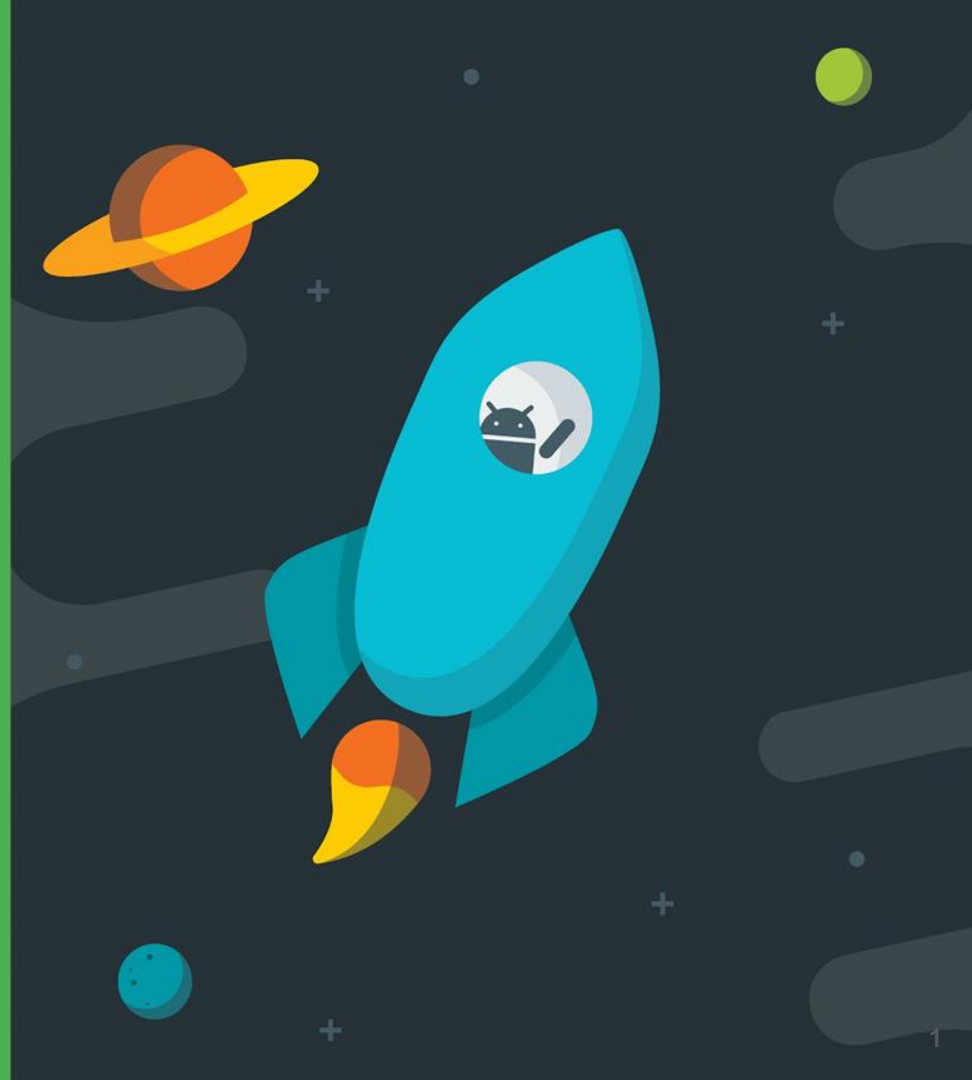


UNITÉ D'ENSEIGNEMENT (UE) :

DÉVELOPPEMENT MOBILE

CHAPITRE II :

# Éléments Graphiques Android



# ÉLÉMENTS GRAPHIQUES DE BASE

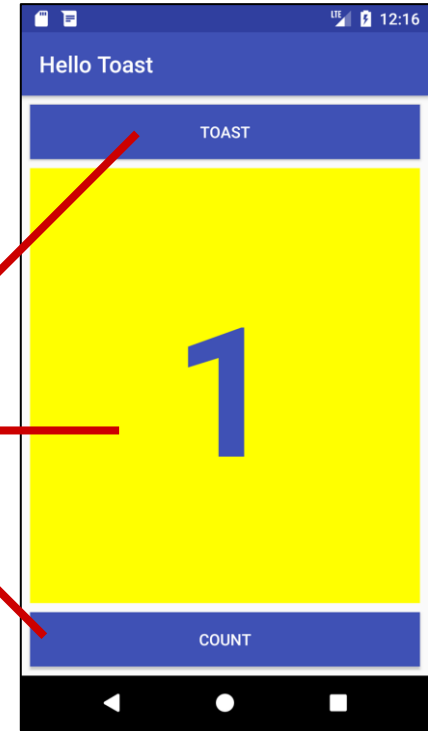
# Interface utilisateur (UI)

