IHM ET COMPOSANTS GRAPHIQUES SOUS LA PLATEFORME ANDROID

19/02/2017

Imene Sghaier

PLAN DU CHAPITRE

- Construction d'une IHM
- Les gabarits
- Les Composants graphiques non conteneurs

Construction d'une IHM

- Construire une IHM, c'est mettre des composants graphiques les uns à l'intérieur des autres
- Les ensembles de composants graphiques sont des classes. On aura la classe des boutons, la classe des cases à cocher, etc.
- Un composant graphique particulier sera une instance particulière d'une classe. Par exemple le bouton "Quitter" et le bouton "Sauvegarder" d'une IHM seront deux instances de la classe des boutons

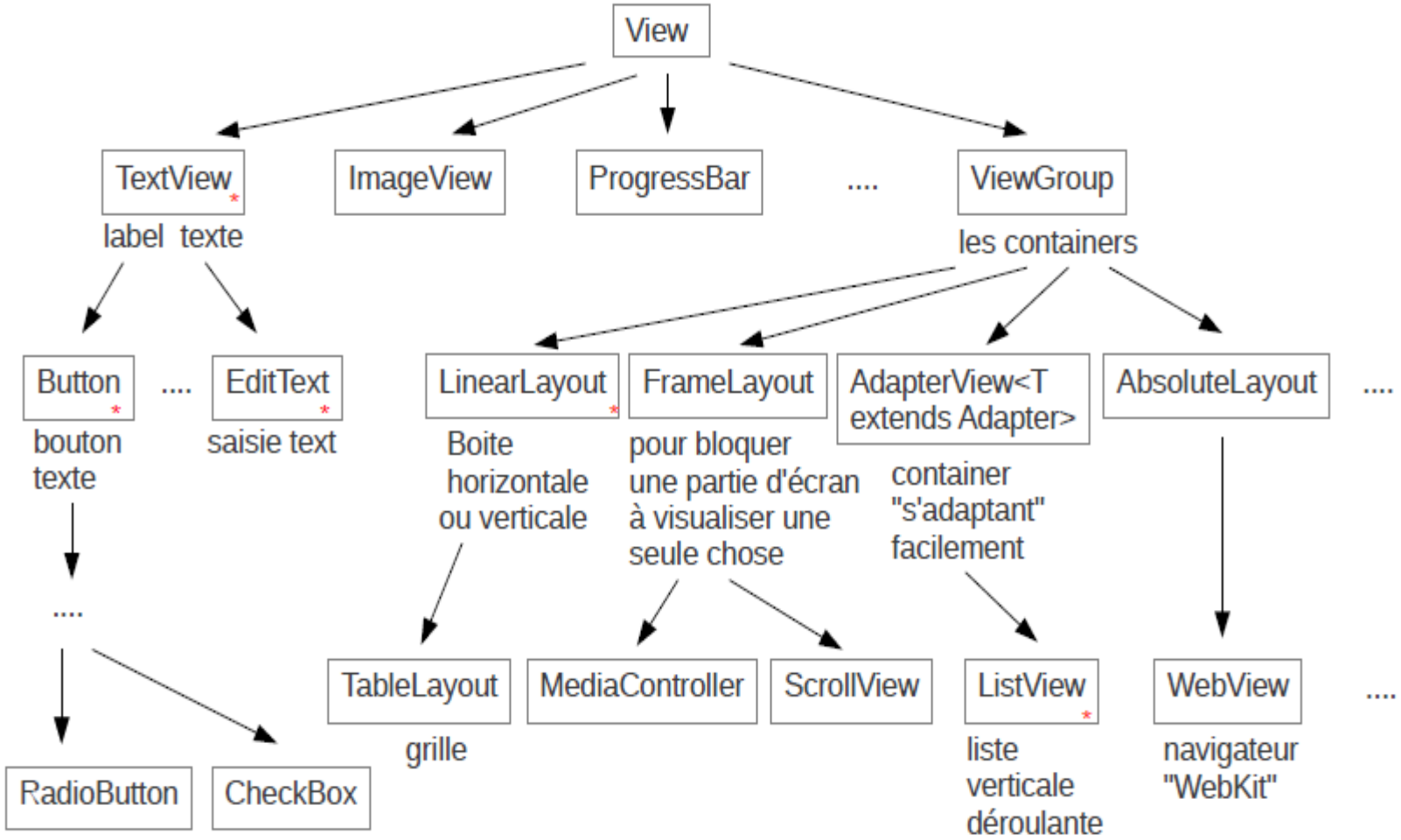
Construction d'une IHM

- Un composant graphique a 3 parties :
  - les données qu'il représente : le modèle (model)
  - le dessin d'affichage : la vue (view)
  - ce qui prend en charge les actions de l'utilisateur sur ce composant : le controleur

