APPLICATIONS MOBILES

L’ interface utilisateur

Ecran, Interface, Vue, I.H.M., U.I, G.U.I.

C’est la partie visible de l’application.

Elle permet l’interaction avec l’utilisateur:

- Lui permettre de saisir des informations.
- Visualiser des informations.
- Lancer des traitements.

• Respecter : l’ergonomie, la charge cognitive,

.. toutes les recommandations des IHM.

L’ interface utilisateur

- Une partie dynamique contenant le code *Java*.
- Une partie statique contenant la description de l’interface utilisateur en *XML*.

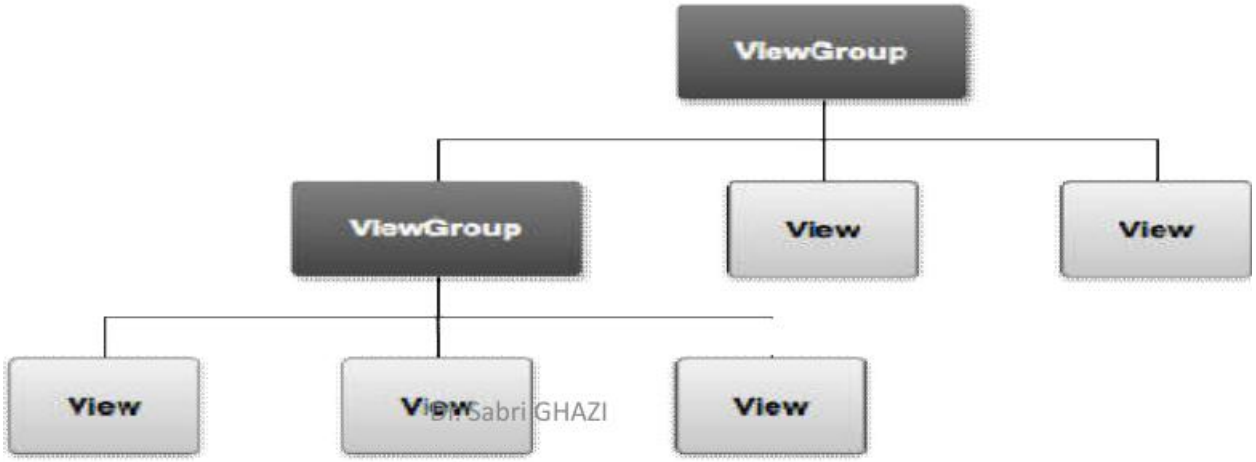
– Cette séparation permet de

- Réaliser des maquettes(prototypes) de l’application .
- Concevoir les interfaces indépendamment de la programmation.
- Collaborer entre plusieurs développeurs sur une même fonctionnalité.
- Faciliter de modifier l’interface sans changer le code source de l’application (externalisation).
- Faciliter l’internationalisation de l’application.

L’ interface utilisateur

- Le package des classes qui permettent la création des interfaces utilisateurs sous Android sont organisés comme suit :
- Chaque composant de l’interface utilisateur est : Un View
-

– Un ViewGroup est un composant qui peut contenir d’autre View ou d’autre ViewGroup



INTERFACE GRAPHIQUE
VUE

- Élément graphique appelé aussi vue (widget en anglais)
- Sont des objets de l’ interface utilisateur
- Exemple :
  - Boutons
  - Champs de texte… etc.
- Une vue hérite de la classe View
- Possède des attribues pour configurer son apparence et comportement

Utilisation du concepteur d’ Interface

- Le concepteur des Interfaces permet de décrire d’une façon graphique les écrans de notre application.
- La description XML est générée automatiquement.

INTERFACE GRAPHIQUE
VUE

- chaque activité possède sa propre IHM.
- Les composants de base (les widgets).
  - ➔ - Les Zones de texte statique **TextView**
  - ➔ - Les Zones de saisi de texte **EditText**
  - ➔ - Les Boutons radio **RadioButton**

INTERFACE GRAPHIQUE
GROUPE DE VUES

- Un groupe de vues (ViewGroup en anglais)
- Responsable de l'organisation d'autres vues
- Également connu en tant que gestionnaire de mise en page
- Sous classe de ViewGroup qui étend la classe android.view.View
- Ils peuvent être imbriqués pour créer des modèles complexes

INTERFACE GRAPHIQUE
LAYOUTS(GABARITS)

- Des ViewGroup particuliers sont prédéfinis
  - Gabarits (layouts)
- Proposent une prédisposition des objets graphiques
- Les plus connus ou utilisés:
  - LinearLayout
  - RelativeLayout
  - GridLayout
  - FrameLayout
  - …
- Les déclarations se font principalement en XML
  - ce qui évite de passer par les instanciations Java.