- L'interface utilisateur de votre application contient tout ce que l'utilisateur peut voir et avec lequel il peut interagir.
- Android fournit une variété de composants d'interface utilisateur prédéfinis, tels que des objets de mise en page structurés (**layout**) et des contrôles d'interface utilisateur, qui vous permettent de créer l'interface utilisateur graphique de votre application.
- Android fournit également d'autres modules d'interface utilisateur pour des interfaces spéciales telles que les boîtes de dialogue, les notifications et les menus.

# Tout ce que vous voyez est une vue (View)

Si vous regardez votre appareil mobile, chaque élément de l'interface utilisateur que vous voyez est une **View**.

Views



# Qu'est-ce qu'une vue ?

Les sous-classes de vue (View) sont des blocs de construction de l'interface utilisateur de base

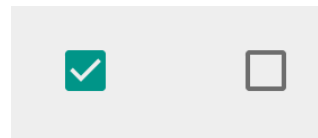
- Afficher le texte (classe `TextView`), modifier le texte (classe `EditText`)
- Boutons (classe `Button`), menus, autres contrôles
- Défilement (`ScrollView`, `RecyclerView`)
- Afficher les images (`ImageView`)
- Vues de groupe (`ConstraintLayout` et `LinearLayout`)

# Exemples de sous-classes de vues

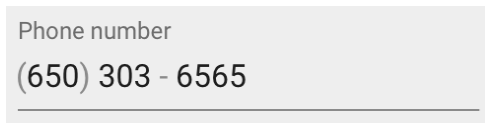
Button



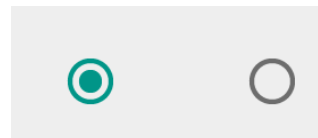
CheckBox



EditText



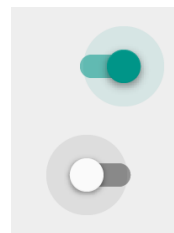
RadioButton



SeekBar



Switch



# Les attributs des vues

- Couleur, dimensions, positionnement
- Peut avoir le focus (par exemple, sélectionné pour recevoir l'entrée de l'utilisateur)
- Peut être interactif (répondre aux clics des utilisateurs)
- Peut être visible ou non
- Relations avec d'autres vues

# Les Composants View et ViewGroup

## . View :

- Classe de base pour la création d'une interface graphique en Android
- Tous les composants graphiques (boutons, images, cases à cocher, etc.) d'Android héritent de la classe View
- Une vue peut être ajoutée à une application :
  - Soit statiquement, dans le fichier de description de l'interface (Layout XML)
  - Soit dynamiquement, dans le code Java de l'application



# Définition de vue en XML

**<TextView**

```
    android:id="@+id/show_count"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:background="@color/myBackgroundColor"  
    android:text="@string/count_initial_value"  
    android:textColor="@color/colorPrimary"  
    android:textSize="@dimen/count_text_size"  
    android:textStyle="bold"
```

**/>**

# Les attributs de vue en XML

**android:<property\_name>=<property\_value>**

**Exemple:** android:layout\_width="match\_parent"

**android:<property\_name>="@<resource\_type>/resource\_id"**

**Exemple:** android:text="@string/button\_label\_next"

**android:<property\_name>="@+id/view\_id"**

**Exemple:** android:id="@+id/show\_count"

# Créer une vue en code Java

Dans une activité :

*context*



```
TextView myText = new TextView(this);  
myText.setText("Afficher ce texte!");
```

# Quel est le contexte ?

- Le contexte ([context](#)) est une interface vers des informations globales sur un environnement d'application
- Obtenez le contexte :

```
Context context = getApplicationContext();
```

- Une activité a son propre contexte :

