3. Interfaces Graphiques

- 3.1 Vues et gabarits
- 3.2 Inclusions de gabarits
- 3.3 Positionnement avancé
- 3.4 Les listes
- 3.5 Les galeries
- 3.6 Les onglets

3.1 Vues et Gabarits

Les éléments graphiques héritent de la classe **View**. On peut regrouper des éléments graphiques dans une **ViewGroup**. Des **ViewGroup** particuliers sont prédéfinis: ce sont des gabarits (*layout*) qui proposent une prédispositions des objets graphiques:

- **LinearLayout**: dispose les éléments de gauche à droite ou du haut vers le bas
- **RelativeLayout**: les éléments enfants sont placés les uns par rapport aux autres
- **TableLayout**: disposition matricielle
- **FrameLayout**: disposition en haut à gauche en empilant les éléments
- **GridLayout**: disposition matricielle avec N colonnes et un nombre infini de lignes
- **ConstraintLayout**: similaire au RelativeLayout mais créé sur mesure pour l'éditeur visuel

Les déclarations se font principalement en XML, ce qui évite de passer par les instanciations Java.

Attributs des Layouts

Les attributs des gabarits permettent d'adapter la disposition des éléments fils. Les plus importants attributs sont:

- **android:layout_width** et **android:layout_height**:
  - **="match_parent"**: l'élément remplit tout l'élément parent
  - **="wrap_content"**: prend la place minimum nécessaire à l'affichage
  - **="fill_parent"**: comme **match_parent** (deprecated, API<8)
- **android:orientation**: définit l'orientation d'empilement
- **android:gravity**: définit l'alignement des éléments

```
<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>
```

Exemples de gabarits

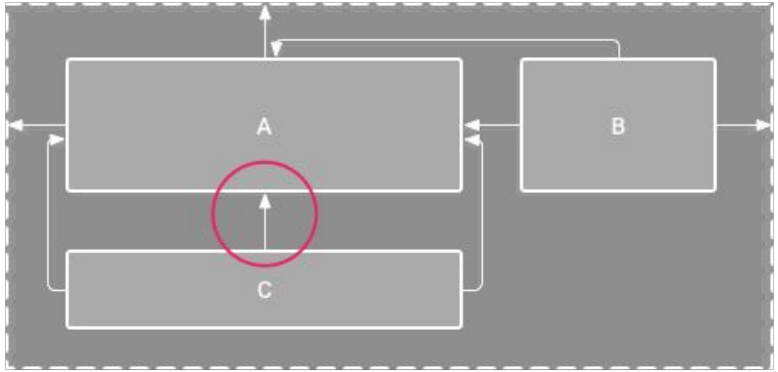
RelativeLayout : Dans ce gabarit on positionne chaque élément par rapport à son conteneur :

- android:layout_alignParentTop (true / false) : Cette option permet de préciser si le haut de l'élément doit être aligné avec celui de son conteneur.
- Même principe pour : android:layout_alignParentBottom, android:layout_alignParentLeft et android:layout_alignParentRight.
- android:layout_centerHorizontal : Indique si l'élément doit être centré horizontalement dans son conteneur.
- Même principe pour : android:layout_centerVertical.
- android:layout_centerInParent : Vous permet d'indiquer que l'élément doit être centré horizontalement et verticalement dans le conteneur.

Afin de pouvoir référencer le positionnement d'un élément par rapport à un autre, on dispose d'un moyen simple et efficace, il s'agit des identificateurs (ID).

Rappel de l'utilisation des ID :

- A la déclaration d'un élément : android:id= "@+id/idElement"
- A l'utilisation : @id/idElement

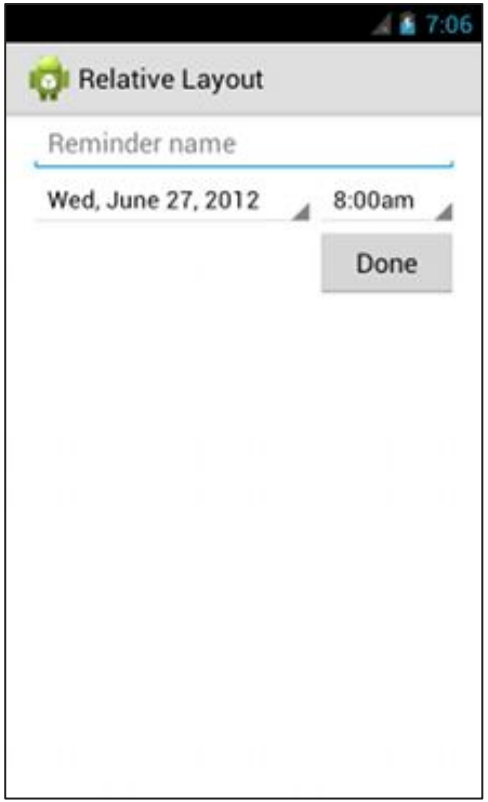


RelativeLayout (positionnement relatif)