Construction d'une IHM

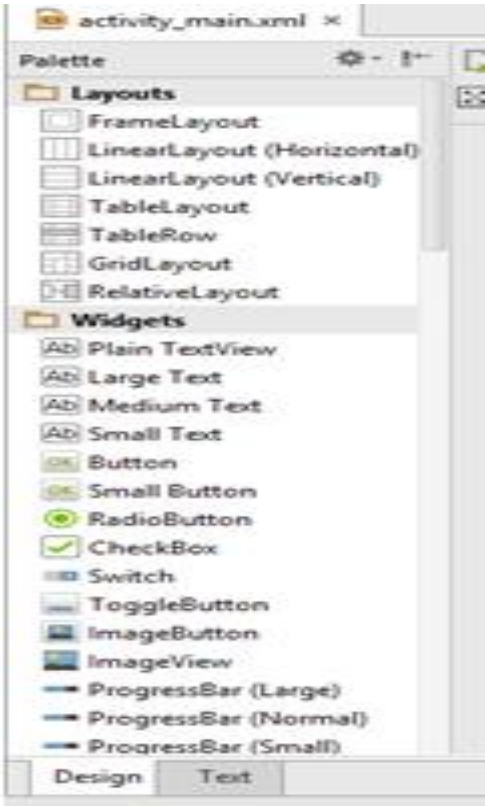
- Les interfaces sont composées d'objets héritant des classes View et ViewGroup ;
- Une vue se dessine et permet les interactions avec l'utilisateur (boutton, textView…)
- Un groupe de vues contient d'autres vues (les gabarits conteneurs des vues)
  - Un groupe de vues organise l'affichage des vues qu'il contient
  - Un groupe de vues est une vue.
- Android fournit une collection de vues et de groupes de vues qui offrent :
  - les principales entrées (textes, champs de texte, listes, etc.) ;
  - différents modèles d'organisation (linéaire, etc.).
- Une classe est associée à chaque type de vue ou groupe de vue.

CONSTRUCTION D'UNE IHM



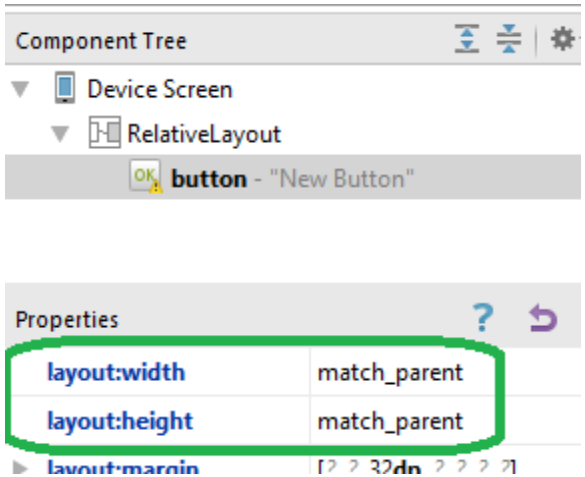
CONSTRUCTION D'UNE IHM

- On peut tout coder en Java dans l'activité
- Mais, en général, pour l'IHM on utilise plutôt les fichiers XML
- Par exemple, créer une nouvelle application Android
  - Ouvrir le fichier activity_main.xml (dans res/layout/activity_main.xml)
- Et on peut construire l'IHM en glisser déposer à partir d'une palette graphique.
- L'IHM est alors générée en XML



CONSTRUCTION D'UNE IHM

- On peut changer la largeur et la hauteur d'un composant à l'aide de ses propriétés
  - Exemple : layout:width et layout:height
- Les valeurs de ces propriétés peuvent être
  - match_parent (anciennement nommé fill_parent avant l'API 8) indique que le composant graphique sera aussi grand en largeur ou hauteur que son conteneur le lui autorise
  - wrap_content indiquant que le composant prend seulement la taille qui lui faut en largeur ou hauteur pour s'afficher correctement



Construction d'une IHM

- Pour positionner une propriété (= valeur d'attribut = données = ...) d'un composant graphique on peut le faire soit en XML soit par appel d'une méthode appropriée en java
- Dans le fichier XML, on positionne une propriété d'un composant graphique à l'aide d'une valeur d'attribut de sa balise XML
- Donc il y a une correspondance entre les noms d'attribut d'une balise XML et une méthode, pour positionner une propriété d'un composant graphique. On a, par exemple :
  - Android:background pour l'arriere plan
  - Android:alpha pour la transparence d'une vue (0 pour complètement transparent et 1 pour complètement opaque)

Construction d'une IHM

- Une fois l'IHM est totalement définie dans le fichier activity_main.xml on peut juste invoquer les composants dans le code java et les utiliser
- Le fichier activity_main.xml repère les composants par android:id

```
<Button
    android:id="@+id/button1"
    android:layout_width="fill_parent"
```

- Le composant est manipulé par cet identifiant dans le programme Java à l'aide de la méthode findViewById(R.id.*nomIdentifiant);*
- La valeur *nomIdentifiant est celle qui apparaît dans le fichier* activity_main.xml après @+id/
- Pour cet exemple ce sera findViewById(R.id.button1);

CONSTRUCTION D'UNE IHM

Exemple de code xml d'une IHM

```
<?xml version="1.0" encoding="utf-8"?>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
<TextView
            android:id="@+id/aller"
            android:text="Reservation Ticket"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
</ LinearLayout>
```

Les gabarits

- Des **ViewGroup** particuliers sont prédéfinis : ce sont les gabarits (layout en anglais) qui proposent une prédisposition des objets graphiques :
- Les principaux Layout Android sont :
  - LinearLayout (le modèle de boîte),
  - RelativeLayout (un modèle à base de règles),
  - TableLayout (le modèle de grille),
  - ScrollView, un récipient conçu pour aider à la mise en oeuvre des conteneurs de défilement.
  - Autres (ListView, GridView, WebView, MapView, ...)
- Les déclarations se font principalement en XML, ce qui évite de passer par les instanciations Java.