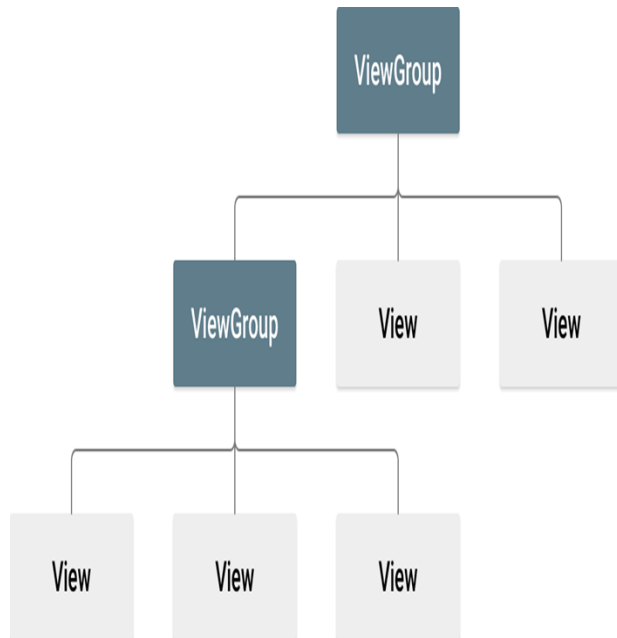
```
TextView myText = new TextView(this);
```

# Des vues personnalisées

- Plus de 100 types de vues différents disponibles à partir du système Android, tous les enfants de la classe [View](#)
- Si nécessaire, [créez des vues personnalisées](#) en sous-classant des vues existantes ou la classe View

# Les Composants View et ViewGroup

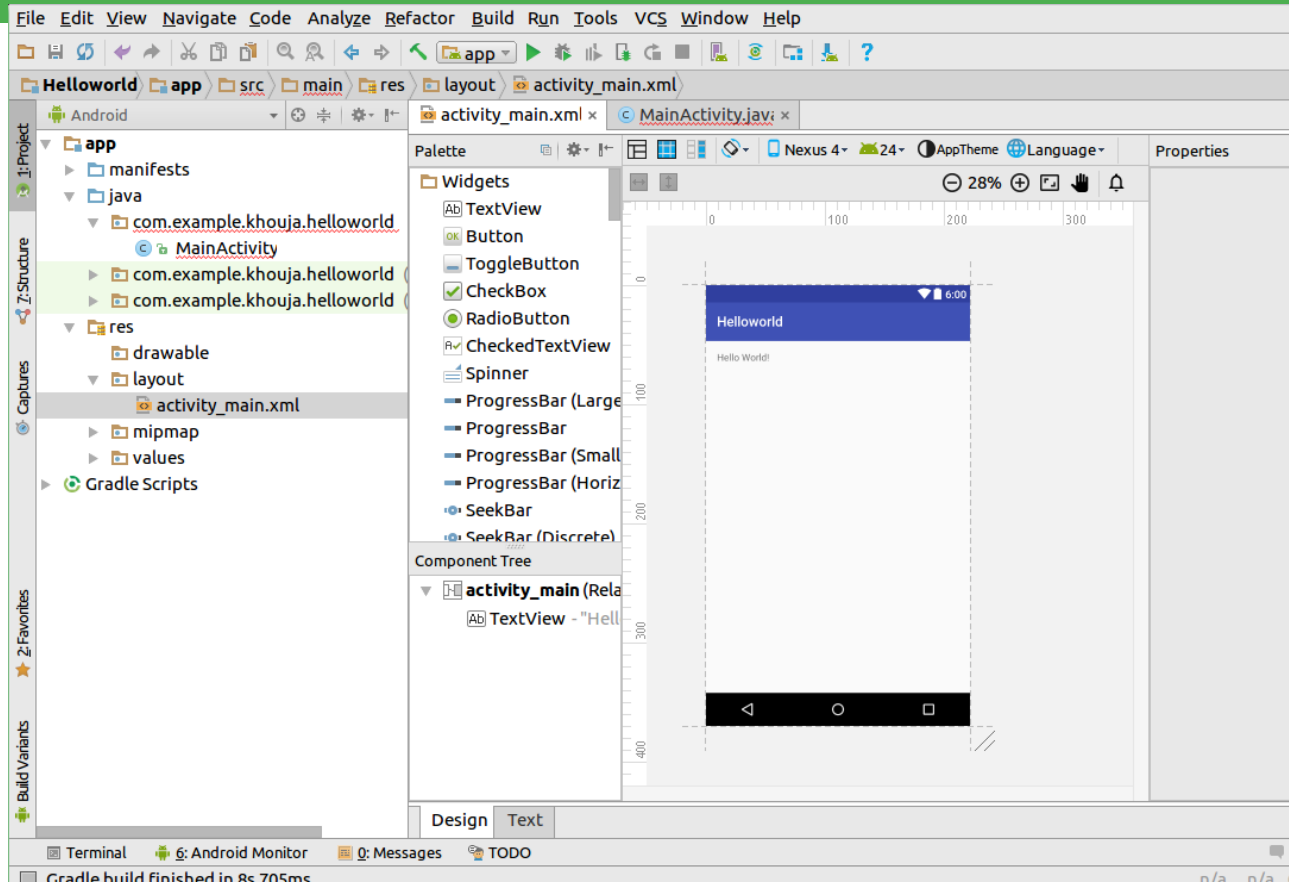
- ViewGroup :
  - Android offre la possibilité de regrouper plusieurs view dans une structure arborescente à l'aide de la classe **ViewGroup**
  - Cette structure peut à son tour regrouper d'autres éléments de la classe ViewGroup et être ainsi constituée de plusieurs niveaux d'arborescence
  - Classe de base de tous les layouts
  - **Layouts** : Conteneurs invisibles rassemblant plusieurs **View** ou **ViewGroup**