- android:layout_above : Indique que l'élément sera placé au-dessus de celui indiqué par son id.
- android:layout_below : Indique que l'élément sera placé en dessous de celui indiqué par son id.
- android:layout_toLeftOf : Indique que l'élément sera positionné à gauche de celui indiqué par son id.
- android:layout_toRightOf : Indique que l'élément sera positionné à droite de celui indiqué par son id.
- android:layout_alignTop : Indique que le haut de notre élément est aligné avec le haut de l'élément indiqué.
- android:layout_alignBottom : Indique que le bas de notre élément est aligné avec le bas de l'élément indiqué.
- android:layout_alignLeft : Indique que le côté gauche de notre élément est aligné avec le côté gauche de l'élément indiqué.
- android:layout_alignRight : Indique que le côté droit de notre élément est aligné avec le côté droit de l'élément indiqué.
- android:layout_alignBaseLine : Indique que les lignes de base des 2 éléments sont alignées.

ATTENTION! Left/Right ⇒ Start/End

RelativeLayout
(exemple)




```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android
"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```

Exemples de gabarits

Positionnement linéaire LinearLayout:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:gravity="center"
android:orientation="vertical"
android:id="@+id/accueilid"
  >
    <TextView android:id="@+id/le_texte"
android:text="@string/hello" />
    <TextView android:id="@+id/le_texte2"
android:text="@string/hello2" />
</LinearLayout>
```

Positionnement en cartes FrameLayout:

```
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:id="@+id/accueilid"
  >
    <ImageView
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:src="@drawable/mon_image" />

    <TextView android:id="@+id/le_texte"
android:text="@string/hello" />
</FrameLayout>
```

L'interface graphique comme ressource

Une interface graphique définie en XML sera aussi générée comme une ressource dans la classe statique **R**. Le nom du fichier xml, par exemple *accueil.xml* permet de retrouver le layout dans le code java au travers de **R.layout.accueil**.

Ainsi, pour associer la première vue graphique à l'activité principale de l'application, il faut faire:

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

L'interface graphique comme ressource

Le layout reste modifiable au travers du code, comme tous les autres objets graphiques. Pour cela, il est important de spécifier un id dans la définition XML du gabarit (**android:id="@+id/accueilid"**). 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 une ressource déjà existante.

En ayant généré un *id*, on peut accéder à cet élément et agir dessus au travers du code Java:

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

Les Labels 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"
  />
```

En Java: (ATTENTION! ç'est juste pour montrer que c'est possible, mais c'est à éviter!)

```
public class Activity2 extends Activity {
    public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    LinearLayout gabarit = new LinearLayout(this);
    gabarit.setGravity(Gravity.CENTER);
    gabarit.setOrientation(LinearLayout.VERTICAL);

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

}
```

Les zones de saisie

En XML:

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

En Java: (ATTENTION! à éviter!)

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

Interception d'événements:

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

Les images

En XML:

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

En Java:

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

Les boutons

En XML:

```
<Button android:text="@string/go"
        android:id="@+id/goButton"

android:layout_width="wrap_content"