Les gabarits

- Les attributs des layouts permettent de spécifier des attributs supplémentaires. Les plus importants sont :
  - **android:layout_width** et **android:layout_heigh**
    - **="fill_parent" :** l'élément remplit tout l'élément parent,
    - **="wrap_content"** : prend la place minimale nécessaire à l'affichage ;
  - **android:orientation** : définit l'orientation d'empilement ;
  - **android:gravity** : définit l'alignement des éléments.

Les gabarits

Exemple

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

Les gabarits

- 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**.

```
public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);
setContentView(R.layout.acceuil);

}
```

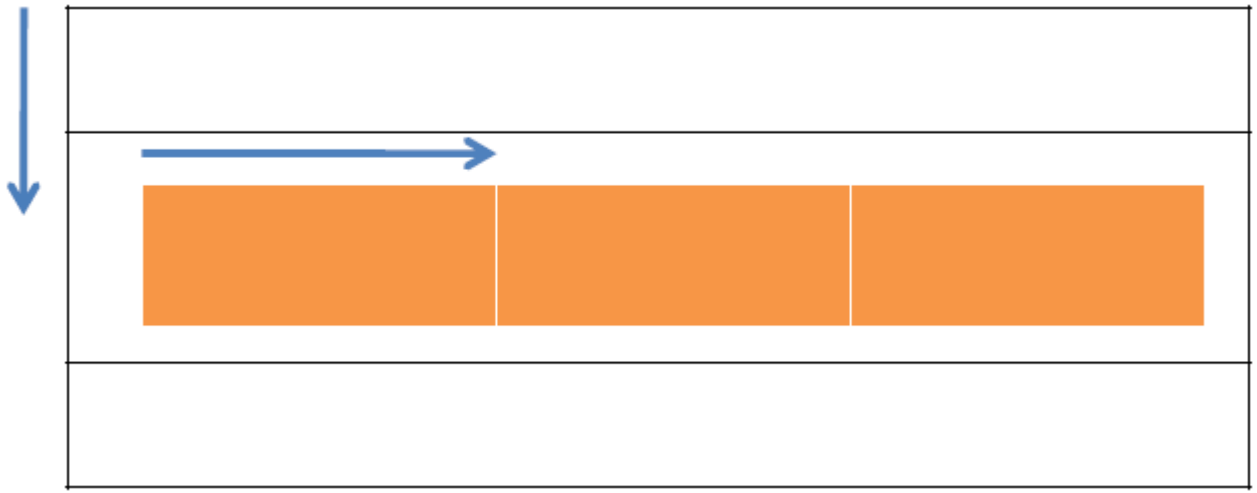
LES GABARITS

- 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"**).
- Un layout peut contenir des éléments graphiques, ou d'autres layouts.
- Les interfaces peuvent aussi inclure d'autres interfaces, permettant de factoriser des morceaux d'interface.
- on peut accéder à un layout par son id et agir dessus à travers du code Java :

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

Les gabarits: LinearLayout

- On peut dire que, *LinearLayout est l'outil de modélisation la plus courante.*
- Il propose une «boîte»



Les gabarits: LinearLayout

- Les composants à l'intérieur d'un LinearLayout sont rangés les uns à la suite des autres horizontalement ou verticalement
- Principales propriétés d'un LinearLayout : l'orientation, le mode de remplissage (fill model)
- L'orientation indique si le LinearLayout présente ces contenus sur une ligne (horizontalement) ou sur une colonne (verticalement)
- La propriété d'orientation à utiliser pour un LinearLayout dans le fichier XML est l'attribut android:orientation de la balise LinearLayout. Les valeurs possibles pour cette propriété sont vertical et horizontal
  - La valeur vertical indique que les contenus seront les uns en dessous des autres,
  - la valeur horizontal indique qu'ils seront les uns à la suite des autres
- L'orientation peut être modifiée à l'éxécution par la méthode setOrientation() lancée sur le LinearLayout en précisant la valeur LinearLayout.HORIZONTAL ou LinearLayout.VERTICAL

LES GABARITS: LINEARLAYOUT

- Tous les composants à l'intérieur d'un *LinearLayout doivent fournir les attributs dimensionnels android: layout_width et android: layout_height pour aider à résoudre la question de l'espace vide. Les valeurs utilisées pour définir la hauteur et la largeur sont les suivants:*
  - une valeur exacte de pixel (125px pour 125 pixels) : c'est fortement déconseillé
  - Fournir *wrap_content, ce qui signifie que le* composant *doit remplir son espace naturel, sauf si cela est trop grand, dans ce cas, Android peut utiliser* ***word-wrap au besoin pour le faire rentrer.***
  - Fournir match_parent (anciennement fill_parent avant l'API 8), *ce qui signifie que le* composant *doit remplir tout l'espace disponible dans son conteneur qui le renferme, après avoir pris en charges tous les autres* composants.