# Layout XML

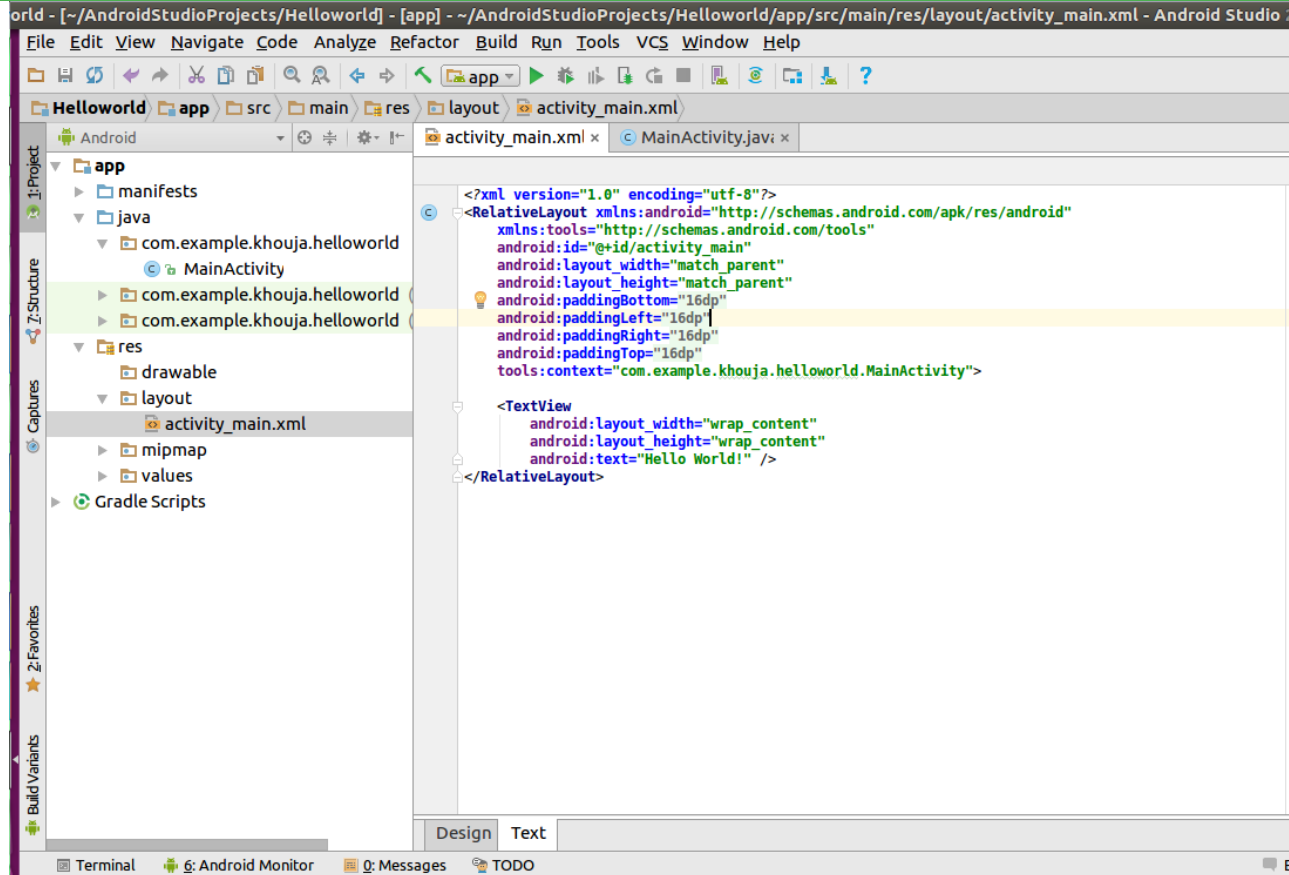
- Ensemble de fichiers XML de spécification des différents composants graphiques d'une application (widgets) et de leurs relations entre eux et avec leurs conteneurs ( layouts )
- Placés dans le répertoire res/layout de l'arborescence du projet
- Possibilité de visualiser le contenu du fichier:
  - Sous forme de code XML (onglet Text )
  - Sous forme graphique (onglet Design ) : mise à disposition d'une palette de composants graphiques à glisser-déplacer vers l'emplacement désiré
- Chaque composant graphique est représenté par un élément XML, dont les attributs décrivent l'aspect ou le comportement.

