Université Ahmed ZABANA de Relizane

Faculté des Sciences et Technologies

Département d'Informatique

جامعة غليزان
RELIZANE UNIVERSITY

Applications Mobiles

3ème Année Informatique

2021 / 2022

- Chapitre 4 -

Responsable: Dr. Sofiane AMARA

Chapitre 4: Interfaces graphiques et widget

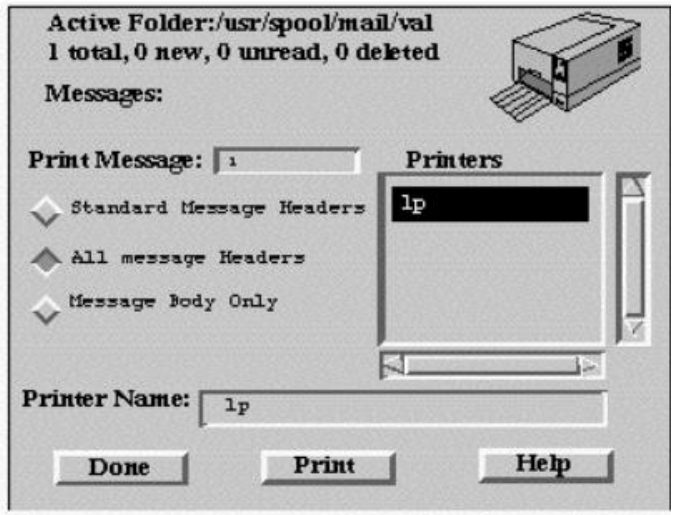
1. **Création des interfaces graphiques**
2. **Gérer les évènements sur les widgets**

1. Création des interfaces graphiques

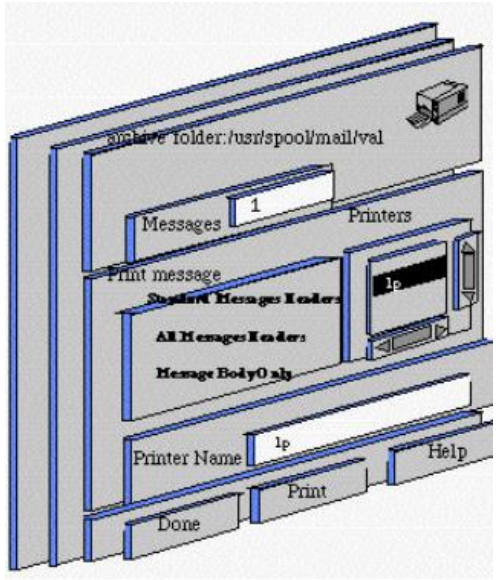
- **La création d'une interface sous Android peut s'effectuer par la création de deux éléments principaux :**
  - **Une définition de l'interface utilisateur (conteneurs, texte, bouton, etc.) dans un fichier XML.**
  - **Une définition du logique métier (comportement de l'interface) dans une classe Activity (classe JAVA)**

