



**Institut Universitaire de Technologie**

**Département Informatique**

Site de Bourg-en-Bresse



**ANDROID**

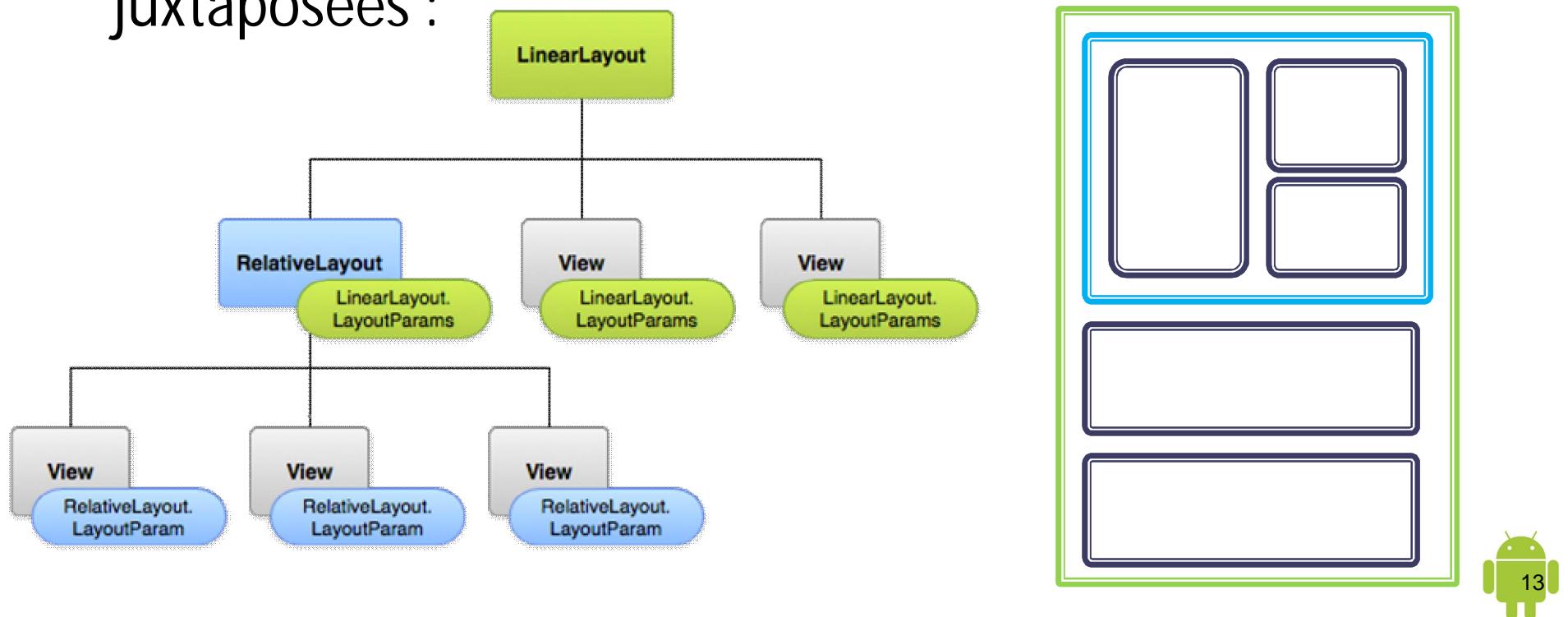
## **TP 2 : Layouts et résolutions d'écran**

**Semestre 4**

**[lionel.buathier@univ-lyon1.fr](mailto:lionel.buathier@univ-lyon1.fr)**

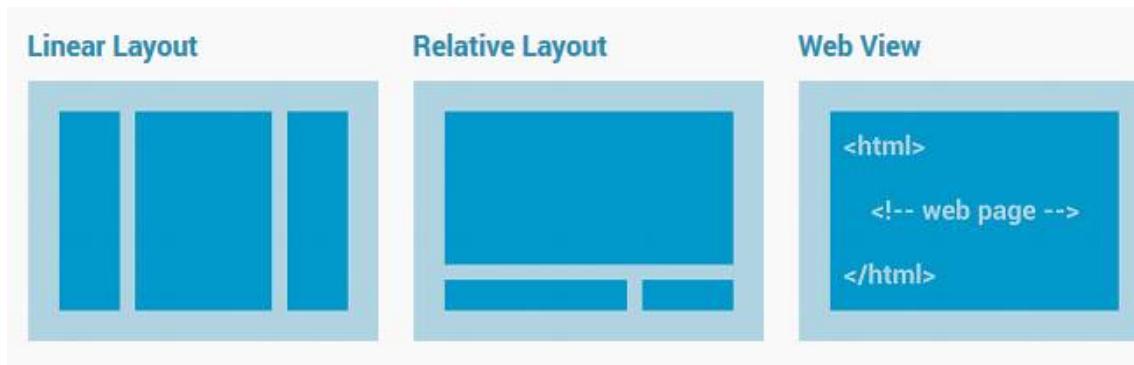
# Principe des conteneurs (Layout)