INTERFACE GRAPHIQUE
ATTRIBUTS

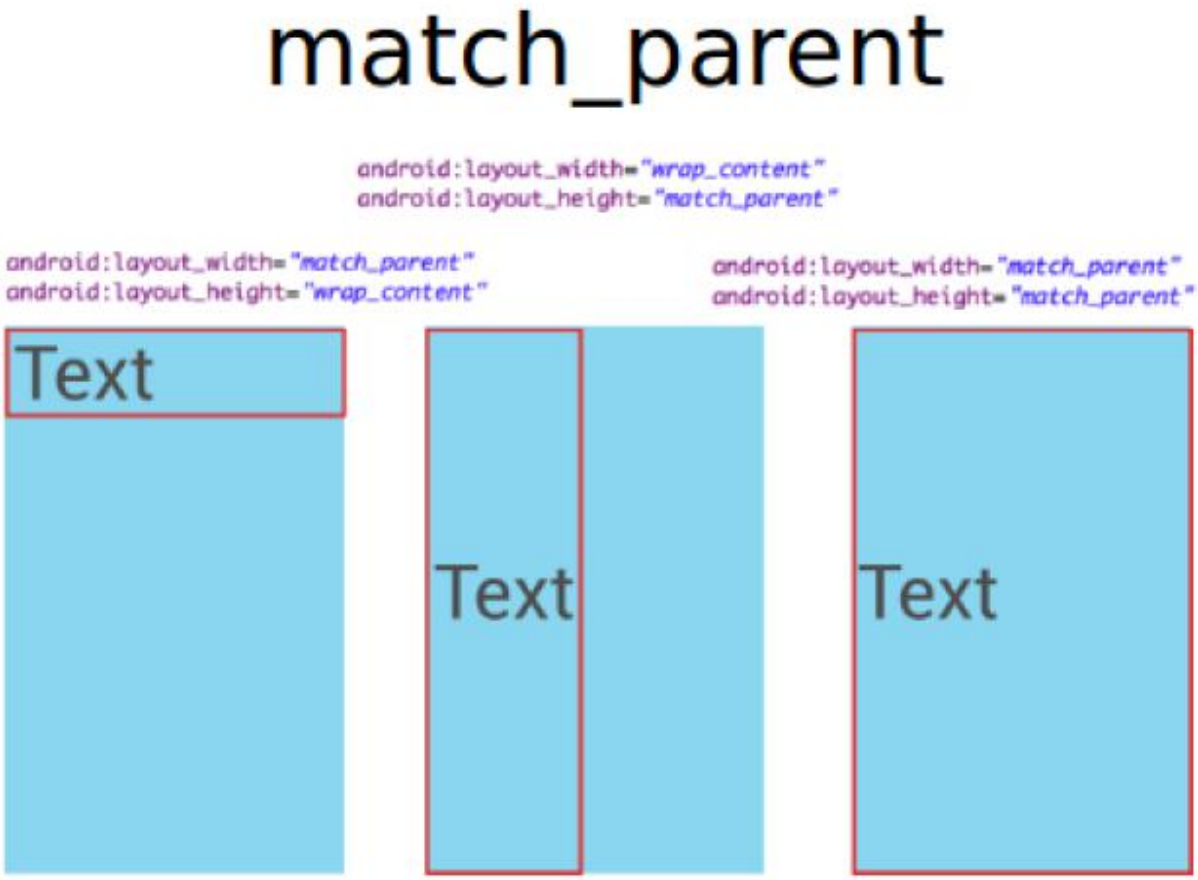
- Comme tout objet graphique, les gabarits ont des attributs qui permettent de les configurer
- Les attributs les plus importants sont :
  - android:layout_width
  - android:layout_height
- Parmi les valeurs possibles :
  - ="match_parent": l'élément remplit tout l'élément parent
  - ="wrap_content": prend la place minimum nécessaire à l'affichage
- d'autres attributs :
  - android:orientation: définit l'orientation d'empilement
  - android:gravity: définit l'alignement des éléments

INTERFACE GRAPHIQUE
ATTRIBUTS

wrap_content

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    ...
    />
```

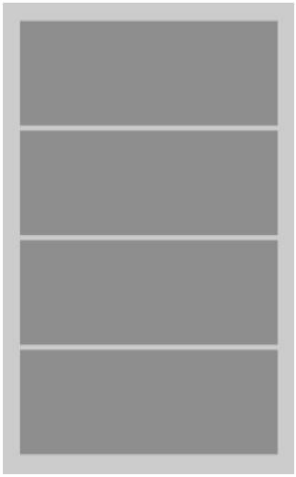
INTERFACE GRAPHIQUE
ATTRIBUTS DES GABARITS 3



INTERFACE GRAPHIQUE
LINEAR_LAYOUT

- dispose les éléments de gauche à droite ou du haut vers le bas
- Dans une seule colonne ou ligne
- En fonction de l'attribut :
  - **android:orientation** = (horizontal ou vertical)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:gravity="center"
  android:id="@+id/accueilid"
  >
</LinearLayout>
```



