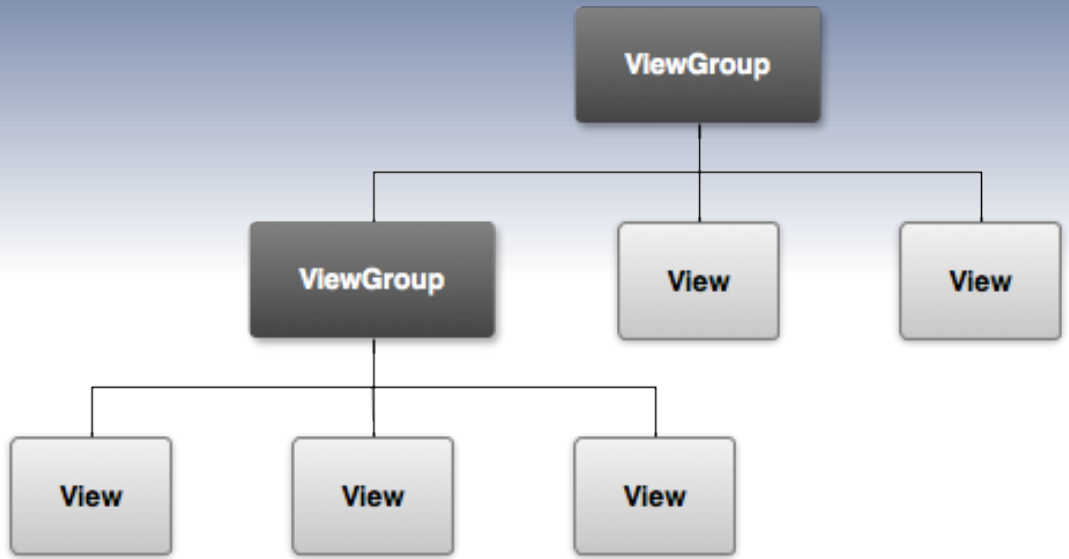
# Programmation Android
# II. GUI

# Plan

# Organisation d'une GUI Android




- **View** : élément de type *widget* (bouttons, champ texte, etc.)
  ▸ View
- **ViewGroup** : un type de **View** gérant d'autres View, par un mécanisme de mise en page : grille, liste verticale/horizontale, contraintes etc.
  ▸ ViewGroup

# Plan

# La classe android.view.View

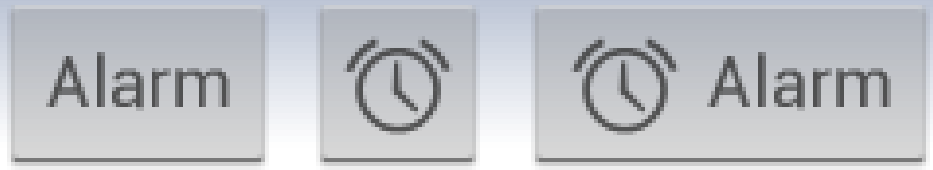
## Principes d'utilisation

- Toutes les *widgets* sont des sous classes de View
- Il est possible de créer une UI dans le code Java, mais on définit généralement une UI grâce à des fichiers XML
- À chaque type de View correspond ainsi une balise XML

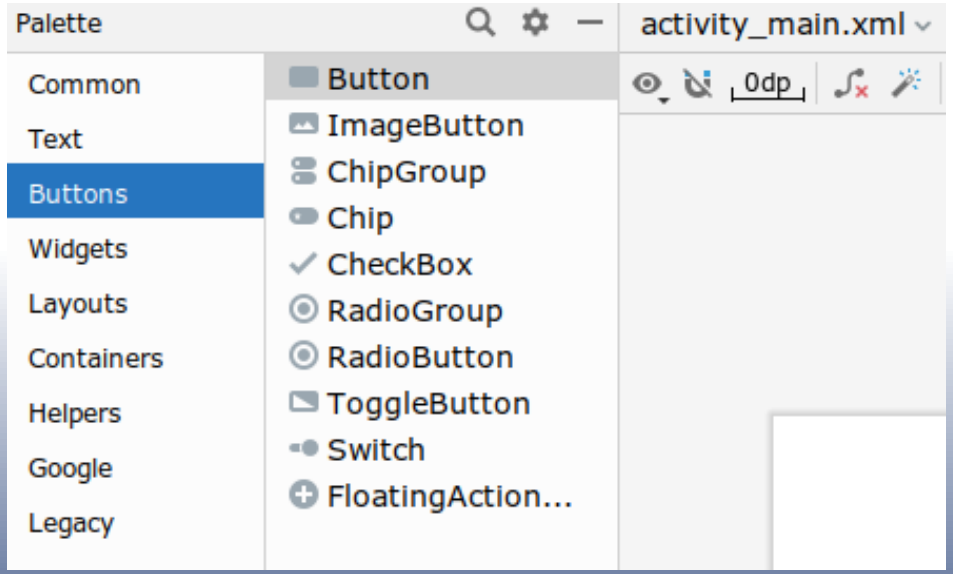
▸ classe Java Button en XML ⇒

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    ... />
```

# Exemple 1 : les différents types de boutons




récupérables dans l'éditeur :



# Texte uniquement

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    ... />
```

# Icône uniquement

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon"
    ... />
```

# Texte et icône

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:drawableLeft="@drawable/button_icon"
    ... />
```