- Les vues sont constituées d'une associations hiérarchique de conteneurs (boites) imbriquées ou juxtaposées :



# Différents types de conteneurs

- Androïd définit des conteneurs :



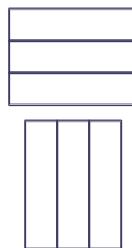
- relatifs (Relative Layout)
  - linéaires (Linear Layout)
  - de page web (FrameLayout, Webview)
  - en tableaux, liste (Grid Layout / ListLayout )
  - déroulants (ScrollView)
- <http://developer.android.com/guide/topics/ui/declaring-layout.html>



# Conteneurs linéaires (Linear Layout)

- Modèle reposant sur les imbrication de boites. Les attributs :

- Orientation
  - Vertical (en lignes)
  - Horizontal (colonnes)
  - Modification dynamique avec setOrientation()

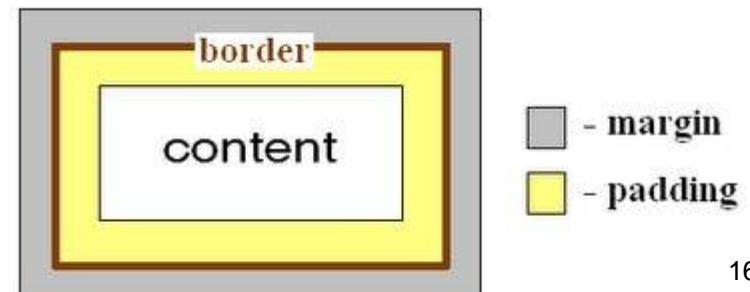


- Remplissage
  - match\_parent (fill\_parent est déprécié)
  - wrap\_content
  - taille fixe : par exemple 25 dp



# Conteneurs linéaires (Linear Layout)

- Poids (weight)
  - % de l'occupation du conteneur parent
  - nécessite que la dimension correspondante soit nulle (width ou height à 0dp)
- Gravité (gravity)
  - Indique comment s'aligner / à l'écran
  - En haut à gauche par défaut
- Marge intérieure (padding)
  - Marge de réserve à l'intérieur du conteneur
- Marge extérieure (marging)
  - Marge entre le conteneur et un autre conteneur (ou widget) extérieur



# Conteneurs relatifs (Relative Layout)

- Modèle reposant sur la position des conteneurs ou widgets relativement aux autres :
  - Dessus (android:layout\_above)
  - Dessous (android:layout\_below)
  - A gauche (android:layout\_toLeftOf)
  - A droite (android:layout\_toRightOf)
- Alignement :
  - En haut (android:layout\_alignTop), en bas (layout\_alignBottom),
  - A gauche, à droite,
  - Du texte (android:layout\_baseline) de différents widgets

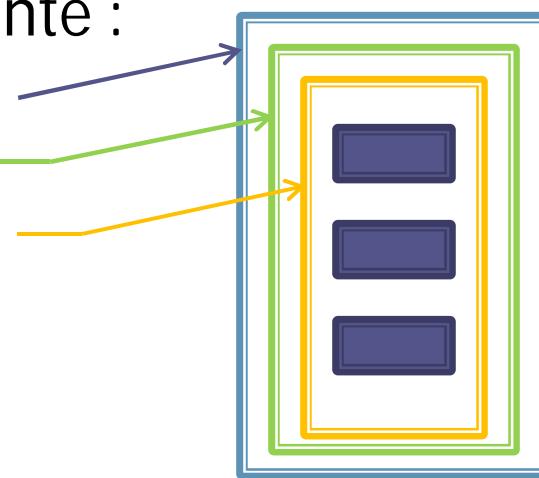


# Conteneurs en Tableau (Grid Layout / Table Layout )

- ▶ Les conteneurs ou widgets sont placés en tableau :
  - Pour GridLayout (depuis API 14):
    - ⇒ On précise le nombre de colonnes souhaitées (columnCount).  
Android fait le reste.
- ▶ TableLayout (déprécié) :
  - TableLayout contrôle le comportement global du conteneur
  - TableRow correspond à une ligne du tableau
  - Les colonnes sont gérées par Android, sachant que :
    - Le nbre de colonnes = nbre maxi de widgets sur une ligne du tableau
    - Un widget peut occuper plusieurs colonnes (layout\_span="2" pour 2 colonne)

