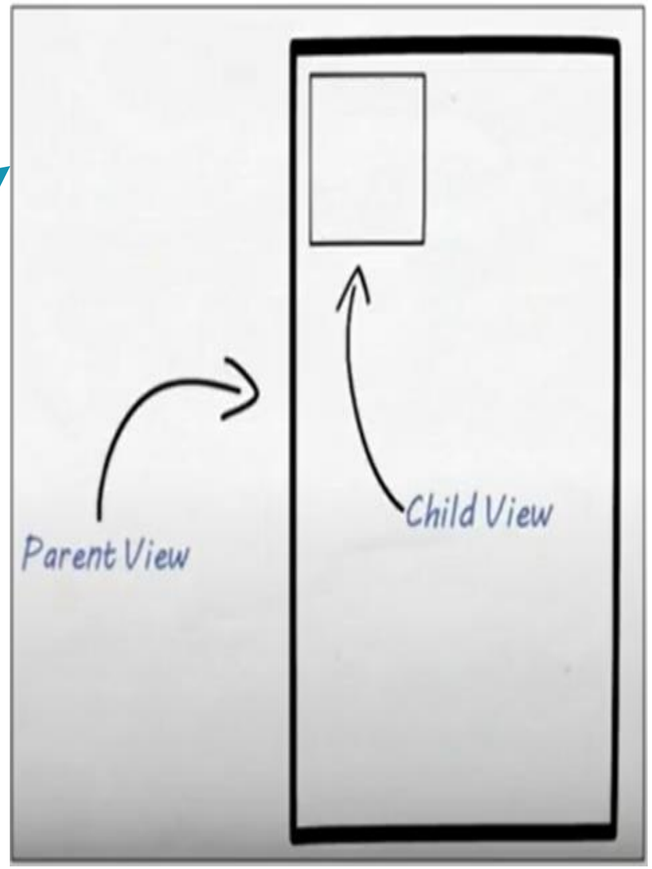
android:layout_toLeftOf="@id/button1«

– « bords alignés » : layout_alignXXX avec XXX valant Top, Bottom, Left et Right. Cette vue aligne son bord avec celui de l'autre vue.

Interface Graphique

RelativeLayout

Il est même possible de placer les composants par rapport au RelativeLayout parent.



Positionnement par rapport au layout.

La valeur de ces contraintes sont de simples booléens et on ne les mentionne que quand ils sont vrais.

– « collé au bord du parent » : layout_alignParentXXX avec XXX valant Top, Bottom, Left et Right,

– « centré dans le parent » : layout_centerInParent, layout_centerHorizontal et layout_centerVertical.

Exemple:
android:layout_alignParentRight : si true, le layout enfant est collé au bord droit de layout parent.

Interface Graphique

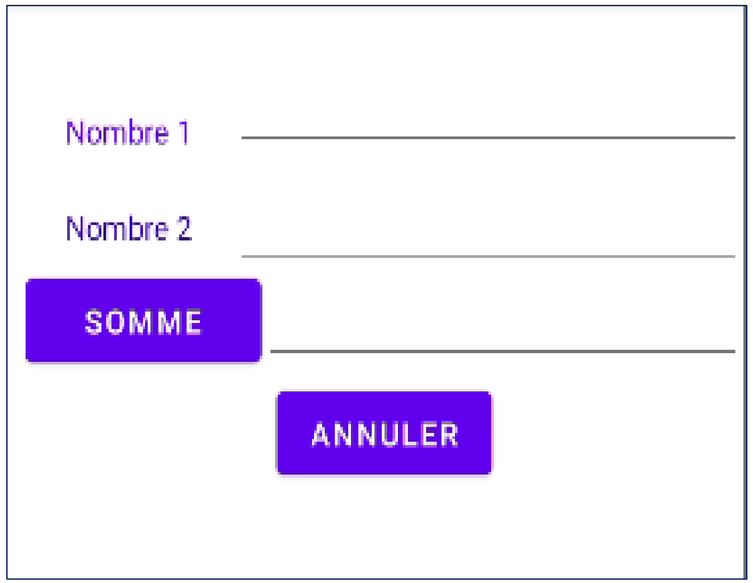
RelativeLayout

- Les Views dans un RelativeLayout peuvent utiliser les attributs :
  - android:layout_alignParentRight : si true, le layout enfant est collé au bord droit de layout parent.
  - android:layout_centerVertical : si true, la View est centrée verticalement à l'intérieur de la vue parente
  - android:layout_below : la vue est en dessous de la View indiquée (par son id)
  - android:layout_toRightOf : le coté gauche de la vue est à droite de la View indiquée (par son l'id)

Interface Graphique

Exercice

- Créer l’interface ci-dessous en faisant l’imbrication entre RelativeLayout et LinearLayout