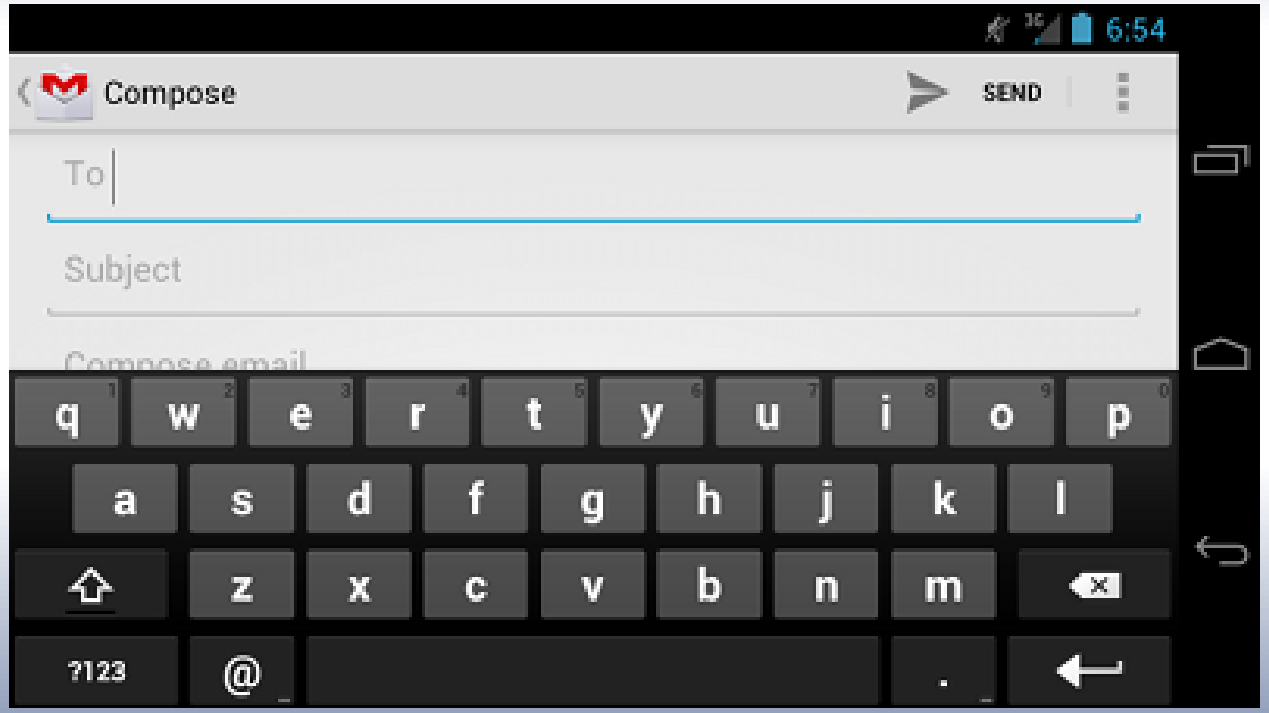
# CheckBox

```
<CheckBox
    android:id="@+id/checkBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/checkbox" />
```

etc.

# Exemple 2, *Text Fields* : < *EditText* >

Sert à créer une ligne de champ texte, par exemple la ligne `to` ci-dessous, avec diverses options possibles



# type de clavier : *android :inputType*

```
<EditText
    android:id="@+id/defaultTextField"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:inputType="text" />
```

Clavier par défaut :

# type de clavier : *android :inputType*

Pour la saisie d'un email :

```
<EditText
    android:id="@+id/email_address"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/email_hint"
    android:inputType="textEmailAddress" />
```

# *android :inputType = “phone”*

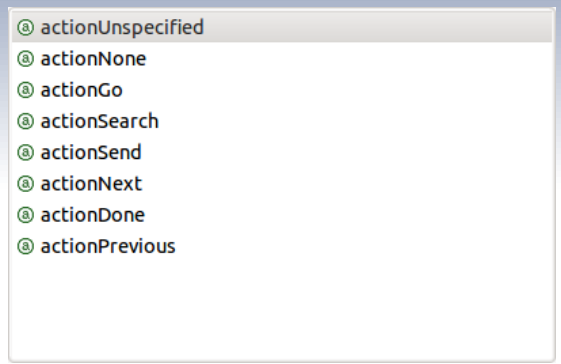
# Autres comportements du clavier

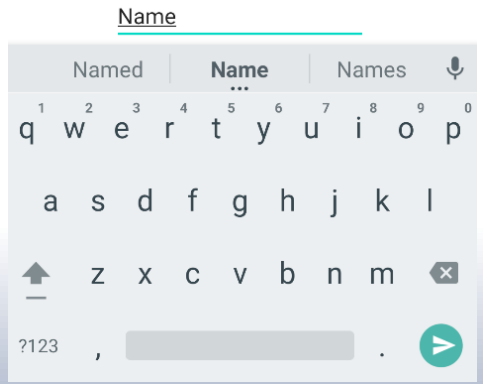
```
<EditText
    android:id="@+id/postal_address"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/postal_address_hint"
    android:inputType="textPostalAddress|
                       textCapWords|
                       textNoSuggestions" />
```

- Tous les possibilités pour android:inputType

# Icône de fin de saisie `imeOptions` : IME → Input Method Editor

```
<EditText
    android:id="@+id/search"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/search_hint"
    android:inputType="text"
    android:imeOptions="actionSend" />
```




▸ Toutes les possibilités pour *android:imeOptions*

Si rien n'est précisé, le focus passe à la *view* suivante dans l'UI : *textField* suivant, bouton, etc.