Les gabarits: LinearLayout

- Le poids est utilisé pour attribuer proportionnellement l'espace pour les widgets dans une vue. Vous définissez à *android: layout_weight une valeur (1, 2, 3, ...) pour indique quelle proportion de l'espace libre devrait aller à ce widget.*
- ***Exemple Les deux widgets :***
- ***TextView et botton ont été réglées dans l'exemple précédent. Ces deux widgets ont la propriété supplémentaire android: layout_weight = "1" alors que le widget EditText android: layout_weight = "2" NB : La valeur par défaut est de 0***

Les gabarits: LinearLayout

- **La proprieté** *weight indique le poids du composants dans le layout*
- Le poids est utilisé pour attribuer proportionnellement l'espace pour les composants dans une vue.
- Vous définissez à *android: layout_weight une valeur (1, 2, 3, ...) pour indique quelle proportion de l'espace libre devrait aller à ce widget.*

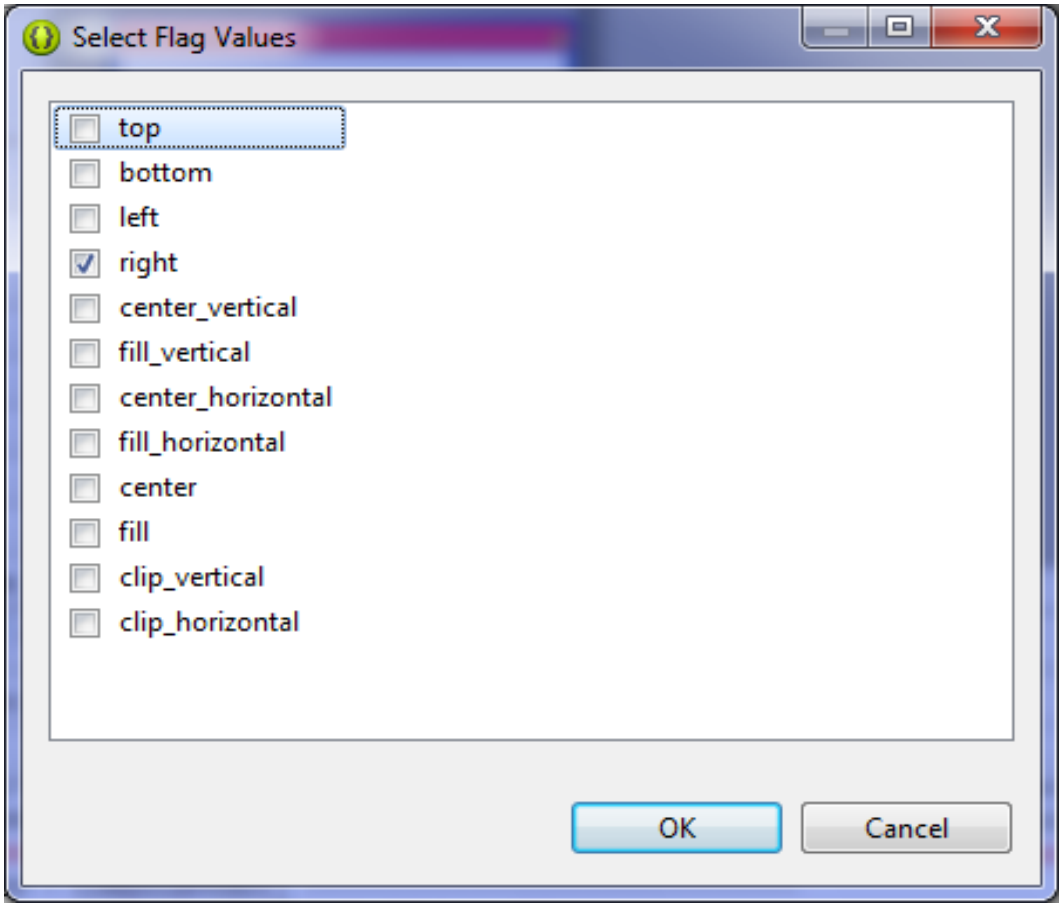
LES GABARITS: LINEARLAYOUT

- ***Exemple Les deux composants : TextView et botton ont la propriété supplémentaire android: layout_weight***
  - ***pour le botton = '1"***
  - ***pour l' EditText = "2"***