1.1. Premier principe des IHM (1/4)

- **Quand on voit cette fenêtre:**

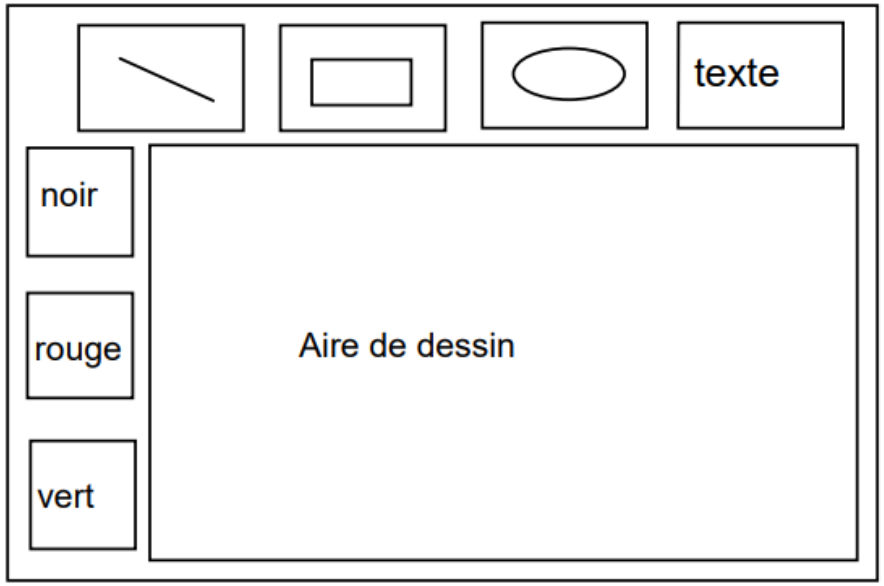


- **C'est qu'on a programmé ceci:**

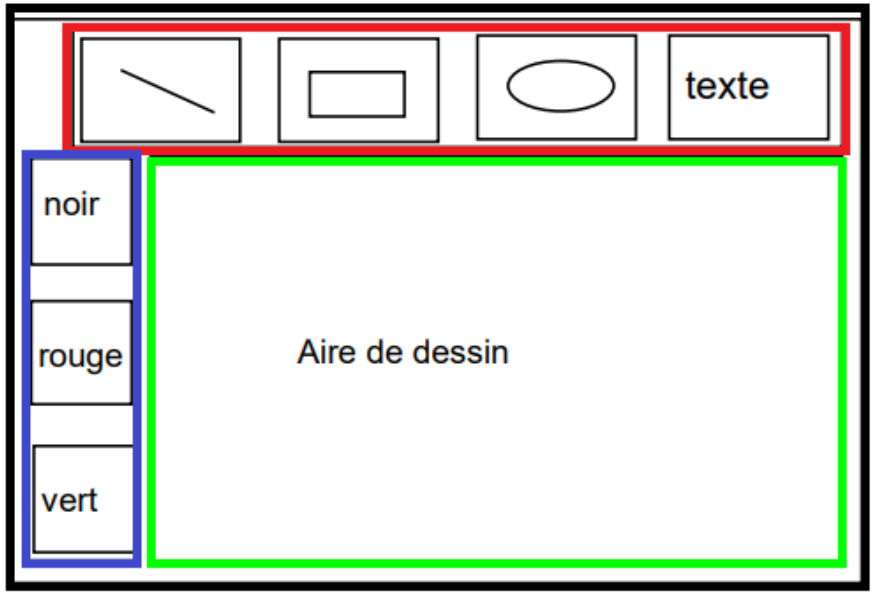


1.1. Premier principe des IHM (2/4)

- **Si on veut avoir ça:**

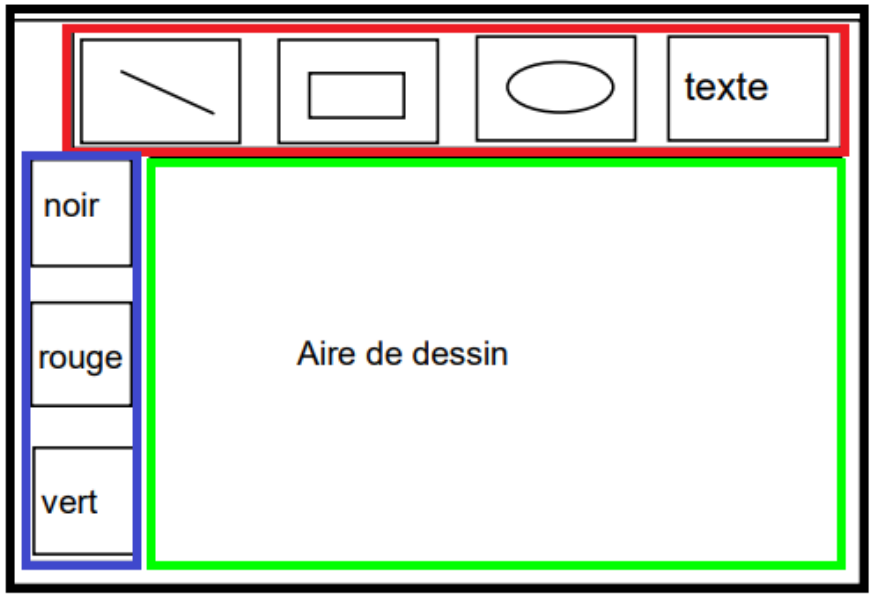


- **On doit construire cela:**

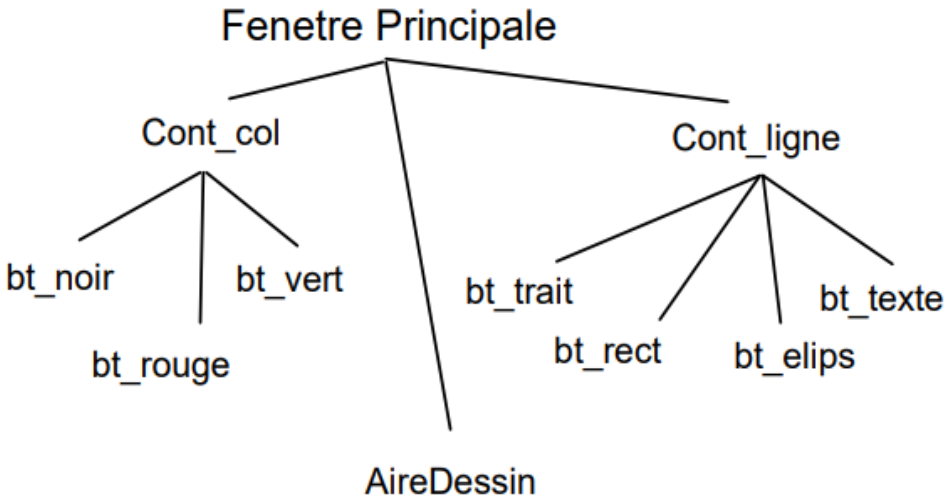


1.1. Premier principe des IHM (3/4)

- **Construire cette interface:**



- **C'est construire cet arbre :**



1.1. Premier principe des IHM (4/4)

- **Construire une IHM, c'est mettre des composants graphiques les uns à l'intérieur des autres**
- **Il y a donc, un arbre de composants graphiques**
- **Être "fils de" dans cet arbre signifie être "contenu dans"**

1.2. Second Principe des IHM

- **Les ensembles de composants graphiques sont des classes. Il y'a la classe des boutons, la classe des cases à cocher, la classe des texte édit, etc.**
- **Un composant graphique particulier sera une instance particulière d'une classe. Par exemple le bouton "Quitter" et le bouton "Sauvegarder" d'une IHM sont deux instances de la classe des boutons**
- **Il y a donc aussi, un arbre de classes pour les composants graphiques**

1.3 les éléments d'un interface graphique

- **Chaque interface graphique est composé par:**
  - **des vues (view)**
    et
  - **des layouts (conteneurs)**