# Plan

# Clic sur bouton, solution 1 : XML

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="sendMessage"
    android:text="@string/button_send" />
```

- Ajout de l'attribut **android :onClick** à l'élément *Button*
- valeur : méthode définie dans l'activité contenant la *view*
- signature standardisée : ***public void*** et un paramètre de type ***View***

```
/** Called when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

# Solution 2
## dans le code Java (à préférer)

Ajout d'un écouteur (listener) au bouton, par exemple au moment de la création de l'activité, dans la méthode `onCreate` :

```
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

# Solution 3, Avec la librairie de `Data Binding` (`Android Jetpack`)

Jetpack est une surcouche des API Android qui facilite le développement ▸ Android Jetpack

Pour profiter du data binding, il faut modifier, si nécessaire, le fichier de build pour activer cette fonctionnalité ▸ plus d'information

To configure your app to use data binding, enable the `dataBinding` build option in your `build.gradle` file in the app module, as shown in the following example:

```
android {
    ...
    buildFeatures {
        dataBinding true
    }
}
```

# Solution 3

Une fois activée, une classe Java est créée automatiquement pour chaque fichier `layout` où le code suivant est rajouté : la balise `layout` englobante

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <layout xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:app="http://schemas.android.com/apk/res-auto"
4       xmlns:tools="http://schemas.android.com/tools">
5
6       <data/>
7
8       <androidx.constraintlayout.widget.ConstraintLayout...>
80
81  </layout>
```

# Solution 3

⇒ La classe Java possède un nom créé automatiquement à partir de celui du fichier `layout` :

**`activity_main.xml` ⇒ `ActivityMainBinding`**

Elle permet un accès direct aux éléments `View`

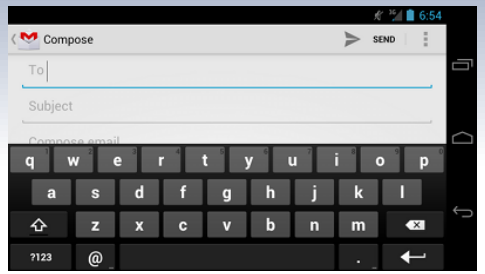
```
public class MainActivity extends AppCompatActivity {

    ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        binding.button.setOnClickListener(view -> {
            System.out.println("click !");
        });

    }
}
```

# Résumé du mécanisme d'interaction

## Principes généraux

- Chaque `View` est une ressource possédant un identifiant
- `android.app.Activity.findViewById (int id)` permet d'y accéder depuis le code Java
- À chaque type de `View` correspond un Listener

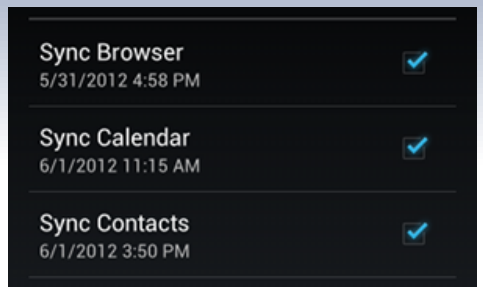
# Récupération des données après validation d'un champ texte :




À rajouter au moment de la créaction de l'activité, e.g. dans `onCreate` :

```
EditText editText = (EditText) findViewById(R.id.search);
editText.setOnEditorActionListener(new OnEditorActionListener() {
    @Override
    public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
        boolean handled = false;
        if (actionId == EditorInfo.IME_ACTION_SEND) {
            sendMessage();
            handled = true;
        }
        return handled;
    }
});
```

# Un listener n'est pas toujours nécessaire




Ici, seul l'état de la checkbox nous intéresse :

```
public class MyActivity extends Activity {
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);

        setContentView(R.layout.content_layout_id);

        final CheckBox checkBox = (CheckBox) findViewById(R.id.checkbox_id);
        if (checkBox.isChecked()) {
            checkBox.setChecked(false);
        }
    }
}
```

# Plan

# La classe `android.view.ViewGroup`

## Principes

- Un `ViewGroup` est un type de `View` pouvant contenir d'autres View, appelés *children*
- C'est la classe de base dont hérite les *layouts* et les *view containers*
- À chaque type de ViewGroup correspond une balise XML
- La différence est qu'elle pourra elle-même contenir des balises de type View

▸ plus d'information

# Layouts : gestion de la mise en page

## Layout : sous classe de ViewGroup

- Un **Layout** définit la manière dont les **Views** contenues sont disposées les unes par rapport aux autres.
- Des **ViewGroup** standards peuvent être créés avec du code XML

## Exemples :

- **RelativeLayout** : chaque **View** définit son déplacement par rapport à une autre **View**
- **LinearLayout** : disposition des éléments en 1 ligne ou 1 colonne dans l'ordre où ils sont définis dans le XML

# Exemple : `android.widget.LinearLayout`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

# Quelques Layouts usuels

### Linear Layout

A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

### Relative Layout

Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

### Web View

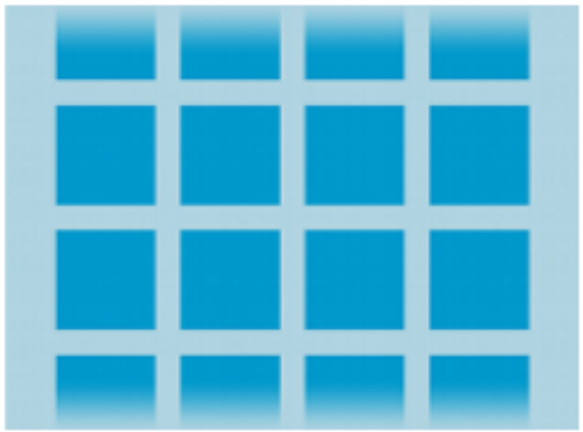

Displays web pages.

# Quelques Layouts usuels

**List View**

Displays a scrolling single column list.