# XML Layout: Design





# XML Layout: XML Code



# L'INTERFACE UTILISATEUR

La création d'une interface se traduit par la création de deux éléments :

- a. une définition de l'interface utilisateur (gabarits, etc.) de façon déclarative dans un fichier XML ;
- b. une définition de la logique utilisateur (comportement de l'interface) dans une classe d'activité.

# DÉFINITION D'UNE INTERFACE EN XML

Les fichiers de définition d'interface XML sont enregistrés dans le dossier */res/layout* du projet

- a. le nom du fichier ne doit comporter que des lettres minuscules et des chiffres (comme l'impose Java).
- b. Chaque fichier de définition d'interface doit se trouver dans le répertoire *res/layout* du projet
- c. possède un identifiant unique généré automatiquement par l'environnement de développement.
- d. Il est possible d'y faire référence directement dans votre code
- e. Par exemple, si le fichier se nomme **monlayout.xml**, vous pouvez y faire référence dans votre code grâce à la constante **R.layout.monlayout**

# DÉFINITION D'UNE INTERFACE DANS UNE ACTIVITÉ

- Dans une application, une interface est affichée par l'intermédiaire d'une activité.
- Il faut créer une activité en ajoutant une nouvelle classe à votre projet dérivant de la classe **Activity**.
- Le chargement du contenu de l'interface s'effectue à l'instanciation de l'activité.
- Il faut redéfinir la méthode **onCreate** de l'activité pour y spécifier la définition de l'interface à afficher via la méthode **setContentView**.

# DÉFINITION D'UNE INTERFACE DANS UNE ACTIVITÉ

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

# CRÉATION MANUELLE D'UNE INTERFACE

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.LinearLayout;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this) ;
        TextView text = new TextView(this) ;
        text.setText("Salut DSI 5.1") ;
        layout.addView(text) ;
        setContentView(layout) ;}}}
```

# LES LAYOUTS (GABARITS DE VUES)

- . C'est une extension de la classe ViewGroup
- . Il s'agit d'un conteneur qui aide à positionner les objets, qu'il s'agisse de vues ou d'autres gabarits
- . Il est possible d'imbriquer des gabarits les uns dans les autres, ce qui vous permettra de créer des mises en forme évoluées
- . Il est possible de décrire des interfaces utilisateur soit par une déclaration XML, soit directement dans le code d'une activité en utilisant les classes adéquates

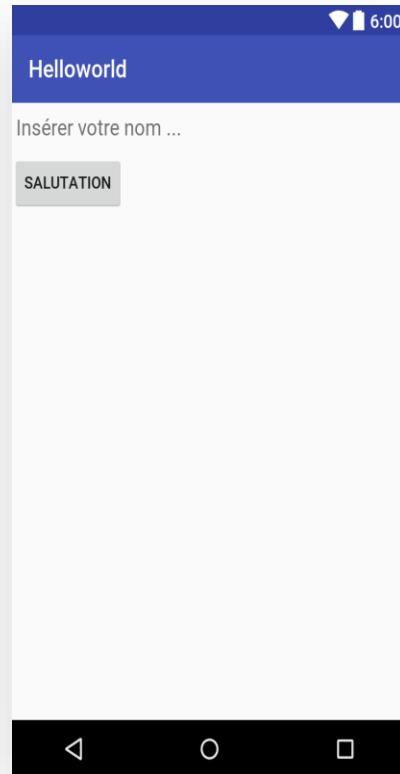
# Exemple de Layouts XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    tools:context="com.example.khouja.helloworld.MainActivity"
    tools:ignore="RtlHardcoded">

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/editText"
        android:hint="@string/ins_rer_votre_nom"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/salutation"
        android:id="@+id/button"/>

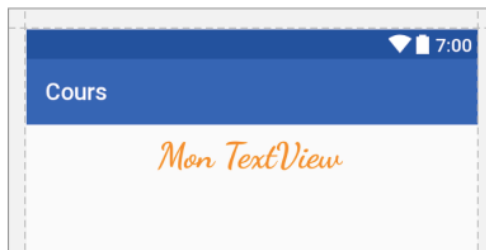
</LinearLayout>
```