- **Les relations entre les vues et leurs conteneurs (layouts) doivent être exprimées sous la forme d'un document XML.**

1.4 Propriétés générales des éléments graphiques dans le fichier XML

1.4.1. Déclarer des identifiants:

- **Chaque élément a un nom unique (identifiant= id) .**
- **Grace à cet e à cet identifiant, nous pouvez mettre en place les interactions et les traitements pour l'élément possédant cet identifiant.**
- **Pour associer un identifiant à un élément d'une vue, il faut utiliser l'attribut suivant : android:id="@+id/nom_identifiant"**

1.4.2. Spécifier la taille des éléments :

- **À chaque déclaration d'un élément graphique (conteneur ou composant), nous devons spécifier sa hauteur et sa largeur en utilisant ces propriétés:**
  - **android:layout_height: permet de définir la largeur de l'élément**
  - **android:layout_width: permet de définir la hauteur de l'élément**
- **Ces attributs peuvent avoir les valeurs suivantes:**

- **match_parent : signifie que la taille de l'élément est égale à celle de l'élément parent.**
- **wrap_content : signifie que la taille de l'élément est égale à celle de son contenu**

- **Par exemple, un bouton possédant une largeur définie à match_parent occupera le même espace que son conteneur.**

activity_main.xml × app × AndroidManifest.xml ×

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp" tools:context=".MainActivity">