# Conteneurs déroulants (ScrollView)

- ▶ Les conteneurs Linear et Relative ne peuvent pas être déroulants.
- ▶ Pour les rendre déroulants, il faut les encapsuler dans une ScrollView (qui ne peut contenir qu'un seul objet : Layout ou widget).
- ▶ Mais ScrollView n'est pas un conteneur principal, il doit lui-même être encapsulé dans un Layout...
- ▶ L'architecture est donc la suivante :
  - LinearLayout (ou RelativeLayout)
  - ScrollView
  - LinearLayout (ou RelativeLayout)
    - Widgets1
    - Widgets1
    - Etc.



# Exercices

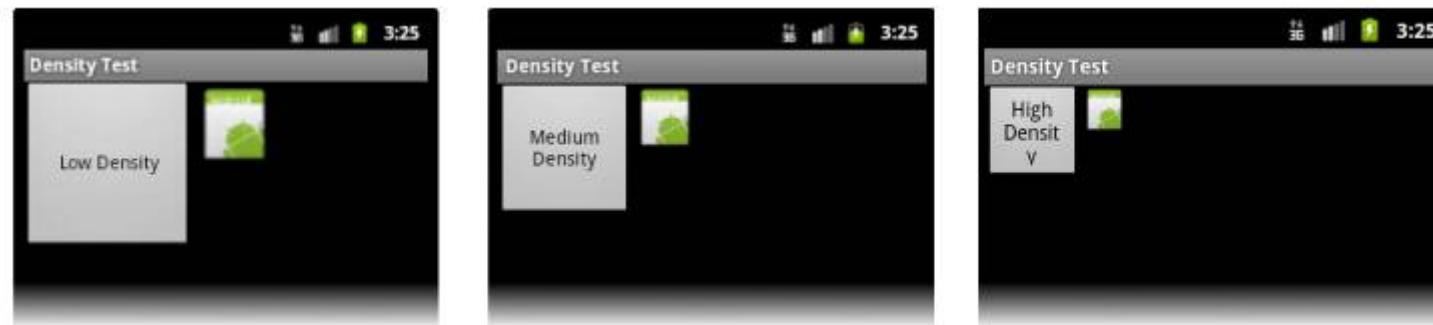
## ► Exercices

- Dessiner sur papier la vue correspondant au fichier linear\_layout.xml
- Dessiner celle correspondant au fichier relative\_layout.xml
- Dessiner celle correspondant au fichier grid\_layout.xml



# La gestion des résolutions d'écrans

- Grande disparité des terminaux Android
  - ⇒ gestion d'une multitude de tailles et de résolutions d'écran
  - ⇒ Une même icône 160x160 pixels aura une taille différente sur des écrans de résolutions différentes.



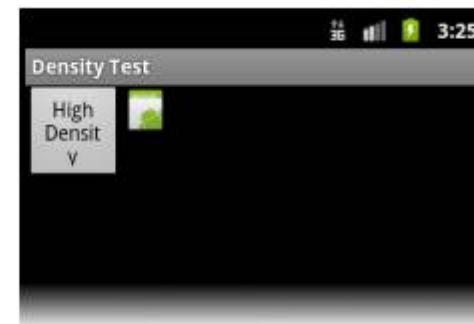
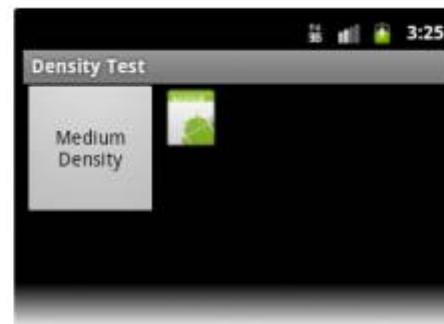
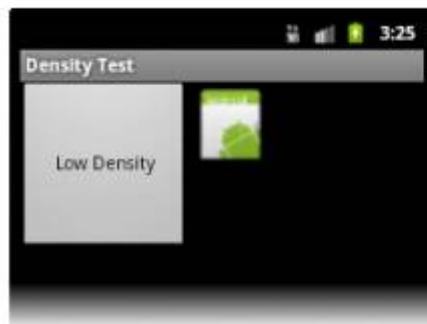
# La gestion des résolutions d'écrans