android:layout_height="wrap_content">
</Button>
```

La gestion des événements de *click* se font par l'intermédiaire d'un listener:

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

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

Il est aussi possible d'utiliser l'attribut onClick="fonction" depuis le code XML pour capturer les évènements de click!

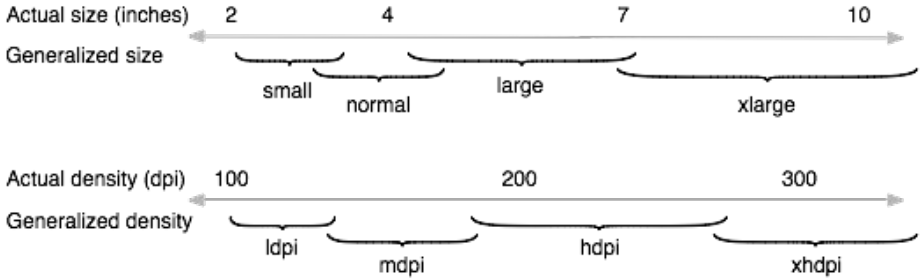
Unités de mesure

- **dp** : Density independent Pixel (Pixel indépendant de la Densité) - Unité abstraite qui est basée sur la densité physique de l'écran. Cette unité est égale à un pixel sur un écran de 160 DPI (Points par pouce). Cette dimension est utilisée pour la mise en page des éléments.
- **sp :** Scale independent Pixel (Pixel indépendant de l'échelle) - Utilisé pour les tailles de polices. On pourrait comparer cette unité aux em du développement web. La police peut être plus ou moins grosse suivant les préférences utilisateurs
- **pt** : Point - 72 points par pouce. basé sur la taille physique de l'écran.
- **px** : Pixels - Corresponds aux pixels réels de l'écran. Cette unité de mesure n'est pas recommandées car le rendu sur les différents types d'écrans peut être différents. Le nombre de pixels par pouce peut varier suivant les appareils.
- **mm** : Millimètre - basée sur la taille physique de l'écran
- **in** : Inches (Pouces) - basée sur la taille physique de l'écran

Interfaces graphiques et propriétés de l'écran

Les densités d'écrans :

- *ldpi* (low) ~120dpi
- *mdpi* (medium) ~160dpi
- *hdpi* (high) ~240dpi
- *xhdpi* (extra-high) ~320dpi
- *xxhdpi* (extra-extra-high) ~480dpi
- *xxxhdpi* (extra-extra-extra-high) ~640dpi



Les tailles d'écrans :

- *xlarge* à partir de 960dp x 720dp
- *large* à partir de 640dp x 480dp
- *normal* à partir de 470dp x 320dp
- *small* à partir de 426dp x 320dp

Les orientations des écrans :

- landscape / portrait /square

3.2 Inclusions de gabarits

Les interfaces peuvent aussi inclure d'autres interfaces, permettant de factoriser des morceaux d'interface. On utilise dans ce cas le mot clé **include**:

```
<?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>
```

Si le gabarit correspondant à accueil.xml contient lui aussi un **LinearLayout**, on risque d'avoir deux imbrications de **LinearLayout** inutiles car redondant:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android">
  <TextView ... />
  <TextView ... />
</LinearLayout>
```

Le résultat de l'inclusion :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android">
<LinearLayout ...>
  <TextView ... />
  <TextView ... />
</LinearLayout>
</LinearLayout>
```

Fusion de gabarits

Le problème précédent est dû au fait qu'un layout doit contenir un unique *root element*, du type **View** ou **ViewGroup**. On ne peut pas mettre une série de **TextView** sans *root element*. Pour résoudre ce problème, on peut utiliser le tag **merge**. Si l'on réécrit le layout accueil.xml comme cela:

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

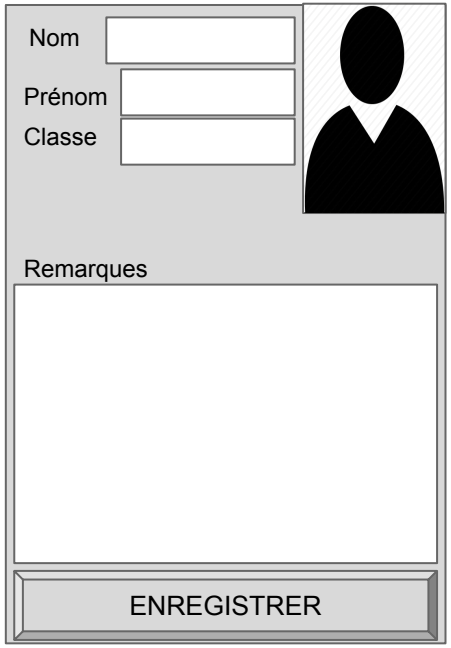
L'inclusion de celui-ci dans un layout linéaire transformera:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android">
  <include android:id="@+id/include01"
layout="@layout/acceuil"></include>
</LinearLayout>
```

en:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android">
  <TextView ... />
  <TextView ... />
</LinearLayout>
```

Exercice : première interface graphique



- En utilisant des LinearLayout, construisez une activité qui correspond à l'interface ci contre :
- En utilisant un RelativeLayout, refaites la même activité.
- Affichez un message temporaire sur l'écran quand le bouton “enregistrer” est cliqué.
- Changez l'arrière plan de l'activité en un dégradé de couleurs.
- Créez une interface adaptée au mode paysage. Elle exploite l'espace de l'écran en affichant la zone des remarques à côté de l'image au lieu d'être en dessous.