    <Button
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="New Button"
        android:id="@+id/button"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="32dp"
        android:layout_alignParentEnd="false" />
</RelativeLayout>
```

1.5 Les Vues

- **Les layouts facilitent l'organisation des différents éléments qui composent une interface.**
- **Tous les éléments basiques d'une vue (bouton, zone de texte...) héritent de la classe View.**
- **Modifier une vue peut s'effectuer de deux manières :**
    1. **Mettre à jour le code XML de l'interface (onglet Text sous Android Studio).**
    2. **Mettre à jour la vue à l'aide de l'éditeur d'interface (onglet Design sous Android Studio).**

1.5.1 Exemple de vues (1/2)

- **Button: un bouton cliquable.**
- **CheckBox : une liste des éléments avec choix multiple.**
- **RadioButton : une liste des éléments avec choix unique.**
- **EditText : un champ de texte éditable.**
- **DatePicker: sélection de dates.**
- **Toast: un pop up message qui s'affiche sur l'écran.**
- **ImageButton : une image qui se comporte comme un bouton.**

1.5.1 Exemple de vues (2/2)

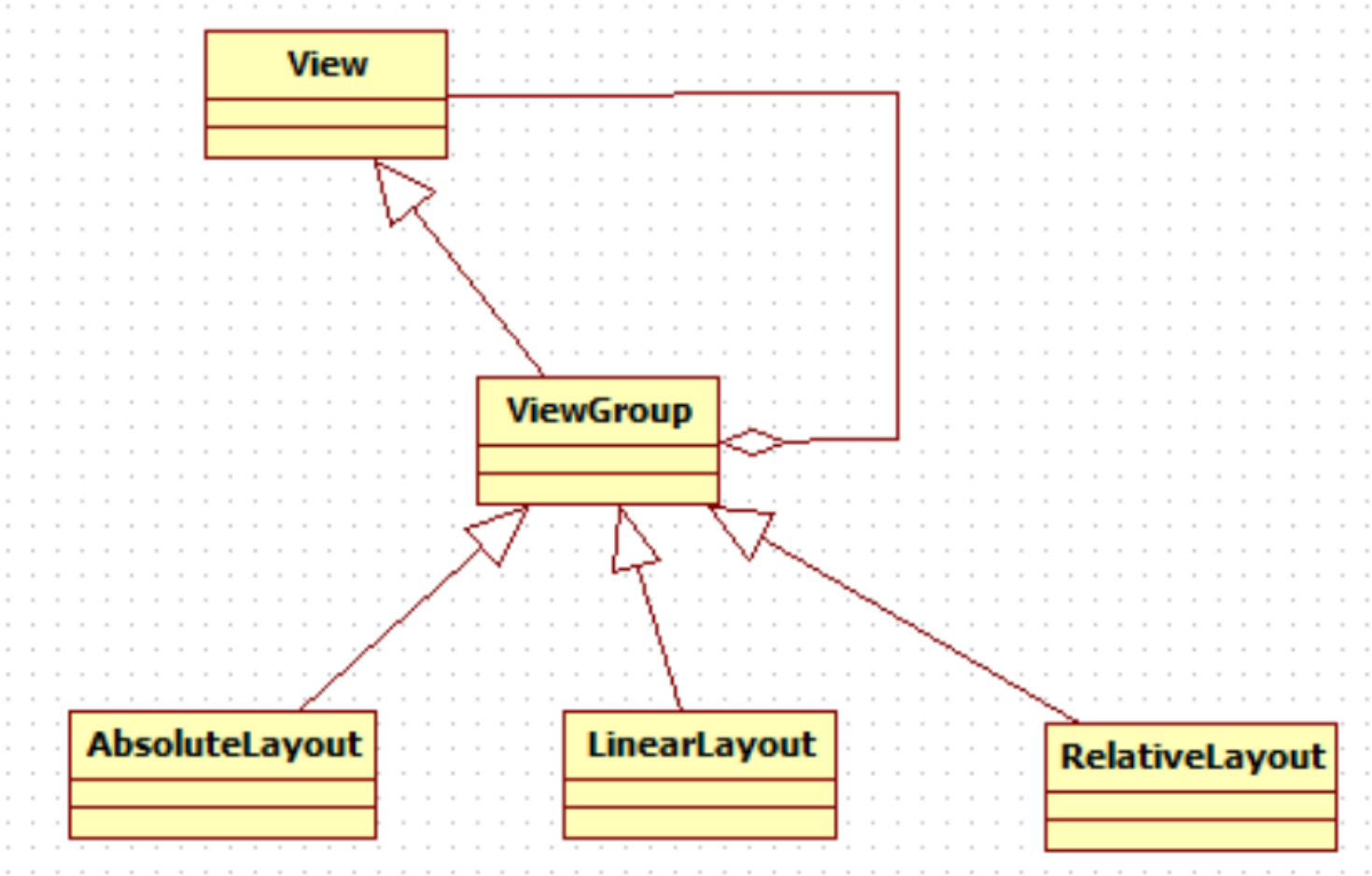
```
<TextView       android:id= "@+id/text"
                android:layout_width= "wrap_content"
                android:layout_height= "match_parent"
                android:text= "@string/letexte" />