- Androïd palie ce problème grâce à unité virtuelle :
  - ⇒ le **dip** (density independant pixel) ou **dp** ⇒ pour les images
  - ⇒ le **sip** (scale-independant pixel) ou **sp** ⇒ pour le texte
- La référence est le mdpi : 1 px = 1 dp - (160 dp = 1 inch)
- Androïd traduit en pixel / à la densité réelle de l'écran :  
$$\text{px} = \text{dp} * (\text{dpi} / 160)$$
 Ex:  $0,75\text{px} = 1\text{dp} (120/160)$

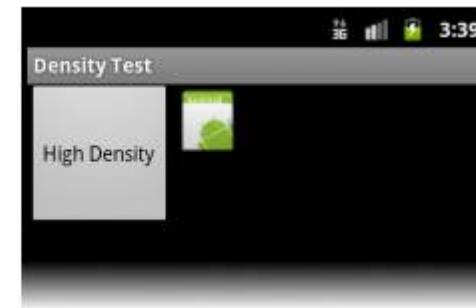
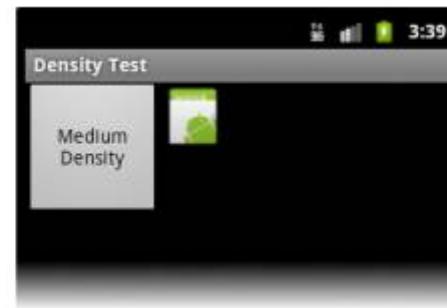
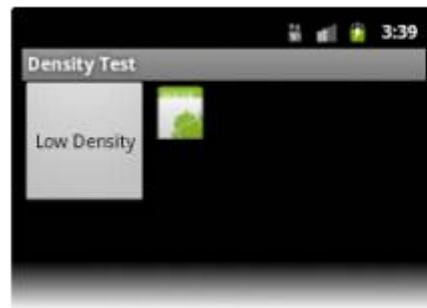
Density Bucket	Scaling Factor	Converted to Pixels	Physical Size
ldpi - 120 dpi	0.75 px/dp	$1\text{ dp} * 0.75\text{ px/dp} = 0.75\text{ px}$	$0.75\text{ px} / 120\text{ dpi} = 1/160\text{ in}$
mdpi - 160 dpi	1.0 px/dp	$1\text{ dp} * 1.0\text{ px/dp} = 1\text{ px}$	$1.0\text{ px} / 160\text{ dpi} = 1/160\text{ in}$
hdpi - 240 dpi	1.5 px/dp	$1\text{ dp} * 1.5\text{ px/dp} = 1.5\text{ px}$	$1.5\text{ px} / 240\text{ dpi} = 1/160\text{ in}$
xhdpi - 320 dpi	2.0 px/dp	$1\text{ dp} * 2.0\text{ px/dp} = 2\text{ px}$	$2.0\text{ px} / 320\text{ dpi} = 1/160\text{ in}$
xxhdpi - 480 dpi	3.0 px/dp	$1\text{ dp} * 3.0\text{ px/dp} = 3\text{ px}$	$3.0\text{ px} / 480\text{ dpi} = 1/160\text{ in}$

# La gestion des résolutions d'écrans

- Images définies en pixel :



- Images précédentes définies en dip :





**Institut Universitaire de Technologie**

**Département Informatique**

Site de Bourg-en-Bresse



android

## TP 3 : Utilisation des Widgets de base

Semestre 4

[lionel.buathier@univ-lyon1.fr](mailto:lionel.buathier@univ-lyon1.fr)

# Approche déclarative et programmatique

- ▶ Une vue est généralement définie de manière statique dans le fichier XML associé.
- ▶ Les widgets ainsi déclarés peuvent être instanciés et modifiés par le code Java de l'activity.
- ▶ *Exemple d'un widget TextView :*
  - *Dans Main.xml*

```
<TextView  
    android:id="@+id/textView1"  
    ... />
```
  - *Dans Main\_Activity.java :*

```
TextView tv = (TextView) findViewById(R.id.textView1);
```



# La documentation de référence

## ► Sur le site developer.android.com

- API : <http://developer.android.com/guide/topics/ui/controls.html>
- Ref. : <http://developer.android.com/reference/android/widget/package-summary.html>