Interface Graphique

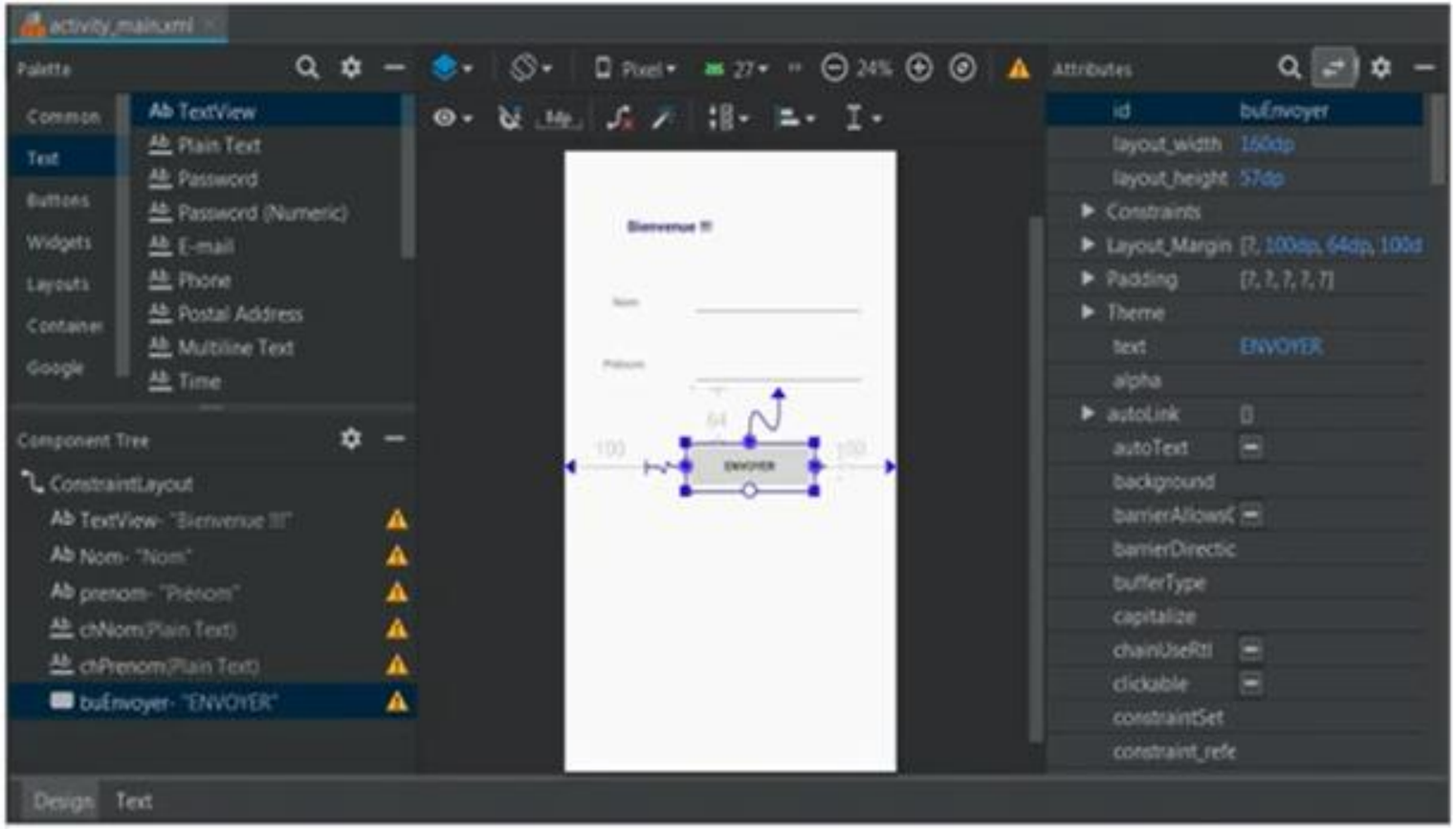
ConstraintLayout

ConstraintLayout = permet de positionner les vues en fonction de contraintes sur les positions fixées (chiffres fixés) par l'utilisateur.

Dans les grandes lignes, le ConstraintLayout permet de placer ses widgets de façon relative aux autres mais en se basant sur des "contraintes" de positionnement en chiffres fixées par l'utilisateur.

Interface Graphique

ConstraintLayout



Interface Graphique

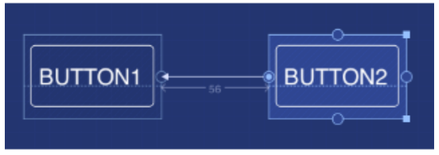
ConstraintLayout

Les attributs pour mettre en place les contraintes :

☞Pour positionner une vue à une certaine distance d'une autre, on utilise les attributs **layout_marginXXX** avec **XXX valant Top, Bottom, Left, Right.**

☞ La forme générale des contraintes est **layout_constraintXXX_toYYYOf** avec **XXX et YYY valant Top, Bottom, Left, Right ou Baseline**.
Avec X le côté du widget où l'on place la contrainte, et Y le côté du widget de destination.