<EditText       android:id= "@+id/edtext"
                android:layout_width= "wrap_content"
                android:layout_height= "match_parent"
                android:inputType= "textMultiLine"
                android:hint= "@string/letexte" />

<Button         android:id= "@+id/b1"
               android:layout_width= "wrap_content"
               android:layout_height= "wrap_content"
               android:text= "@string/boutonTexte" />

<CheckBox       android:id= "@+id/c1"
               android:layout_width= "wrap_content"
               android:layout_height= "wrap_content"
               android:text= "@string/texte"
               android:checked="true" />
```

1.6 Les Layouts (conteneurs) (1/2)

- **Les layouts facilitent l'organisation des différents éléments qui composent une interface.**
- **Ils servent de conteneur aux composantes d'une vue.**
- **On peut imbriquer des layouts les uns dans les autres, cela permet de créer des interfaces évoluées.**
- **Tous les éléments contenus dans le layout sont appelés des enfants. Par exemple, un layout enfant est inclut dans le layout parent (il faut noter que l'écran est le parent du layout parent)**

1.6 Les Layouts (conteneurs) (2/2)

- **Tous les layouts Android héritent de la classe ViewGroup.**
- **La classe ViewGroup hérite de la classe View (voir la figure suivante).**


View
ViewGroup
AbsoluteLayout
LinearLayout
RelativeLayout

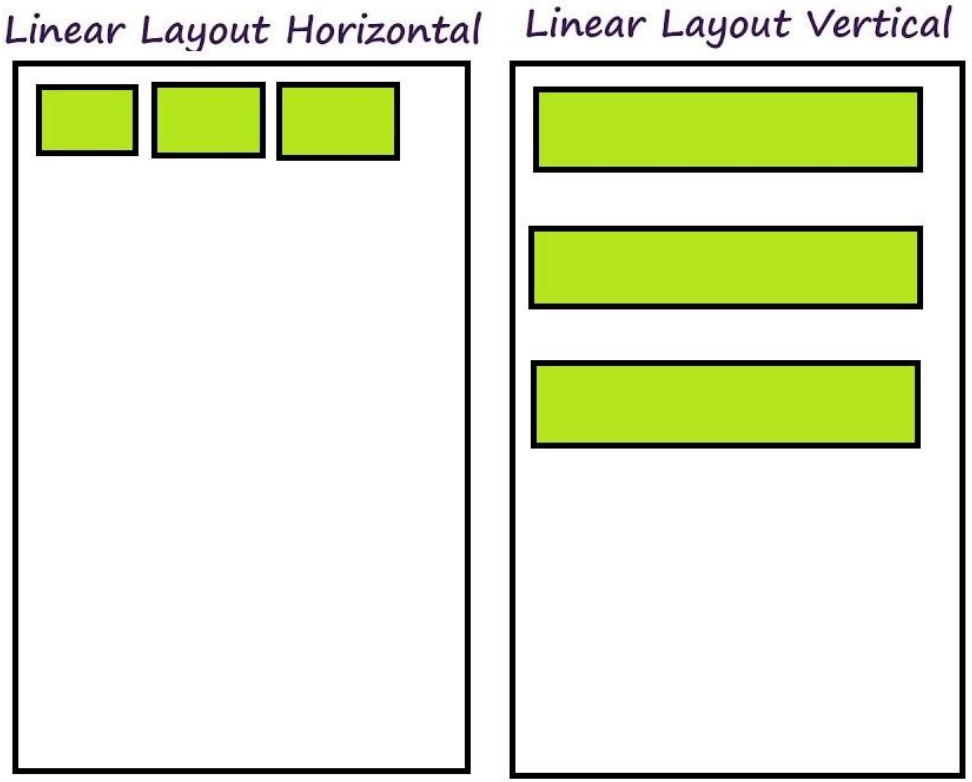
1.6.1 Les types de layouts:

Les éléments de l'interface (vues et layouts) sont disposés selon le type de layout choisi :

- **LinearLayout**
- **AbsoluteLayout**
- **RelativeLayout**

LinearLayout (1/2):

- **il organise les différents éléments de votre interface sur une ligne ou sur une colonne.**
- **L'attribut orientation permet de préciser dans quel sens s'affichent les éléments :**
  - **Avec la valeur horizontal, l'affichage est de gauche à droite**
  - **Avec la valeur vertical affichage est de haut en bas.**



LinearLayout (2/2):

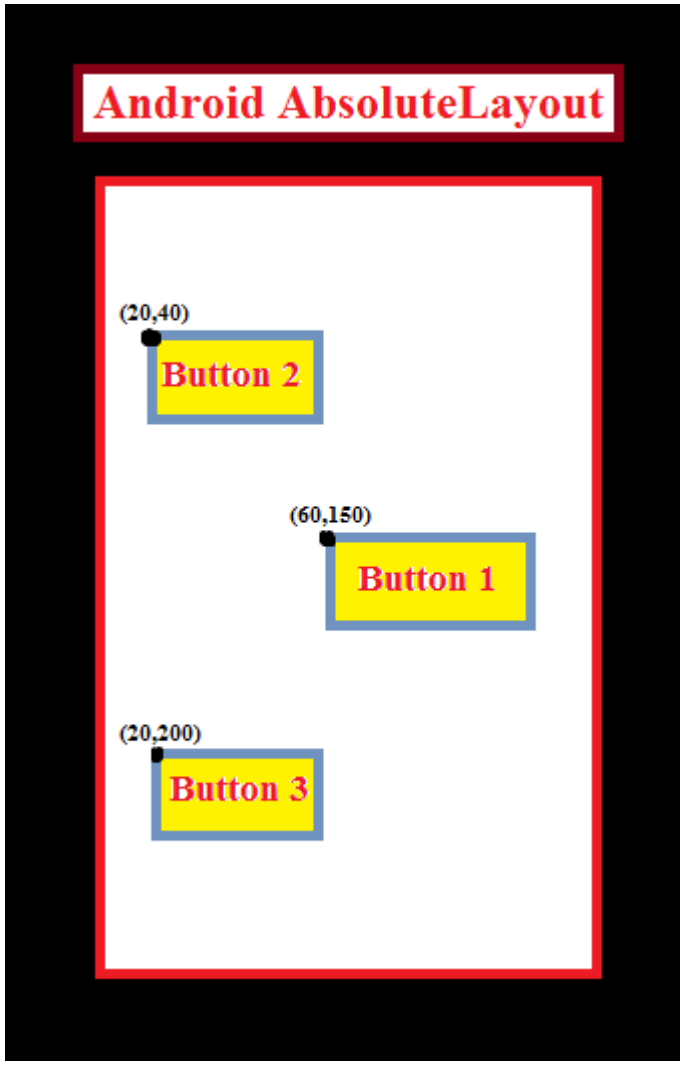
- **L'extrait de code suivant montre comment inclure un LinearLayout dans un fichier layoutXML. Il faut noter que le fichier XML qui contient l'interface est stocké dans le répertoireres/layout**