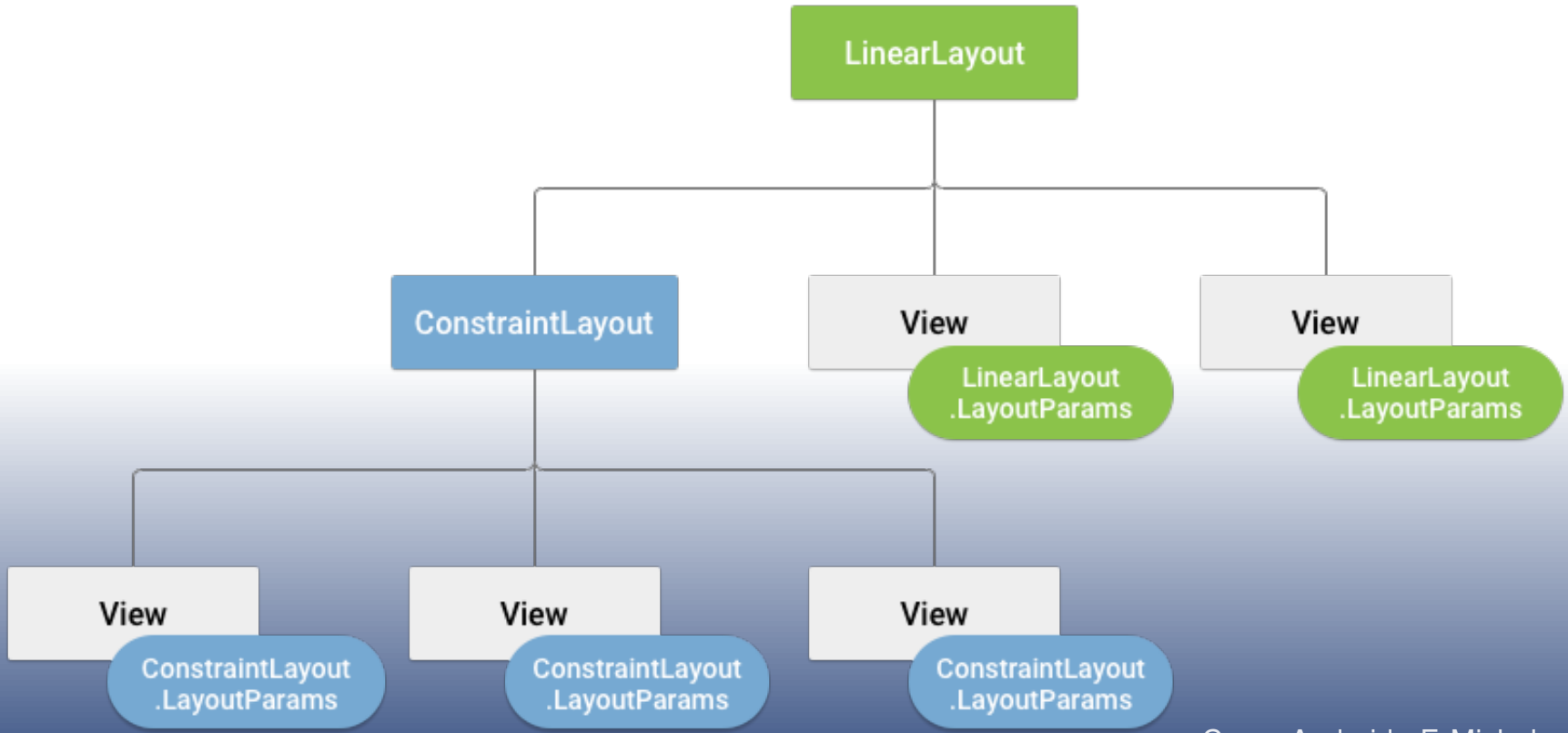
**Grid View**



Displays a scrolling grid of columns and rows.

# Mise en page hiérarchique

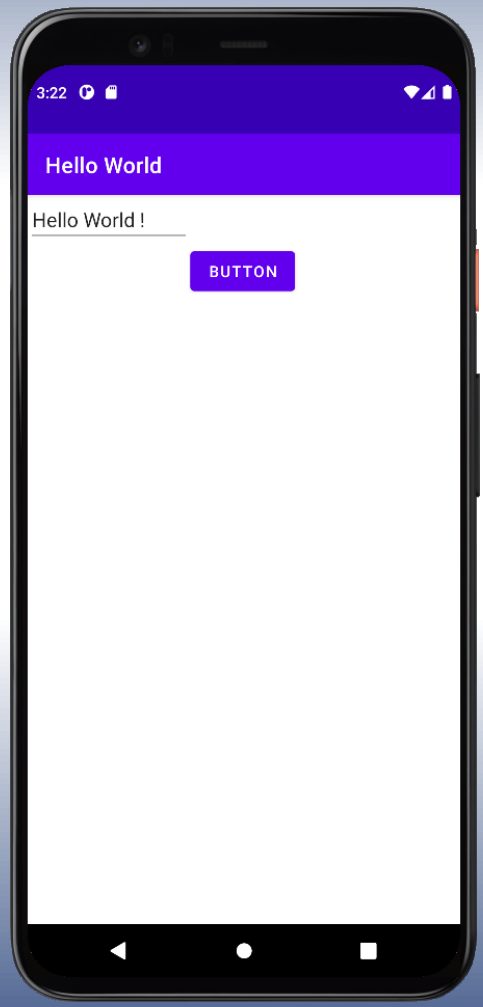
Chaque layout gère ses *children* à son niveau de l'arbre des composants, en fonction de son implémentation interne de `LayoutParams`. Exemple ▸ android.widget.LinearLayout.LayoutParams



# Exemples de code : RelativeLayout

relativelayout.xml | MainActivity.java

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginTop="15dp"
    android:layout_marginLeft="15dp"
    android:layout_marginRight="15dp"
    >

    <EditText
        android:id="@+id/editText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="7"
        android:inputType="textPersonName"
        android:text="Hello World !" />