LES GABARITS: LINEARLAYOUT

- **La proprieté** *gravity* est utilisé pour indiquer comment un contrôle s'aligne sur l'écran.
- Par défaut, les composants sont de gauche et alignée en haut.
- Vous pouvez utiliser la propriété XML *android: layout_gravity = "..." pour définir d'autres modalités possibles: gauche (left) , centre (center) , droite (right) , haut (top) , bas (bottom), etc*

LES GABARITS: LINEARLAYOUT



LES GABARITS: LINEARLAYOUT

- *Il y a* une différence entre: ***android: gravity et android:layout_gravity***
- ***android: gravity spécifie la manière de placer le contenu d'un objet, à la fois sur les axes : x (abcisses) –y (ordonnées) , à l'intérieur de l'objet lui-même.***
- android:gravity=*"center"* => *centrer le contenu du composant (par exemple centrer le texte)*
- android: layout_gravity=*"center«* => *centrer le composant lui-même dans son conteneur (layout)*

LES GABARITS: LINEARLAYOUT

- *La proprieté Padding* spécifie combien y a t'il d'espace entre les limites de la "cellule" composant et le contenu du composant réels.
- Si vous souhaitez augmenter l'espace interne entre les bords du composant et de son contenu, utilisez la propriété android:padding ou en appelant *setPadding () lors de l'exécution de Java*
- **La proprieté layout_margin spécifie la marge EXTERNE**
- Par défaut, les widgets sont serrés les uns à côté des autres. Pour augmenter l'espace entre eux on utilise l'attribut **android:layout_margin**
- Exemple: **android:padding="*30dip*"**

LES GABARITS: RELATIVELAYOUT

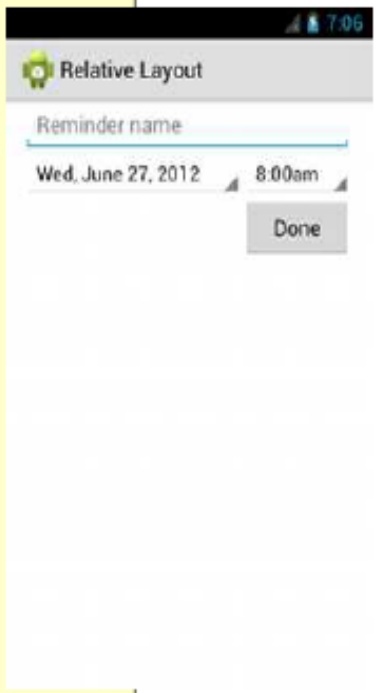
- Un RelativeLayout est un conteneur qui permet de placer ses contenus les uns par rapport aux autres
- C'est le conteneur proposé dans le (nouveau) outil de construction d'IHM
- Les Views contenues dans le RelativeLayout indiquent leur positionnement à l'aide de leurs attributs (dans le fichier XML de l'IHM)
- Il ne doit pas y avoir de dépendance cyclique (bon sens)
- Les Views indiquent leur position par rapport à la vue parente ou leurs Views soeurs (en utilisant leur id)
- Les valeurs des attributs sont soit des boolean, soit l'id d'une autre View

LES GABARITS: RELATIVELAYOUT

- Les Views dans un RelativeLayout peuvent utiliser les attributs :
  - android:layout_alignParentTop : si true, le haut de la View est calé sur le haut de la vue parente
  - android:layout_centerVertical : si true, la View est centrée verticalement à l'intérieur de la vue parente
  - android:layout_below : le haut de la vue est en dessous de la View indiquée (par son l'id)
  - android:layout_toRightOf : le coté gauche de la vue est à droite de la View indiquée (par son l'id)