```
<LinearLayout       xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation="horizontal">
    <!— Ajoutez ici d'autres vues ou layouts. Elles sont considérées
        comme des "enfants" de LinearLayout -->
 </LinearLayout>
```

AbsoluteLayout :

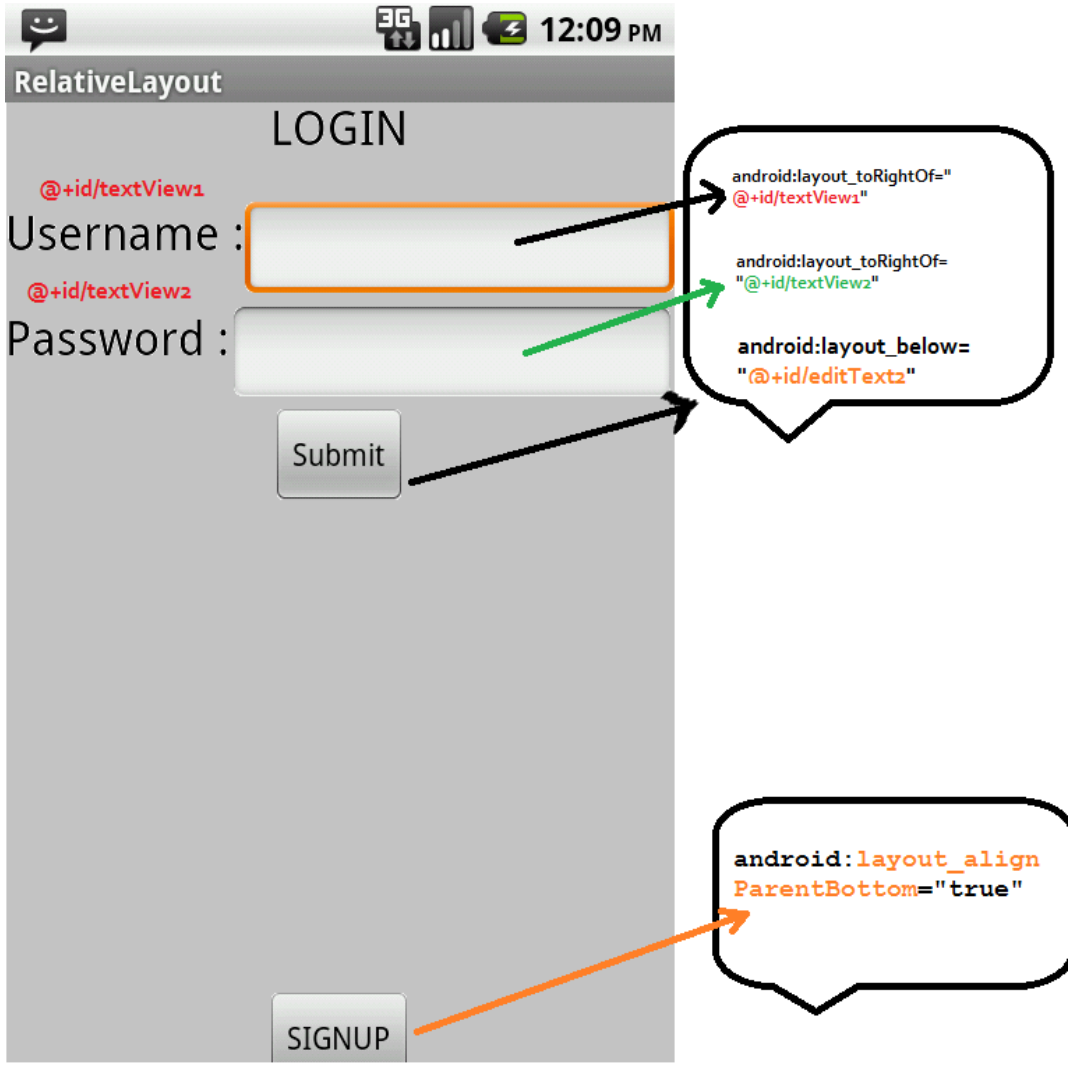
- **Cette mise en page nous laisse définir les coordonnées exactes des éléments qui la composent**

- **il nous permet de positionner les vues à l'écran en utilisant les coordonnées X, Y. C'est-à-dire que nous pouvons spécifier l'emplacement exact des éléments en utilisant les coordonnées x et y.**



RelativeLayout :

- il permet de définir la position des éléments en fonction de la position de leurs éléments parents.
- Le premier élément sert de référence pour les autres.



1.7. Associer une interface à une activité (logique métier)

- **Chaque interface est gérée par une instance d'une sous-classe d'Activity.**
- **Le contenu de l'interface est chargé lorsque l'activité est instanciée.**
- **La méthode onCreate est redéfinie(surchargée) pour spécifier la définition de l'interface à afficher via la méthode: setContentView**

1.8. Associer une interface à une activité (logique métier)