    <Button
        android:id="@+id/button6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        android:layout_toRightOf="@id/editText"
        android:layout_below="@id/editText"
        />
</RelativeLayout>
```



# Exemples de code : LinearLayout

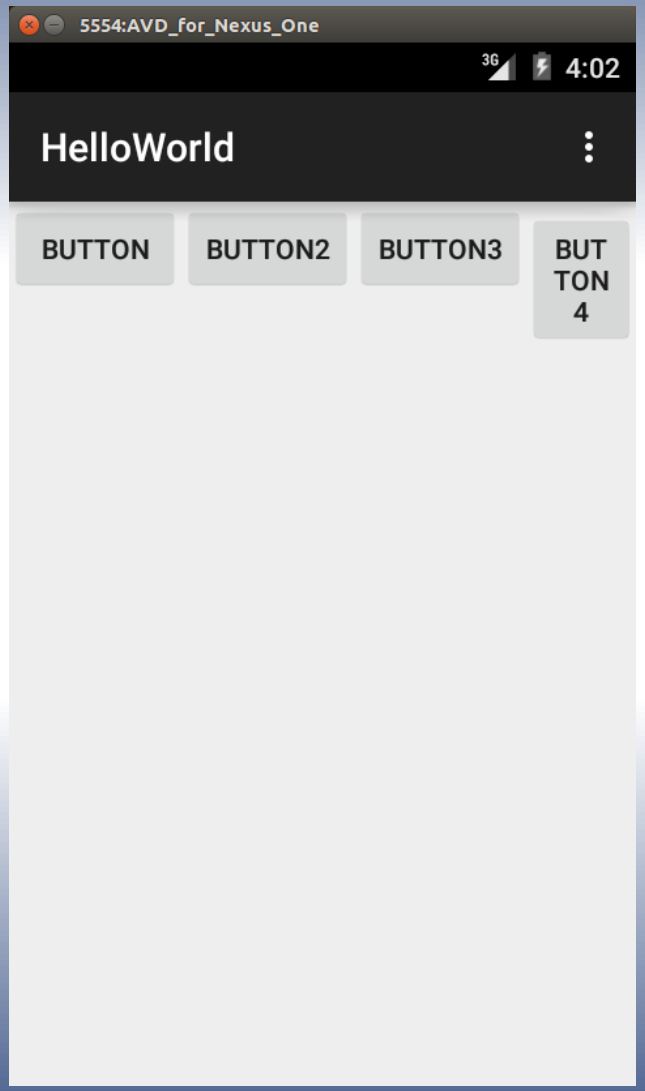
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
</LinearLayout>
```

# Exemples de code : LinearLayout

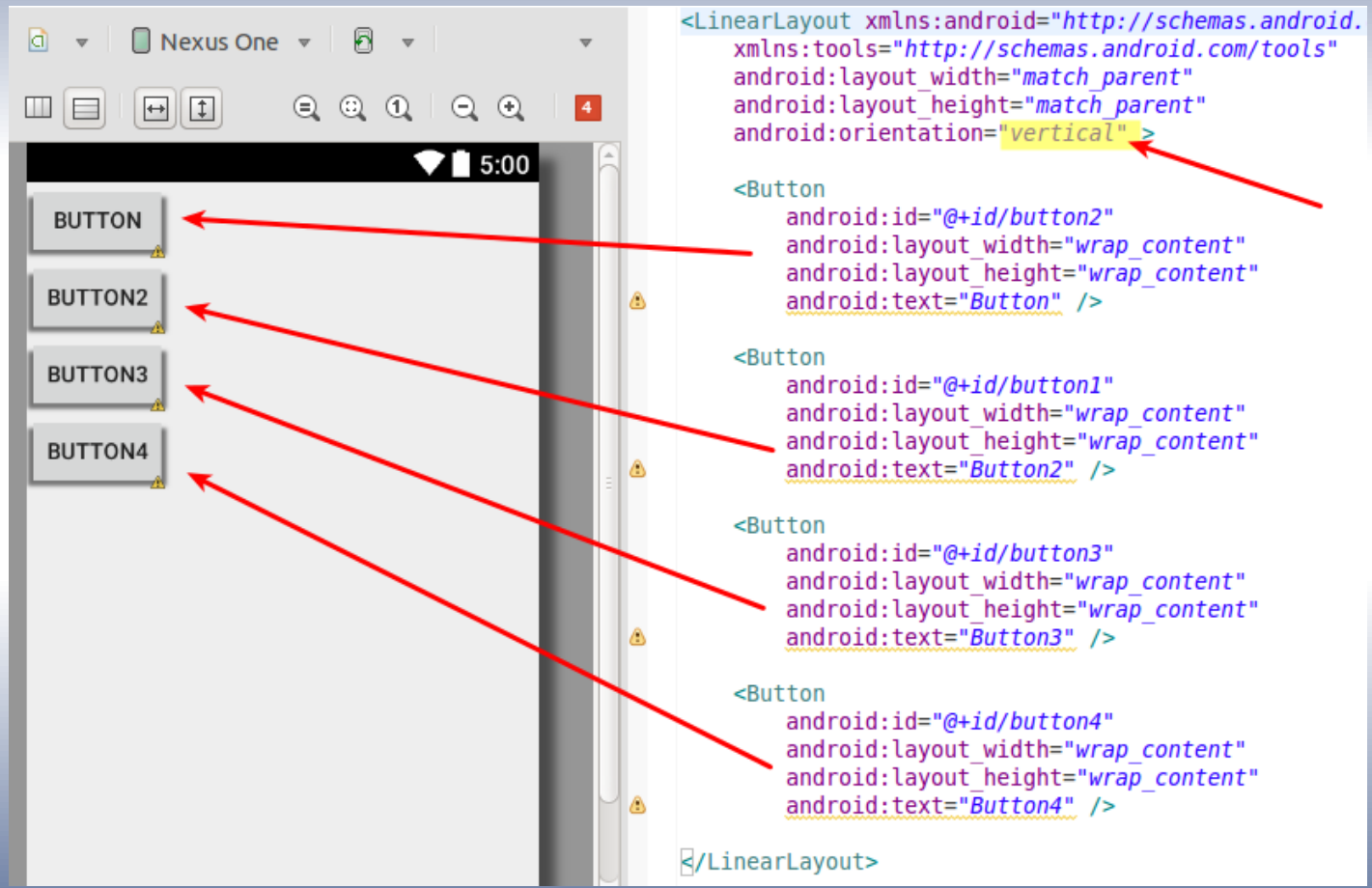
activity_main.xml

```
<LinearLayout 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:orientation="horizontal" >

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button2" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button3" />

    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button4" />

</LinearLayout>
```