LES GABARITS: RELATIVELAYOUT

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="wrap_content"
        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>
```



LES GABARITS: RELATIVELAYOUT

- Pour se référer au conteneur :
  - android: layout_alignParentTop indique que le haut du composant doit être aligné avec le haut du conteneur
  - android: layout_alignParentBottom en bas du composant doit être aligné avec le bas du conteneur
  - android: layout_alignParentLeft côté gauche du composant doit être aligné avec le côté gauche du conteneur
  - android: layout_alignParentRight côté droit du composant doit être aligné avec le côté droit du conteneur
  - android: layout_centerInParent le composant doit être positionnée à la fois horizontalement et verticalement au centre du conteneur
  - android: layout_centerHorizontal le composant doit être positionné horizontalement au centre du conteneur
  - android: layout_centerVertical le widget doit être positionnée verticalement au centre du conteneur

LES GABARITS: RELATIVELAYOUT

- Pour se référer à d'autres composants:
  - android: layout_above indique que le composant doit être placé au-dessus du composant référencé dans la propriété
  - android: layout_below indique que le composant doit être placé au-dessous du composant référencé dans la propriété
  - android: layout_toLeftOf indique que le composant doit être placé à la gauche du composant référencé dans la propriété
  - android: layout_toRightOf indique que le composant doit être placé à la droite du composant référencé dans la propriété
  - android: layout_alignTop indique que le haut du composant doit être aligné avec le haut du composant référencé dans la propriété
  - android: layout_alignBottom indique que le bas du composant doit être aligné avec le bas du composant référencé dans la propriété
  - android: layout_alignLeft indique que le gauche du composant doit être aligné avec la gauche du composant référencé dans la propriété
  - android: layout_alignRight indique que le droit du composant doit être aligné avec la droite du composant référencé dans la propriété
  - android: layout_alignBaseline indique que les lignes de base des deux composants doivent être alignés
- Exemple: **android: layout_below = "@ + id /ediUserName"**

LES GABARITS: TABLELAYOUT

- Le TableLayout agence les widgets sur un quadrillage exactement comme on pourrait le faire en HTML avec la balise <table>
- *Les lignes sont déclarées par vous en mettant widgets comme des fils d'un* ***TableRow à l'intérieur du TableLayout globale.***

LES GABARITS: TABLELAYOUT

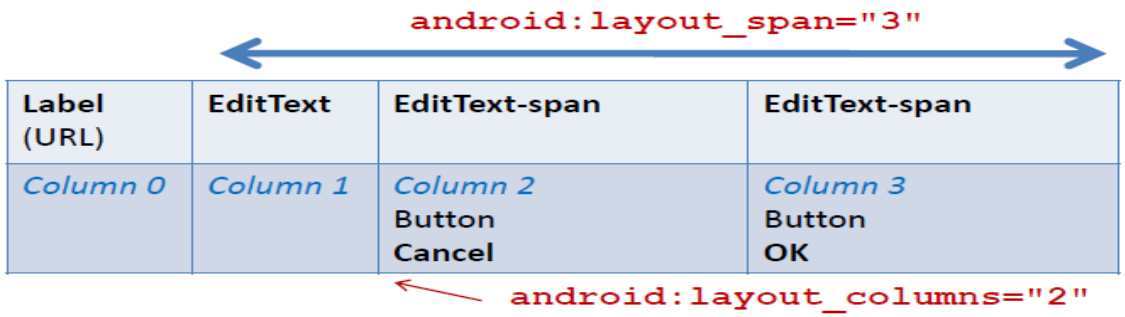
- ***Le nombre de colonnes est déterminé par Android (vous pouvez contrôler le nombre de colonnes d'une manière indirecte).***
- ***si vous avez trois lignes: une avec deux widgets, une avec trois widgets, et une autre avec quatre widgets, il y aura au moins quatre colonnes.***

0		1	
0		1	2
0	1	2	3

- un seul widget peut prendre jusqu'à plus d'une colonne en incluant : la propriété *androïd :layout_span, indiquant le nombre de colonnes des travées widget (ceci est similaire à l'attribut* ***colspan que l'on trouve dans les cellules du tableau en HTML)***

LES GABARITS: TABLELAYOUT

- Ordinairement, les widgets sont mis dans la première colonne de chaque ligne disponible.
- Dans l'exemple ci-dessous, l'étiquette ("URL") irait dans la première colonne (colonne 0, les colonnes sont comptées à partir de 0), et le *EditText irait dans un ensemble s'étend sur trois colonnes (colonnes 1 à 3).*



LES GABARITS: TABLELAYOUT


```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myTableLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff0033cc"
    android:orientation="vertical" >

    <TableRow>

        <TextView android:text="URL:" />

        <EditText android:id="@+id/ediUrl"
          android:layout_span="3"/>
    </TableRow>

    <View
        android:layout_height="3dip"
        android:background="#0000FF" />

     <TableRow>

        <Button
            android:id="@+id/cancel"
            android:layout_column="2"
            android:text="Cancel" />

        <Button
            android:id="@+id/ok"
            android:text="OK" />
    </TableRow>

    <View
        android:layout_height="3dip"
        android:background="#0000FF" />


</TableLayout>
```

S'étirer jusqu'à la colonne 3

Sauter les colonnes : 0, 1

LES GABARITS: TABLELAYOUT

- Par défaut, chaque colonne sera dimensionnée en fonction de la "naturelle" de la plus grande widget dans cette colonne.
- Si votre contenu est moins large que l'espace disponible, vous pouvez utiliser la propriété **TableLayout: android:stretchColumns="..."**
- Sa valeur doit être un nombre unique de colonne (0-based) ou une liste séparée par des virgules des numéros de colonne. Ces colonnes seront étirés de manière occuper tout l'espace encore disponible sur la ligne.
- Exemple : nous étirons les colonnes 2, 3, et 4 pour remplir le reste de la ligne.