☞ Cela donne des attributs comme layout_constraintTop_toTopOf, layout_constraintLeft_toRightOf, etc. Le code Baseline fait aligner sur la base du texte des vues.



Interface Graphique

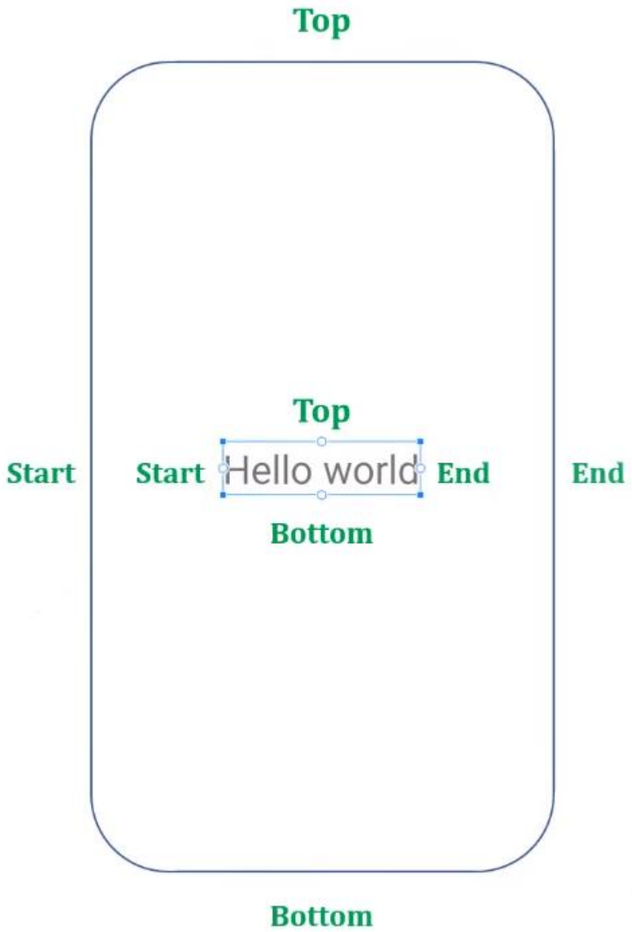
ConstraintLayout

Les attributs pour mettre en place les contraintes :

☞ La forme générale des contraintes est **layout_constraintXXX_toYYYOf** avec **XXX et YYY valant Top, Bottom, Start ou End**.

Exemple:

si vous avez un bouton B et que vous le contraintes avec **layout_constraintStart_toEndOf="@+id/otherButton"** , le côté de départ du bouton B sera aligné avec le côté de fin de **otherButton**.



Interface Graphique

ConstraintLayout

Tous les attributs prennent pour valeur une référence d'identifiant (et non pas un booléen comme certains avec les RelativeLayout).

Exemple

on écrit layout_alignParentTop="true" dans un RelativeLayout, mais layout_constraintTop_toTopOf="@id/id_du_layout" dans un ConstraintLayout

on peut aussi mettre une valeur spéciale pour indiquer le layout : "parent".

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="5dp"
    android:text="tout le monde"
    app:layout_constraintLeft_toRightOf="@+id/bonjour"
    app:layout_constraintBaseline_toBaselineOf="@+id/bonjour" />
```

Interface Graphique

TableLayout

- TableLayout permet d'organiser les éléments en tableau, comme avec l'élément <table> en HTML, mais sans les bordures.
- Il utilise pour cela l'élément <tableRow> qui déclare une nouvelle ligne à l'intérieur du TableLayout globale.
- Les cellules sont définies par les composants qu'on ajoute aux lignes

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_etabli" android:layout_width="match_parent"
    android:layout_height="match_parent" android:stretchColumns="1"
    android:background="#ffffffff"
    tools:context="com.example.admin.etablissements.Etablissement_Activi

    <TableRow>
        <TextView android:text="Name:" android:textSize="20dp" />
        <EditText android:id="@+id/name" android:background="#ffffff77"/
    </TableRow>

    <TableRow>
        <TextView android:text="Address:" android:textSize="20dp"/>
        <EditText android:id="@+id/addr" android:background="#ffffff88"
    </TableRow>
    <TableRow>
        <TextView android:text="Type:" android:textSize="20dp" />
        <RadioGroup android:id="@+id/types" >
            <RadioButton android:id="@+id/ingenieurie" android:text="Ingenieurie" />
            <RadioButton android:id="@+id/science_technique"
                android:text="Science_Technique" />
            <RadioButton android:id="@+id/droit_economie" android:text="Droit_Economie" />
        </RadioGroup>
    </TableRow>

    <Button android:id="@+id/save" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="Save"/>
</TableLayout>
```

Interface Graphique

Dispositions de Layouts et Views (Paramètres de disposition)

- La plupart des groupes utilisent des paramètres de placement sous forme d'attributs XML.
- Par exemple, telle vue à droite de telle autre, telle vue la plus grande possible, telle autre la plus petite.