# Exemples de code : LinearLayout



# Exemples de code : LinearLayout en vertical




```
<LinearLayout xmlns:android="http://schemas.android.
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button2" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button3" />

    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button4" />

</LinearLayout>
```

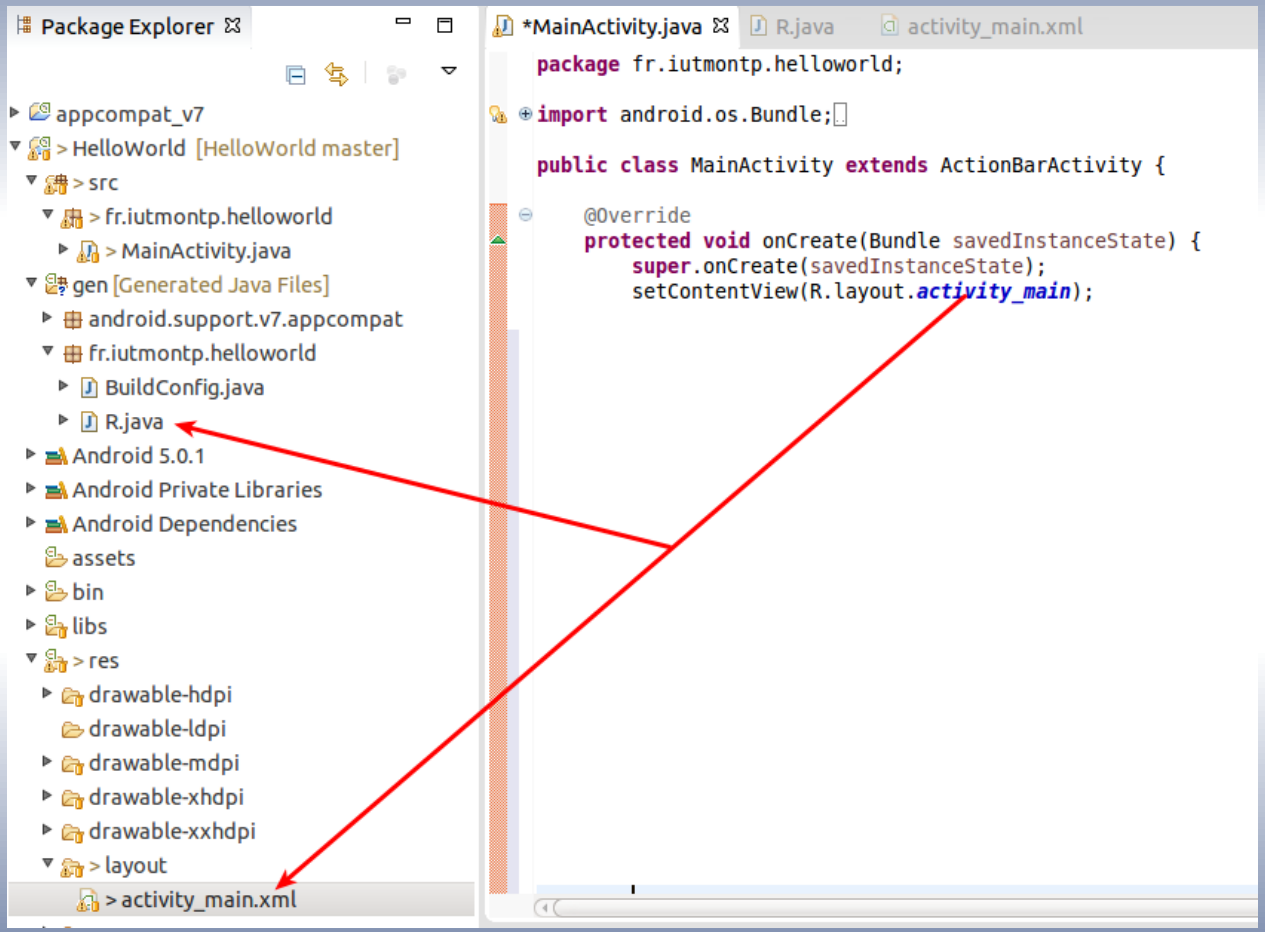
# Liaison d'un layout avec une activité

- La méthode `onCreate` est appelée automatiquement à la création de l'activité (cf. cours 4)
- **setContentView** permet de spécifier le layout associé à l'activité, c'est-à-dire le fichier XML définissant le **ViewGroup**
- Ce layout est identifié par un entier unique via la classe R : c'est une ressource de type layout.

*MainActivity.java

```
package fr.iutmontp.helloworld;

import android.os.Bundle;

public class MainActivity extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    }
```

# Liaison d'un layout avec une activité



# Plan

# Identifiant d'une ressource

```
<EditText android:id="@+id/edit_message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/edit_message" />
```

## android :id

- identifiant unique pour l'élément : permet l'accès à l'élément dans le code
- @ est utilisé pour faire référence à une ressource dans du code XML
- **+** nécessaire uniquement lors de la première déclaration ⇒ provoque la génération automatique d'un identifiant dans la classe **R**.