```
import android.app.Activity;

import android.os.Bundle;


public class Main extends Activity {

      @Override

      public void onCreate(Bundle savedInstanceState) {

       super.onCreate(savedInstanceState);

       setContentView(R.layout.main);

        }

}
```

2. Gérer les évènements sur les widgets

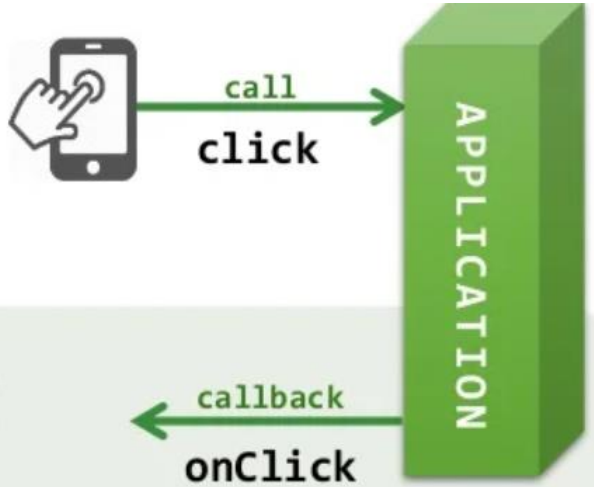
- **Contrairement aux icônes que l'on peut voir dans le menu principal, un widget n'est pas un raccourci permettant de lancer une application, mais une application qui fonctionne directement sur l'accueil du téléphone.**
- **Un widget permet par exemple d'afficher en temps réel les dernières actualités, les derniers tweets postés par vos abonnements ou encore les derniers bulletins météo. Il peut également se présenter sous la forme d'un outil tel qu'une calculatrice ou un agenda**

2.1. Gestion des évènements en utilisant les listeners

- **Toute action faite par l'utilisateur pour interagir avec l'interface graphique (le click, le maintien du click, les touches, etc.) est perçues comme un événement. Cet événement est intercepté par un mécanisme basé sur la notion d'écouteur (listener).**

- **Un listener permet d'attribuer un événement à une méthode pour l'appeler dés que cet événement sera produit.**
- **Par exemple, la classe android.view.View.OnClickListener permet d'associer un écouteur d'événements de type click, et la méthode à surcharger pour traiter ces événements est onClick(View)**

2.3. Gérer les évènements sur les widgets



/java/MainActivity.java

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ...
    Button b = (Button) findViewById(R.id.btn);
    b.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v){
            ... // bouton cliqué!!!
        }
    });
}
```

2.3.1 Liste des interfaces de Listeners graphiques

android.View

```
OnClickListener                    // clic
OnLongClickListener                // clic long
OnDragListener                     // glissement
OnTouchListener                    // touché
OnHoverListener                    // survol
OnKeyListener                      // frappe de clavier
OnAttachStateChangeListener        // changement de l'état d'attachement
OnLayoutChangeListener             // changement du layout
OnCreateContextMenuListener        // création du menu contextuel
OnFocusChangeListener              // changement du focus
OnGenericMotionListener            // un mouvement (mouse, pen, finger, ...)
OnSystemUiVisibilityChangeListener // changement de la visibilité de
                                   // la barre d'état
```

2.3.2 Gestion des évènements au niveau du layout (fichier XML)

/res/layout/activity_main.xml

```
<Button
    android:onClick="func"
    android:id="@+id/btn"/>
...
```

/java/MainActivity.java

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ...
}
public void func(View v) {
    ... // bouton cliqué!!!
}
```

Merci de votre attention!

N’hésitez pas à poser vos questions…