- Ces paramètres sont de deux sortes :
  - ceux qui sont demandés pour toutes les vues,

android:layout_width,
android:layout_height

  - ceux qui sont demandés par le groupe englobant et qui en sont spécifiques, comme:

android:layout_alignParentBottom,
android:layout_centerInParent. . .

Interface Graphique

Dispositions de Layouts et Views (Paramètres de disposition)

- Toutes les vues doivent spécifier les deux attributs suivants:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="right"
    android:text="Hello"/>
```

android:layout_width = largeur de la vue
android:layout_height = hauteur de la vue

- Ils peuvent valoir :

= **une valeur exacte** de pixel (125px pour 125 pixels) : c'est fortement déconseillé
="**wrap_content**" : force la vue à prendre la taille de son contenu;
="**match_parent**" : (anciennement fill_parent avant l'API 8) spécifie que toute la vue va automatiquement prendre la même taille que son parent.

android:gravity: Positionnement du contenu de l'élément (left, right, …)
android:layout_gravity: positionnement de l'élément dans son conteneur (left, right, …).
android:background : pour attribuer une couleur prédéfinie comme fond à chaque vue.

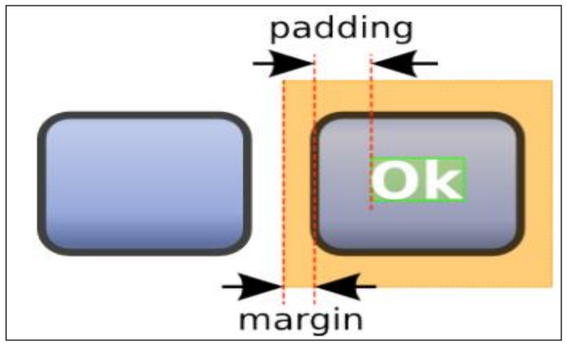
Interface Graphique

Dispositions de Layouts et Views (Autres paramètres géométriques)

- Il est possible de modifier l'espacement des vues :
  - Padding: espace entre le texte et les bords, géré par chaque vue,
  - Margin: espace autour des bords, géré par les groupes,

Exemple:

android:layout_marginLeft=‘5dp’



Interface Graphique

TextView (Propriétés)

TextView Peut servir de zone de texte non éditable et dans ce cas, sert souvent pour présenter les widgets.

- Propriétés importantes :
  - android:text : le texte du TextView (Ex. android:text="Hello World !";)

    En général le texte sera spécifié à l'aide d'une ressource

    Ex. android:text="@string/idDeMonTexte";
  - android:typeface : le type de police utilisée;
  - android:textStyle : modifie le style du texte (italic pour l'italique, bold_italic pour gras et italique, ...);
  - android:textColor pour la couleur d'affichage du texte. Les valeurs sont en hexadécimal en unité RGB (par exemple #FF0000 pour le rouge)

Interface Graphique

EditText (Propriétés)

EditText permet de mettre en place des champs de saisi.

- **Quelques attributs possibles**
  - android:hint permet d'afficher une étiquette de texte descriptive dans l'élément lorsqu'il est vide

    Ex. android:hint="login"

  - android:inputType permet de préciser le type de clavier à utiliser (text, textPassword, number, …)
    Plusieurs valeurs possibles, séparée par |

    Ex. android:inputType="textPassword"

  - android:textColor pour la couleur d'affichage du texte. Les valeurs sont en hexadécimal en unité RGB (par exemple #FF0000 pour le rouge)
  - android:ems : Il définit la largeur d'un EditText

Interface Graphique

ImageView (Propriétés)

ImageView permet d'utiliser des images dans l'interface.

- **Quelques attributs possibles**
  - app:srcCompat

    Pour indiquer l'image à utiliser
  - androind:contentDescription

    Pour spécifier une petite description de l'image

Interface Graphique

Association entre l'interface et le fichier de contrôle

Associer la première vue graphique (activity_main.xml) à l'activité principale de l'application (MainActivity.java):

La méthode onCreate définit ce qui doit être affiche sur l'écran:

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

Interface Graphique

Views (Gestion des évènements)

Les évènements permettent de gérer les actions utilisateurs sur les vues:
Pour gérer les évènements sur les vues, il suffit d'ajouter un écouteur (listener) en utilisant une de ces méthodes suivantes :

- setOnClickListener(View.OnClickListener) associe un écouteur d'événements aux clics sur la vue
- setOnLongClickListener(View.OnLongClickListener) associe un écouteur d'événements aux clics longs sur la vue
- setOnKeyListener(View.OnKeyListener) associe un écouteur d'événements aux actions clavier sur la vue
- setOnTouchListener(View.OnTouchListener) associe un écouteur d'événements aux touchés sur la vue

Interface Graphique

Evènement Toast

- La classe **Toast** permet de créer un texte qui apparait en premier plan puis disparait au bout d'un temps donné. A considérer comme un message d'information ou d'avertissement.

- La classe **Toast** permet de créer un message avec la méthode makeText(), cette dernière prend 3 paramètres:
  - ☞ Le contexte de l'application (this ->depuis une activité)
  - ☞ le message à afficher,
  - ☞ La durée d'affichage (la durée pourra prendre deux valeurs:

    Toast.LENGTH_SHORT ou Toast.LENGTH_LONG).

Interface Graphique

Evènement Toast

Exemple:

```
Toast.makeText(Context, "Bonjour tout le monde", Toast.LENGTH_SHORT).show();
```

- La méthode makeText construit une instance de Toast avec ces paramètres que vous n'aurez plus qu'à afficher via la méthode show().

Interface Graphique

Views (les évènements)

```
import android.widget.Button;
import android.app.Activity;
import android.widget.EditText;
import android.widget.Button;

public class Main extends MapActivity{
        private EditText zoneText;
        private Button bouton;
        @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
            setContentView(R.layout.main);
            zoneText = (EditText)findViewById(R.id.editText1);
            bouton = (Button)findViewById(R.id.button1);
            bouton.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View v){
                switch(v.getId()){
                    case R.id.button1:
                        zoneText.setText("Hello World");
                }
            }
        }
    }
}
```

1. Accéder au contrôle bouton

2. Assigner au bouton un listener (écouteur d'évènement clic) grâce à "setOnClickListener"

3. Faire appel de la méthode **onClick** de l'interface **OnClickListener** et exécuter l'instruction du bloc. "v.**getId**" récupère l'identification du ou des widgets cliqués.

Interface Graphique

ListViews

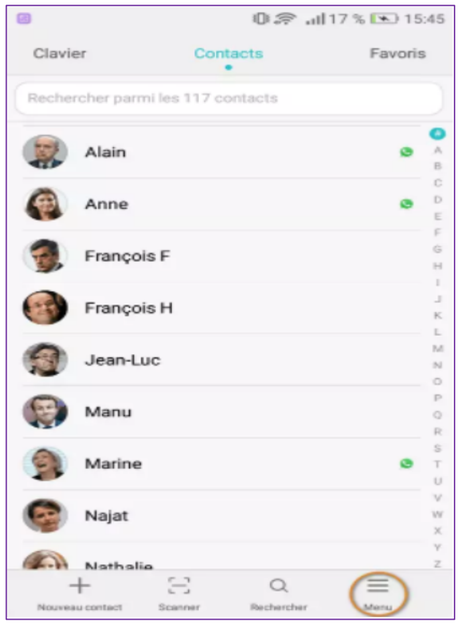
- Au sein d'un layout, on peut implanter une liste que l'on pourra dérouler si le nombre d'éléments est important.

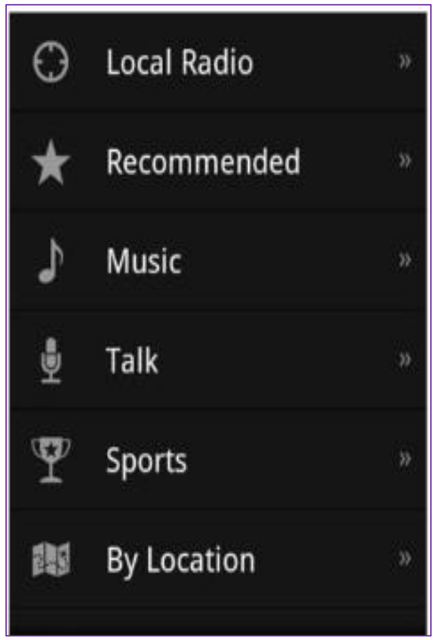
- Si l'on souhaite faire une liste plein écran, il suffit juste de poser un layout linéaire et d'y implanter une **ListView**.

Interface Graphique

ListViews

- Les listViews sont comme toutes les autres listes où nous avons une série d'éléments alignés de manière verticale. Dans Android, ces listes sont utilisées de nombreuses façons, généralement comme des menus pour accéder aux différents écrans.





Interface Graphique

ListViews

- Une ListView est faite à partir d'un groupe de ListItem.
  ListItem est une ligne (row) individuelle dans listview où les données seront affichées.
- Toutes les données dans listview sont affichées uniquement via listItem.
- Android construit certaine forme ListItem différente, appelée le layout prédéfini.

ListView

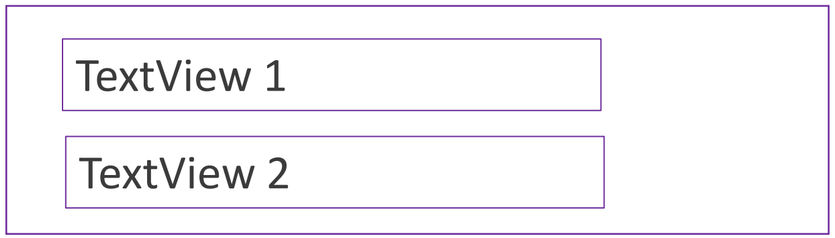
ListItem

Interface Graphique

ListViews

- Une ListItem est une pièce de l'interface qui peut être créée par un nombre de view.

ListItem



Interface Graphique-ListView

Affichage d’une Liste

- **Deux possibilités :**
  - classe **ListActivity**,
  - classe **Activity** de base.
- Ces deux possibilités sont très similaires : un layout contenant un ListView pour l'activité, un layout pour les items de la liste et un adaptateur pour accéder aux données.
- La ***ListActivity*** prépare un peu plus de choses pour gérer les sélections d'items, tandis qu'avec une simple Activity, c'est à nous de tout faire

Interface Graphique-ListView

Affichage d’une Liste

Utilisation de la classe ListActivity

- Créer une instance de **ListActivity**, qui intègre automatiquement :
  - ☞ Une vue, instance de la classe **ListView**
  - ☞ Un **adaptateur**, instance d’une implémentation de **ListAdapter**

Interface Graphique-ListView

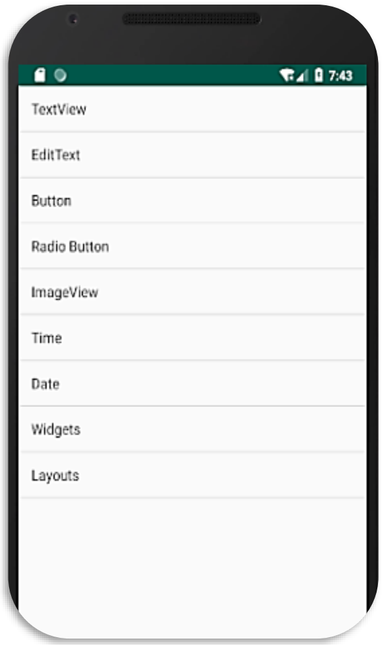
Affichage d'une Liste

Utilisation de la classe ListActivity