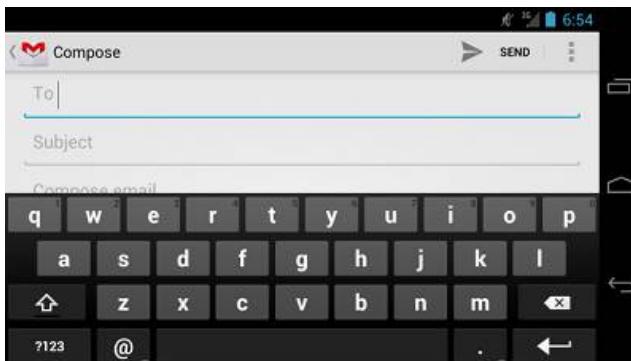


# TextView

- Déjà utilisé dans « Hello World »
- Permet d'afficher un texte prédéfini dans res/values/strings
- Nombreuse propriétés : style, couleur, taille, etc. (cf. api android)
- Le texte est modifiable par la méthode setText(), sur une instance de TextView



# Champs de textes: EditText & TextView



= TextView éditable

- Nombreuse propriétés : formats (chiffres, n° tel, etc.), nbre de lignes, auto-complétion, etc.
  - Attention : le type retourné par getText n'est pas une chaîne !!
- <http://developer.android.com/guide/topics/ui/controls/text.html>



# Boutons : Button & Image Button



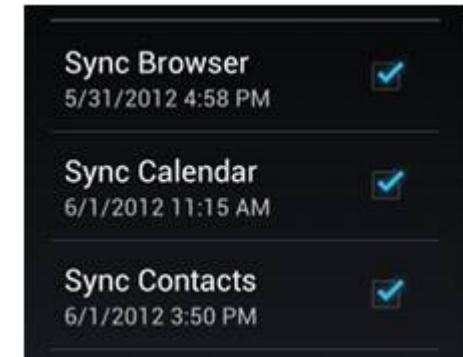
Button = TextView cliquable (hérite de textView)

- On peut implémenter un écouteur onClickListener() sur le Button
- <http://developer.android.com/guide/topics/ui/controls/button.html>
- ▶ Ou utiliser l'attribut onClick du bouton (dans le layout)
  - Attention à déclarer la méthode java correspondante comme suit :  
⇒ public void maGestionDuClick (View v)



# CheckBox

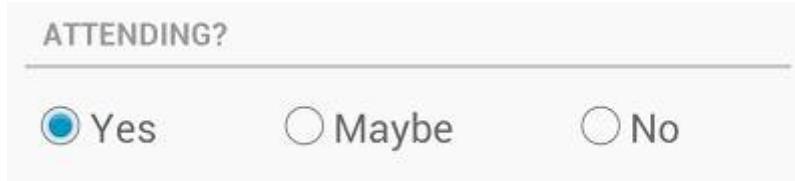
- = TextView cliquable (hérite de textView)
- Écouteur : onCheckedChangeListener
  - Associée à onCheckedChanged()
  - ou attribut android:onClick dans le xml
- Méthodes (sur CheckBox):
  - isChecked()      setChecked()      toggle()



► <http://developer.android.com/guide/topics/ui/controls/checkbox.html>



# RadioButton



- Idem CheckBox, mais l'utilisateur ne peut cocher qu'un seul radio-bouton parmi plusieurs regroupés dans un Layout de type RadioGroup
- Écouteur : RadioGroup.OnCheckedChangeListener
  - ⇒ getCheckedRadioButtonId() retourne l'ID du bouton coché
- On peut utiliser l'attribut onClick sur les RadioButtons (pas sur le RadioGroup)

► <http://developer.android.com/guide/topics/ui/controls/radiobutton.html>



## ImageView (et ImageButton)

- ▶ Équivalents de TextView et Button pour les images
  - Attribut android:src pour le chemin de l'image
  - ou méthode setImageResource  
(R.drawable.nom\_image)  
pour charger le contenu d'une image se trouvant dans un répertoire commençant par drawable.
    - ⇒ R.drawable suffit, même si l'image se trouve dans un répertoire drawable-ldpi par exemple.

<http://developer.android.com/reference/android/widget/ImageView.html>



# Exercice

- ▶ **Créer un formulaire avec**
  - **Une photo (avatar)**
  - **des boutons de choix (le sexe M, F )**
  - **Des champs d'édition pour le nom, le prénom, la date de naissance**
  - **le numéro de téléphone, l'adresse mail**
  - **Le code postal, l'adresse**
- **Lorsqu'on appuie sur un bouton de validation, afficher un message dans un Toast (ou une Alert Dialog) contenant toutes les informations saisies**
- <http://developer.android.com/guide/topics/ui/notifiers/toasts.html>
- <http://developer.android.com/guide/topics/ui/dialogs.html#AlertDialog>