# Taille d'une View

```
<EditText android:id="@+id/edit_message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/edit_message" />
```

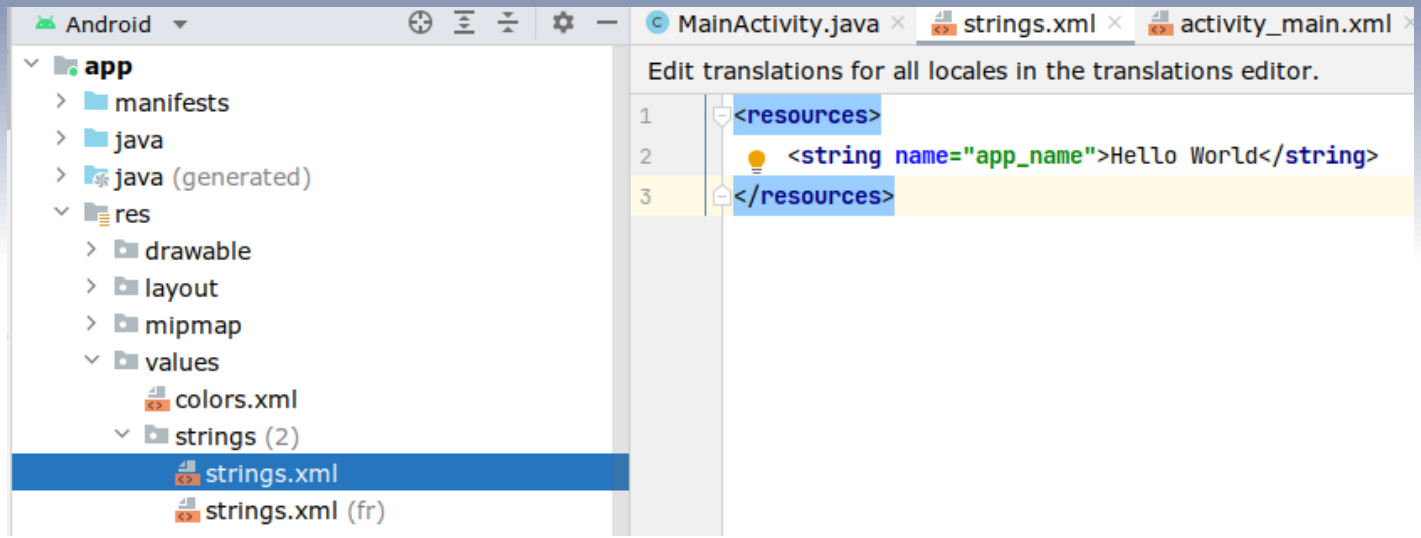
## android :layout_width et android :layout_height

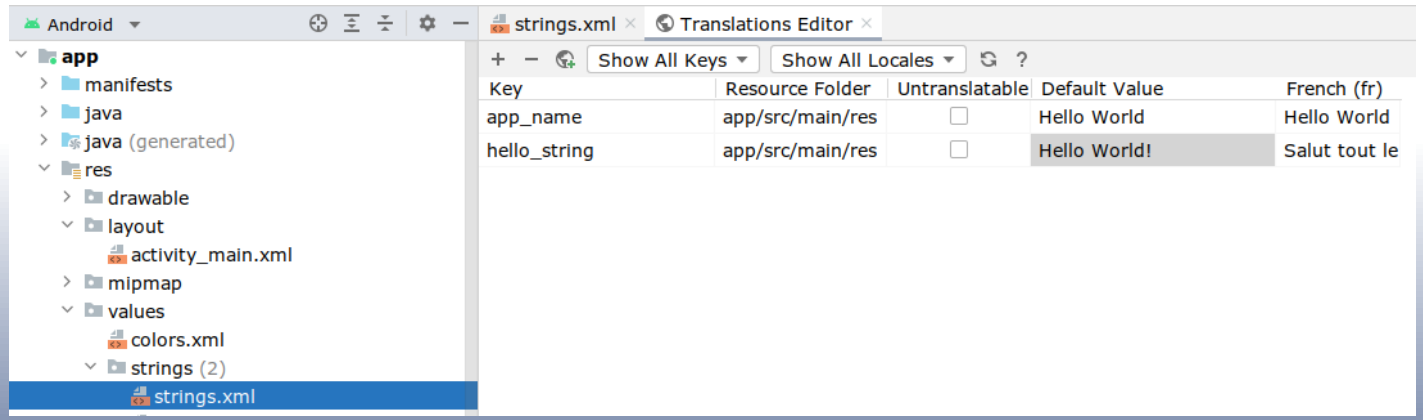
- permet de spécifier la taille d'une **View** :
  - relativement à la taille du contenu : **wrap_content**
  - relativement à la taille du **ViewGroup** parent : **match_parent**

## android :hint

- texte par défaut dans le TextField
- **Note** : pas de conflit de noms sur *edit_message*, **string** et **id** sont des types de ressources différents

# Parenthèse : ressources de type string



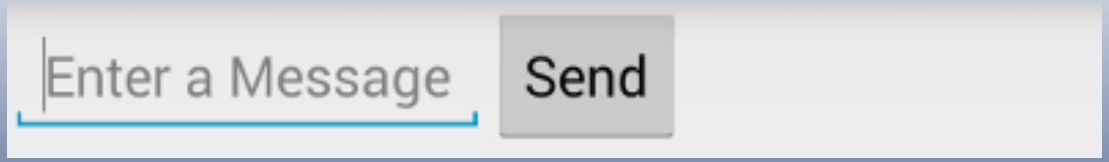




Intérêts : **centralisation** et **internationalisation**

# Taille d'une View

activity_main.xml | main.xml | MainActivity.java

```
<LinearLayout 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:orientation="horizontal" >

    <EditText
        android:id="@+id/edit_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send" />

</LinearLayout>
```