```
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/myTableLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff0033cc"
android:stretchColumns="2,3,4"
android:orientation="vertical" >
<TableRow>
```

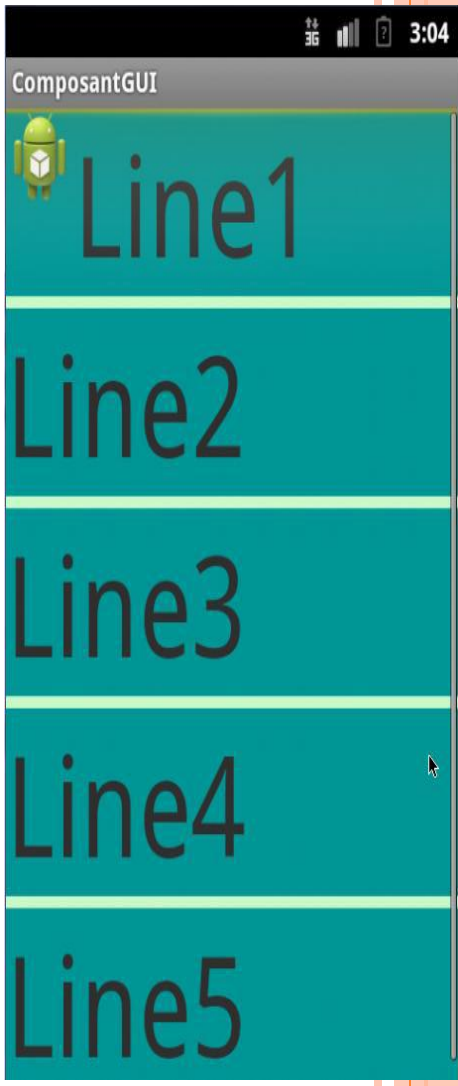
LES GABARITS: SCROLLVIEW LAYOUT

- Lorsque nous avons plus de données que ce qui peut être sur un seul écran, vous pouvez utiliser la *ScrollViewcontrol*.
- Il offre un accès glissement de défilement ou pour les données. De cette façon, l'utilisateur ne peut voir qu'une partie de votre mise en page en une seule fois, mais le reste est disponible via le défilement.
- Ceci est similaire à la navigation un page Web volumineuse qui oblige l'utilisateur à faire défiler la page pour voir la partie inférieure du formulaire.

LES GABARITS: SCROLLVIEW LAYOUT

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/myScrollView1"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff009999" >
<LinearLayout
android:id="@+id/myLinearLayoutVertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical" >
<LinearLayout
android:id="@+id/myLinearLayoutHorizontal1"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="horizontal" >
```

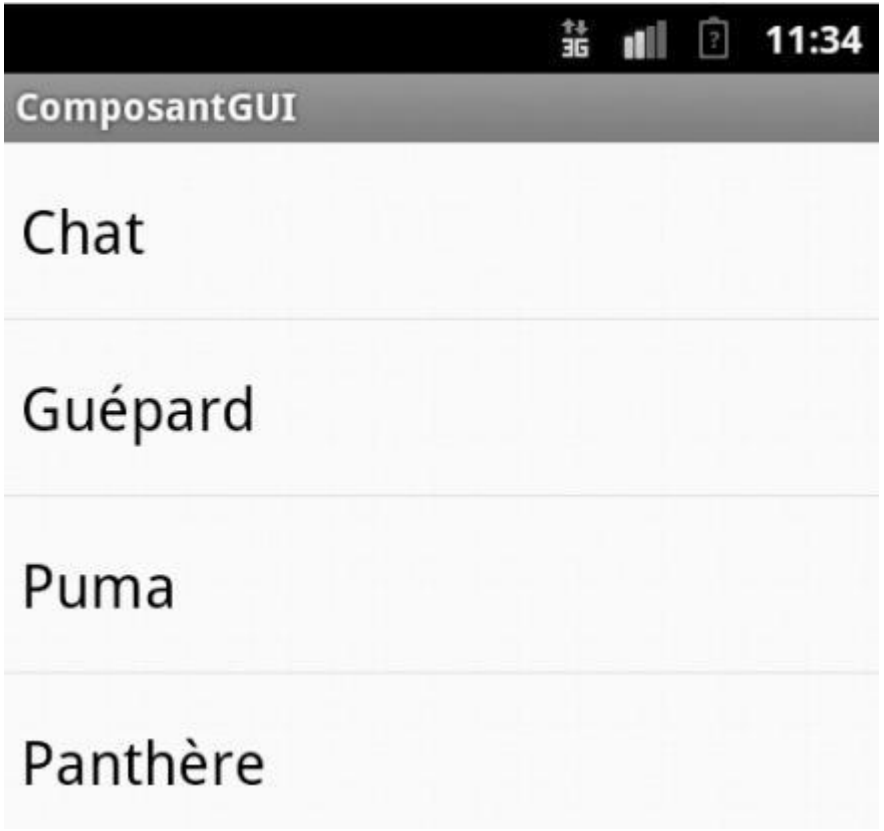
```
<ImageView
android:id="@+id/myPicture"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/ic_launcher"
/>
<TextView
android:id="@+id/textView1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Line1"
android:textSize="70dip" />
</LinearLayout>
<View
android:layout_width="fill_parent"
android:layout_height="6dip"
android:background="#ffccffcc" />
.....
<View
android:layout_width="fill_parent"
android:layout_height="6dip"
android:background="#ffccffcc" />
<TextView android:id="@+id/textView5"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Line5"
android:textSize="70dip" />
</LinearLayout>
</ScrollView>
```