```
public class MainActivity extends ListActivity {

    private String [] listVues = new String []
            {"TextView", "EditText", "Button", "Radio Button",
                    "ImageView", "Time", "Date", "Widgets", "Layouts" };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);
        ListView mLisview= getListView();
        ArrayAdapter<String> listAdapter=new ArrayAdapter <String>(
                context: this,
                android.R.layout.simple_list_item_1,
                listVues
        );
        mLisview.setAdapter(listAdapter);
    }
}
```



Interface Graphique-ListView

Types de ListViews

- **Liste statique**: le contenu est fixé et connu avant l'exécution de l'application. Déclarer les éléments de la liste dans le fichier de ressources string.xml

```
<!-- res/values/strings.xml -->
<resources>
<string-array name="components">
      <item>Activity </item>
      <item> Service </item>
      <item> Content Provider </item>
      <item>Intent </item>
   </string-array>
</resources>
```

```
    <!-- res/layout/activity_main.xml -->
    <ListView
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:id="@+id/maylist"
       android:entries="@array/components">
    </ListView>
```

Interface Graphique-ListView

Types de ListViews

- **Liste dynamique**: le contenu est lu ou généré pendant l'exécution du programme.il provient d'un fichier de données, d'Internet, etc
- Le remplissage de la liste doit être défini dans le code Java.

Interface Graphique-ListView

Liste dynamique

Réalisation d'une liste simple se fait en 4 étapes:

1. Déclaration de la liste dans le fichier XML de l'activité.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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"
    android:orientation="vertical"
    tools:context=".MainActivity2">

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

Interface Graphique-ListView

Liste dynamique

- **Réalisation d'une liste simple se fait en 4 étapes:**

  2.Déclarer le tableau des données qui va contenir les éléments de la liste (Source de données).

  Exemple