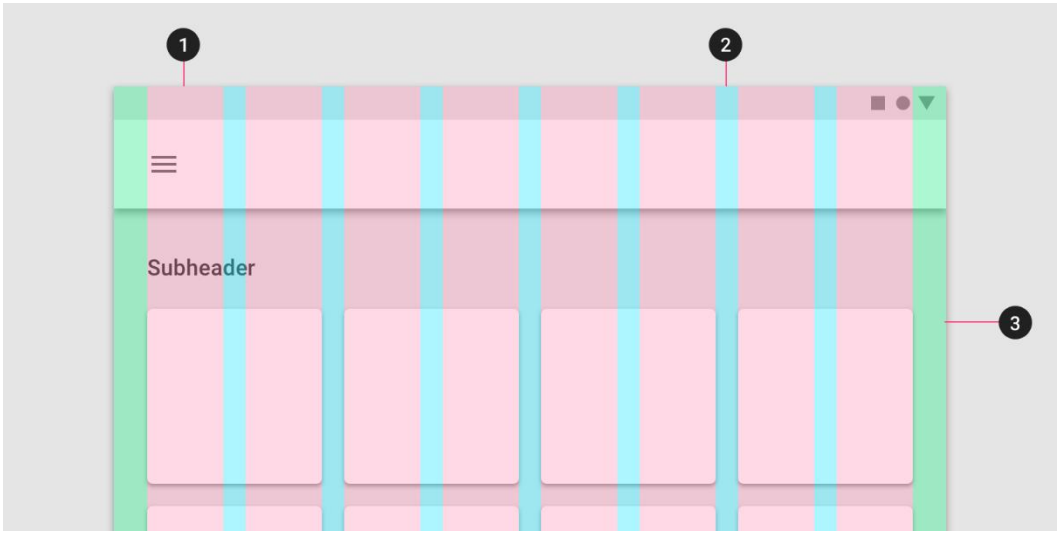
INTERFACE GRAPHIQUE
RELATIVE_LAYOUT

- les éléments enfants sont placés les uns par rapport aux autres

```
<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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <EditText
        android:id="@+id/main_input"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="14dp"
        android:ems="10" >

        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editText1"
        android:layout_below="@+id/editText1"
        android:layout_marginTop="31dp"
        android:onClick="onClick"
        android:text="Start" />

</RelativeLayout>
```

INTERFACE GRAPHIQUE
GRID_LAYOUT

Cette disposition permet d'organiser une vue sur une grille.



1. Columns
2. Gutters
3. Margins

- GridLayout sépare sa zone de dessin en lignes, colonnes et cellules.

INTERFACE GRAPHIQUE
FRAME_LAYOUT

- FrameLayout est un gestionnaire de mise en page qui positionne tous les éléments enfants au-dessus des autres en les empilant

INTERFACE GRAPHIQUE COMME RESSOURCE

- L'interface graphique définit en xml
  - sera aussi générée comme une classe dans la classe statique R
- Exemple :

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.acceuil);
}
```

  - accueil.xml (par exemple)
- Layout reste modifiable au travers le code java
  - Il est important de lui spécifier un ID dans xml : android:id="@+id/accueilid »
  -

```
LinearLayout l = (LinearLayout)findViewById(R.id.accueilid);
l.setBackgroundColor(Color.BLACK);
```

  - (Le "+" signifie que cet id est nouveau et doit être généré dans la classe R. Un id sans "+" signifie que l'on fait référence à un objet déjà existant.)

INTERFACE GRAPHIQUE
LABEL DE TEXTE

En xml :

```
<TextView
  android:id="@+id/le_texte"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/hello"
  android:layout_gravity="center"
  />
```

Par programmation :

```
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); // empiler vers le bas !