LES GABARITS: LISTVIEW

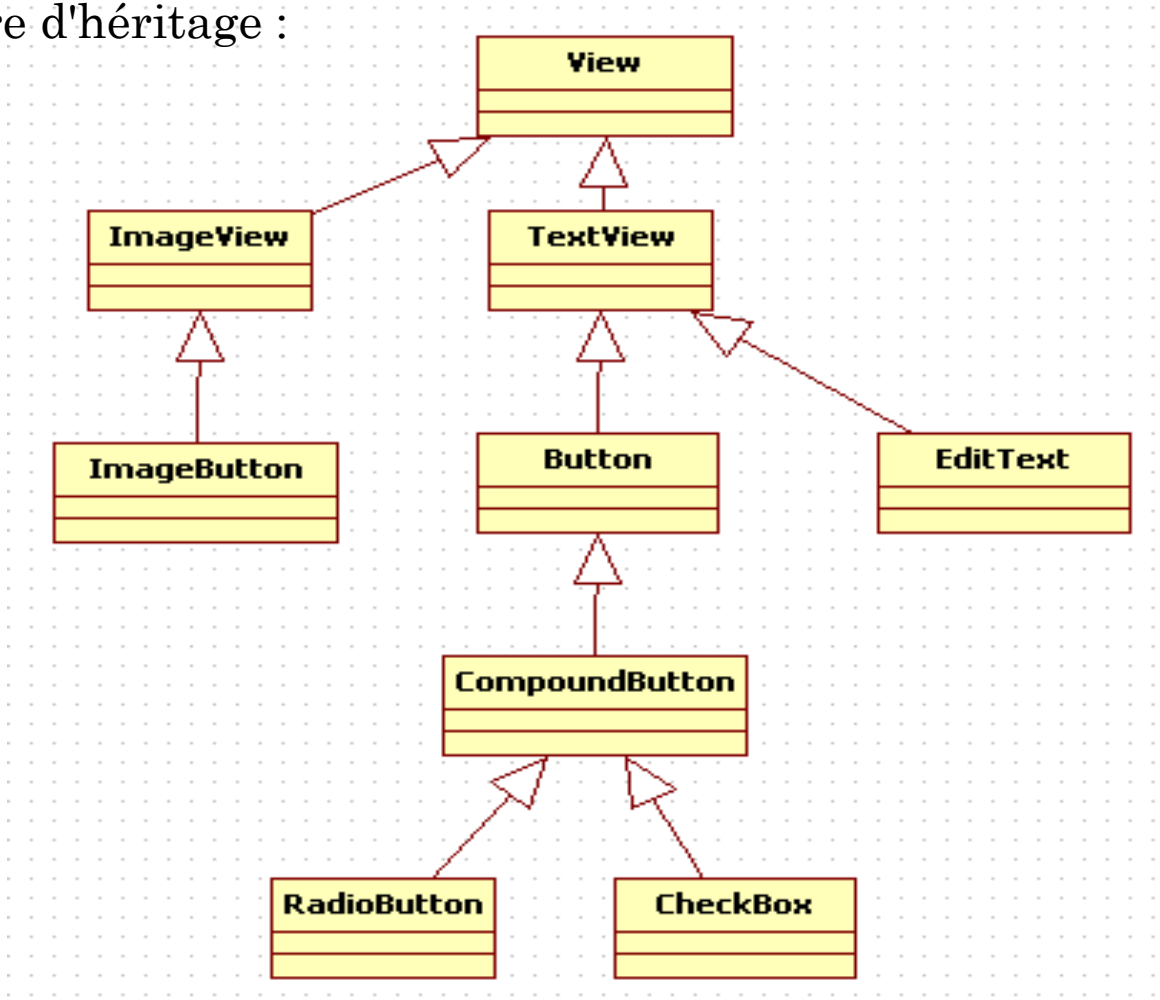
- *ListView place les composants qu'il contient verticalement, les uns après les autres et un seul par ligne : ils sont visuellement séparés par une ligne.*
- *Les composants sont fournis par une instance de ListAdapter. Si la hauteur cumulée de tous les composants ajoutés dépasse la taille de l'écran, la liste devient scrollable.*
- Ce layout pourrait, par exemple, être employé pour créer un menu avec sous-menu : les options du menu seraient des *TextView qui en cas de sélection afficheraient un autre ListView*.

LES GABARITS: LISTVIEW



Les Composants graphiques non conteneurs

Les classes de composants non conteneurs (contrôles) sont rangées dans l'arbre d'héritage :



Composants graphiques non conteneurs: textView

- Peut servir de zone de texte non éditable (~ Label AWT) et dans ce cas, sert souvent pour présenter les widgets qui suivent
- Propriétés importantes :
  - android:text : le texte du TextView
  - android:typeface : le type de police utilisée (monospace,...)
  - android:textStyle : le style (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)

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

Composants graphiques non conteneurs: ImageView

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

RÉFÉRENCES

- Cours de Jean-Marc Farinone, maître de conférences au Conservatoire National des Arts et Métiers (CNAM) de Paris
- Cours "Développement sous Android" de Jean-François Lalande, INSA, Institut National des Sciences Appliquées, Centre VAL DE LOIRE