```
public class listView1 extends AppCompatActivity {
2 usages
ListView ls;
1 usage
String[] d=new String[]{"item1","item2","item3","item4"};
```

Interface Graphique-ListView

Liste dynamique

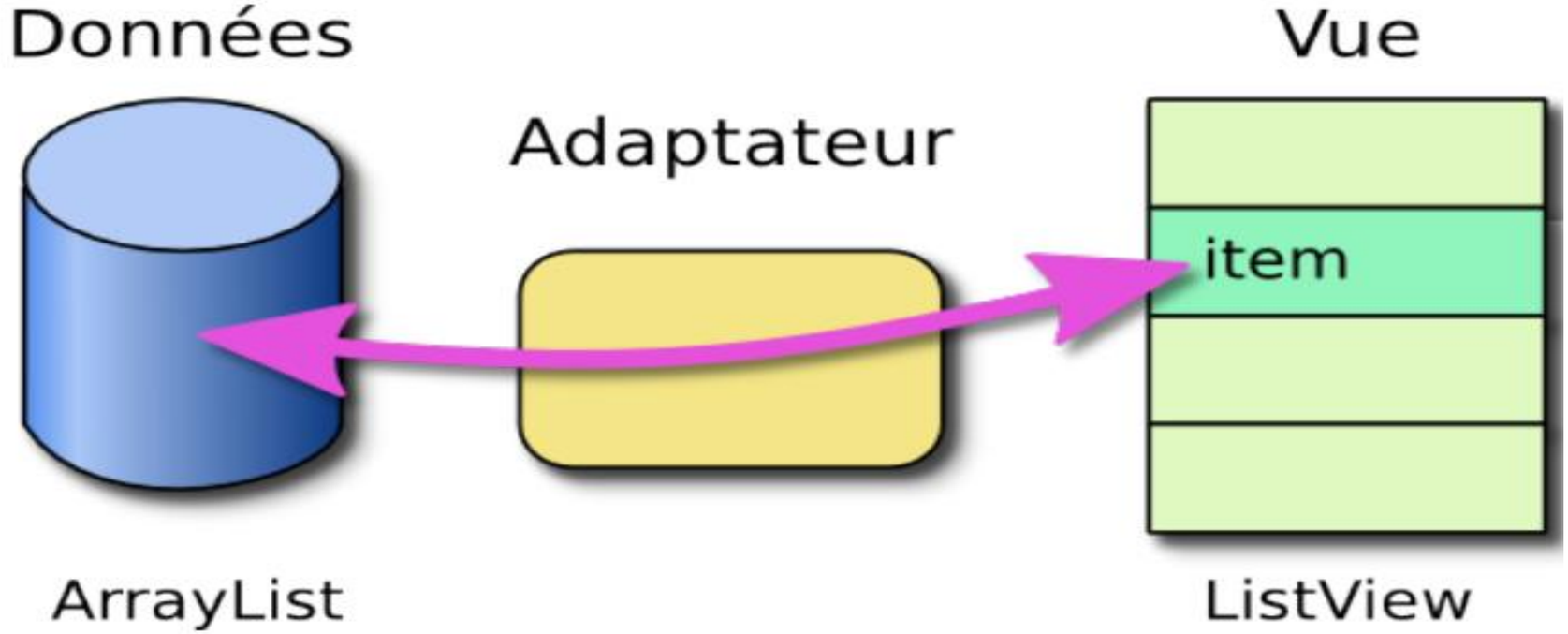
- **Réalisation d'une liste simple se fait en 4 étapes:**

3. Instancier le "Adapter" qui va associer les données et les vues.
Dans la classe Java de votre activité, créer le Adapter de la liste.
4. Attribuer le "Adapter » à votre ListView créé.

```
public class listView1 extends AppCompatActivity {
2 usages
ListView ls;
    @SuppressLint("MissingInflatedId")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list_view1);
        ls=findViewById(R.id.list);
        String[] source=new String[]{"item1","item2","item3","item4"};
        ArrayAdapter<String> ad=new ArrayAdapter<>( context: this, android.R.layout.simple_list_item_1,source);
        ls.setAdapter(ad);
    }
}
```

Adaptateurs(Adapter)

- C'est un pont entre des Views (par exemple listView) et les données sous-jacentes pour ce View.
- un Adapter gère des données et adapte les données dans les lignes individuelles (ListItem) du View.



Adaptateurs(Adapter)

Adaptateurs prédéfinis

- Android propose quelques classes d'adaptateurs prédéfinis, dont :
  - ArrayAdapter pour un tableau simple (liste dynamique),
  - SimpleCursorAdapter pour accéder à une base de données,

Adaptateurs(Adapter)

ArrayAdapter<Type> pour les listes

- Cet adaptateur est utilisé quand la source de données est un tableau (ou ArrayList).Il permet d'afficher les données d'un tableau, mais il est limité à une seule chaîne par item, et il la place dans un objet TextView.

- Syntaxe d'utilisation d'ArrayAdapter:

ArrayAdapter(le contexte, style des items, la source de données);