    TextView texte = new TextView(this);
    texte.setText("Programming creation of interface !");
    gabarit.addView(texte);
    setContentView(gabarit);
} }
```

INTERFACE GRAPHIQUE
EDITEUR DE TEXTE

En xml :

```
<EditText android:text=""
          android:id="@+id/EditText01"
          android:layout_width="match_parent"
          android:layout_height="wrap_content">
</EditText>
```

Par programmation :

```
EditText edit = new EditText(this);
edit.setText("Edit me");
gabarit.addView(edit);
```

Réception d'évènements :

```
edit.addTextChangedListener(new TextWatcher() {
  @Override
  public void onTextChanged(CharSequence s, int start,
                            int before, int count) {
    // do something here
  }
);
```

INTERFACE GRAPHIQUE
IMAGES

En xml :

```
<ImageView
  android:id="@+id/logoEnsi"
  android:src="@drawable/ensi"
  android:layout_width="100px"
  android:layout_height="wrap_content"
  android:layout_gravity="center_horizontal"
  ></ImageView>
```

Par programmation :

```
ImageView image = new ImageView(this);
image.setImageResource(R.drawable.ensi);
gabarit.addView(image);
```

INTERFACE GRAPHIQUE
BOUTONS

En xml :

```
<Button android:text="Go !"
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
</Button>
```

Par programmation :

```
Button b = (Button)findViewById(R.id.Button01);
b.setOnClickListener(new OnClickListener() {

  @Override
  public void onClick(View v) {
    Toast.makeText(v.getContext(), "Stop !", Toast.LENGTH_LONG).show();
  }
  });
}
```

INTERFACE GRAPHIQUE
INCLUSION DE LAYOUTS (GABARITS)

- Les interfaces peuvent aussi inclure d'autres interfaces, permettant de factoriser des morceaux d'interface

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >
<include android:id="@+id/include01"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         layout="@layout/acceuil"
         ></include>
</LinearLayout>
```

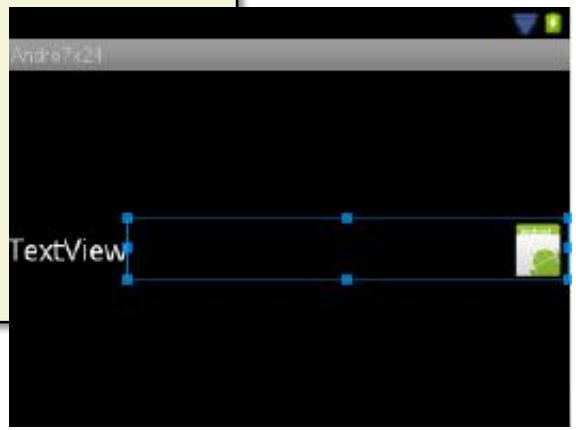
avec utilisation du mot clé merge dans accueil.xml

```
<merge xmlns:android="http://schemas.android.com/apk/res/android">
  <TextView ... />
  <TextView ... />
</merge>
```

INTERFACE GRAPHIQUE
POSITIONNEMENT AVANCÉ

- Pour obtenir une interface agréable, il est souvent nécessaire de réaliser correctement le positionnement des éléments graphiques
- Les interfaces peuvent aussi inclure d'autres interfaces, permettant de factoriser des morceaux d'interface

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_height="match_parent" android:orientation="horizontal"
     android:layout_width="match_parent"
     android:gravity="center">
     <TextView ...></TextView>
     <LinearLayout android:layout_height="wrap_content"
             android:orientation="horizontal"
             android:layout_width="match_parent"
             android:gravity="right"
             android:layout_gravity="center">
     <Image .../>
</LinearLayout></LinearLayout>
```



INTERFACE GRAPHIQUE
LISTES

- La vue liste permet d'afficher un ensemble d'éléments graphique sous forme de liste déroulante si le nombre d'éléments est assez important
- Exemple de liste plain écran
- On définit un layout *montexte*
- *ListAdapter* définit un élément de la liste
- *arrayAdapter* contient les éléments de la liste

```
<TextView ...> </TextView>
```

```
<LinearLayout ...>
<ListView android:id="@+id/listView1" ...>
</ListView></LinearLayout>
```

```
ListView list = (ListView)findViewById(R.id.listView1);
ArrayAdapter<String> tableau = new ArrayAdapter<String>(list.getContext(),
                                                        R.layout.montexte);
for (int i=0; i<40; i++) {
  tableau.add("coucou " + i); }
list.setAdapter(tableau);
```

INTERFACE GRAPHIQUE
FRAGMENTS

- À partir de android 3.0
- Composant graphique similaire à une activity.
- Un fragment est un composant qui fonctionne dans le contexte d'une activité.
- Une sorte de sous activité qui possède son propre cycle de vie.
- On peut intégrer plusieurs fragments dans une activités, et changer un fragment par un autre dynamiquement, sans changer l'activité principale.
- → programmation graphique dynamique plus aisée.

BIBLIOGRAPHIE

- Applications Mobile; cours de R.meghatria et M.Khaled ; UDBKM 2018
- **Développement Android - Jean-Francois Lalande** - March 2019
- developer.android.com 2020.