# Adaptation de la taille d'une View à l'espace disponible

```
<EditText
    android:layout_weight="1"
    ... />
```

## android :layout_weight

- permet de spécifier le ratio d'espace restant à occuper, en fonction du poids des autres View de même niveau
- 0 est la valeur par défaut de toutes les vues.
- si cette valeur est $>$ 0, la View prend un espace proportionnel aux autres valeurs, ou l'espace innocupé restant si les autres vues n'ont pas de poids

# Adaptation de la taille : optimisation

```
<EditText
    android:layout_weight="1"
    android:layout_width="0dp"
    ... />
```

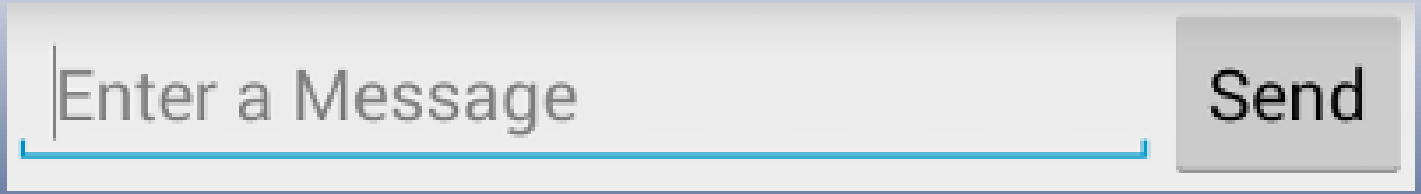
Spécifier la taille à 0 permet d'éviter un calcul qui ne sera pas utilisé, cest-à-dire la valeur de **wrap_content**.

# Adaptation de la taille

activity_main.xml | main.xml | MainActivity.java

```
<LinearLayout 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:orientation="horizontal" >

    <EditText
        android:id="@+id/edit_message"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send" />

</LinearLayout>
```

⇒



# Plan

# Aadaptation de l'UI au contexte

## Propriétés d'un écran

- **size** $\rightarrow$ **small**, **normal**, **large** ou **xlarge**
- **density** $\rightarrow$ **low** (ldpi), **medium** (mdpi), **high** (hdpi), **extra high** (xhdpi)

## Principe et gestion de l'adaptation

- Chaque layout ou bitmap est placé dans un sous répertoire de **res** ayant un nom lié à la taille et/ou à la résolution correspondantes.
- Note : le **changement d'orientation** (portrait ou paysage) est une **modification de la taille de l'écran**

# Gestion de différents layout

```
MyProject/
    res/
        layout/
            main.xml
        layout-large/
            main.xml
```

## Un layout par configuration

- Pour chaque taille à supporter : **un fichier layout de même nom**.
- Chaque configuration est placée dans un sous répertoire de **res** correspondant à la taille : **res/layout-**$< screen\_size >$ **e.g. res/layout-large.**
- **Par défaut, layout/ est utilisé pour l'orientation portrait.**

# Gestion de différentes orientation

```
MyProject/
    res/
        layout/                 # default (portrait)
            main.xml
        layout-land/            # landscape
            main.xml
        layout-large/           # large (portrait)
            main.xml
        layout-large-land/      # large landscape
            main.xml
```

# Gestion de différentes résolutions

## Exemple pour les images

- Il est important de fournir des icônes avec différentes résolutions
- À partir d'une image vectorielle, on génère 4 images ayant les rapports suivants : xhdpi : 2.0, hdpi : 16, mdpi : 1.0, ldpi : 0.75
- e.g. 200x200 xhdpi, 150x150 hdpi, 100x100 mdpi, 75x75 ldpi

# Gestion de différentes résolutions

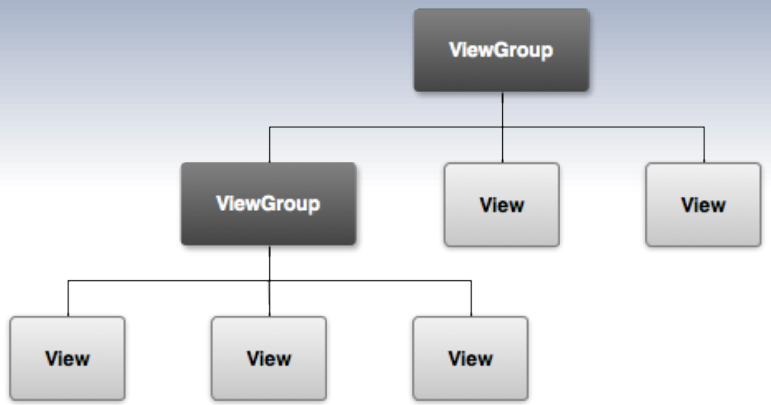
Idem, pour les sous répertoires de **res** :

```
MyProject/
    res/
        drawable-xhdpi/
            awesomeimage.png
        drawable-hdpi/
            awesomeimage.png
        drawable-mdpi/
            awesomeimage.png
        drawable-ldpi/
            awesomeimage.png
```

## Accès automatique à la ressource correspondante

- Dans le code, **`@drawable/awesomeimage`** sélectionnera automatiquement la ressource correspondant à la résolution courante.

# Résumé global




- **View** et **ViewGroup** pour l'UI
- Gestion des actions par ***listeners*** : (1) Xml ou (2/3) Java
- Adaptation : **gestion par répertoires standardisés**

Ce cours reprend largement les tutoriaux en ligne proposés par Google :

▸ Android developers