```
ArrayAdapter<String> nomAdp=new ArrayAdapter<String>(context, item_layout_id,source_données)
```

- **Context:** représente l'activité dans laquelle la AdapterView (ListView) est créée, mettre **this**,
- **item_layout_id :** identifiant du layout des items (élément de la liste),
  Ex. **android.R.layout.simple_list_item_1 qui est un layout prédéfini qui contient un TextView pour chaque élément du tableau ou R.layout.item_filiere.**
- **Source_données: la source de données (un tableau String[], une liste comme ArrayList<String>**

Interface Graphique-ListView

ArrayAdapter<Type> pour les listes

Layouts prédéfinis

- Android contient des layouts pour des éléments de listes simples :
  - android.R.layout.simple_list_item_1;
    C'est un layout qui affiche un seul TextView. Son identifiant est **android.R.id.text1**,
  - android.R.layout.simple_list_item_2;
    C'est un layout qui affiche deux TextView :
    - un titre en grand et un sous-titre. Ses identifiants sont android.R.id.text1 android.R.id.text2.
  - Il suffit de les fournir à l'adaptateur. Il n'y a pas besoin de créer des fichiers XML, ni pour l'écran, ni pour les items.

Adaptateurs(Adapter)

ArrayAdapter<Type> pour les listes

- Cet adaptateur crée sera ensuite associé à la **AdapterView** correspondante, en utilisant la méthode **setAdapter**.
- Exemple:

Etape 1: la création de la source de données (utiliser tableau String[] ou bien ArrayList..).

Etape 2: créer un adaptateur et le lier avec la source de données.

Activité actuelle qui contient la listView.

Format de l'affichage de ListItem.

Nom de la source de données

```
ArrayList<String> myArray=...

ArrayAdapter<String> adapter=
new ArrayAdapter<String>(
this,
androd.R.layout.simple_list_item_1,
myArray);

ListView list=findViewById(R.id.mylist);
list.setAdapter(adapter);
```

Etape 3: Faire la liaison entre la listView et l'adaptateur (méthode setAdapter.

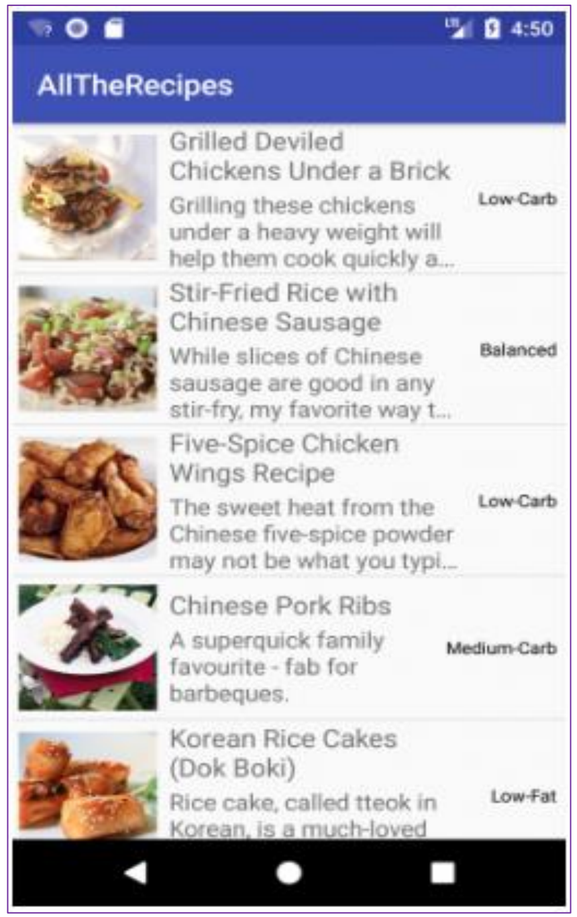
Interface Graphique-ListView Personnalisée

- Pour personnaliser le contenu de la liste, **il faut définir votre propre adaptateur**. Pour cela, on doit suivre les étapes suivantes:

1. Définir un **layout** dans lequel vous définissez un **modèle de l'Item** (une ligne de la liste).
2. Créer **une classe de votre adaptateur** qui hérite ArrayAdapter:

   ☞ redéfinir la méthode getView() pour définir quelle vue doit être renvoyée pour chaque ligne.

   ☞ Utiliser un LayoutInflater qui permet de convertir un layout en XML à un nouvel objet View (ou Layout). Cet objet (v) est utilisable dans le code.



Interface Graphique-ListView Personnalisée

Création d’adaptateur

- La méthode View getView(int position, View convertView, ViewGroup parent) est appelée à chaque fois qu'un item est affiché dans la liste.
- Un LayoutInflater: un objet permettant de convertir les éléments d'un fichier layout XML à un nouvel objet de type View .
- La classe LayoutInflater contient la méthode View inflate(int Id, ViewGroup root)

Interface Graphique-ListView Personnalisée

setContentView(View) & inflate(…)

- Les deux méthodes permettent d'accéder au fichier XML et d'en récupérer la vue:

  ☞ Il est préférable d'utiliser la méthode setContentView(View) si elle est accessible comme c'est le cas dans une activité.

  ☞ La méthode **setContentView(View)** n'existe pas toujours dans les autres classes, dans ce cas, on utilise la méthode inflate(...).