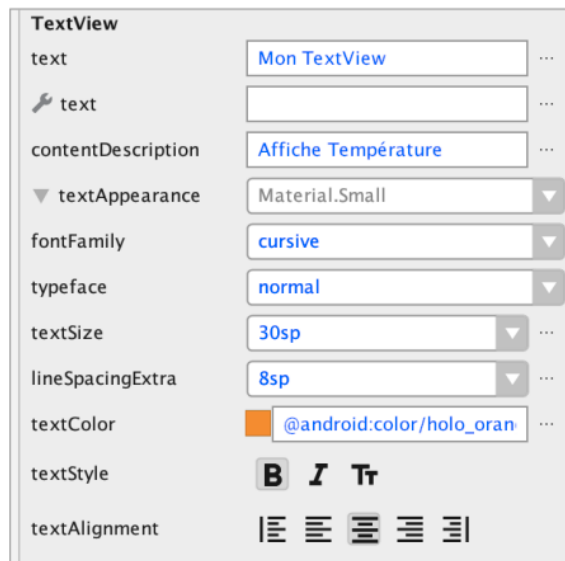


# QUELQUES COMPOSANTS ET PROPRIÉTÉS

## ○ TextView

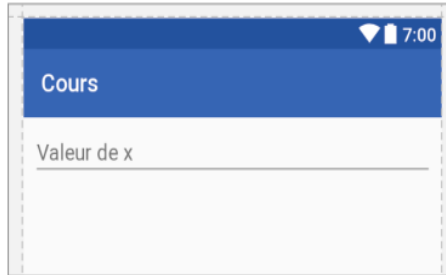


▼ autoLink	[web]
none	<input type="checkbox"/>
web	<input checked="" type="checkbox"/>
email	<input type="checkbox"/>
phone	<input type="checkbox"/>



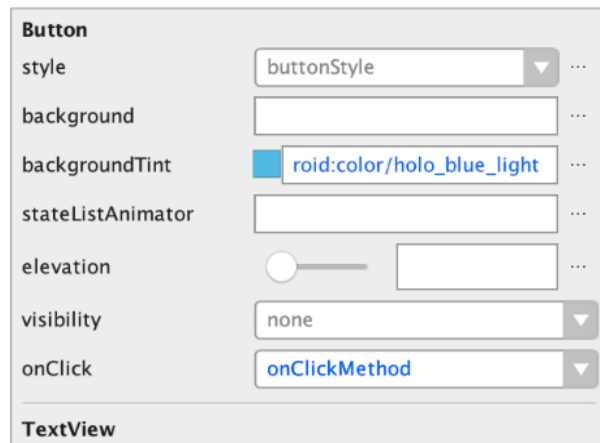
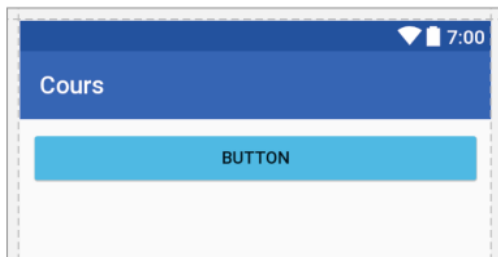
# QUELQUES COMPOSANTS ET PROPRIÉTÉS

## ○ EditText



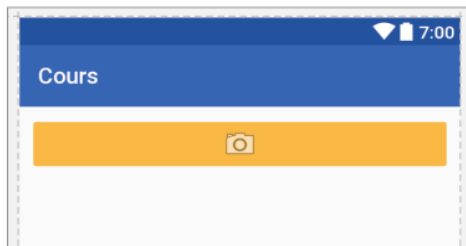
# QUELQUES COMPOSANTS ET PROPRIÉTÉS

## ○ Button



# QUELQUES COMPOSANTS ET PROPRIÉTÉS

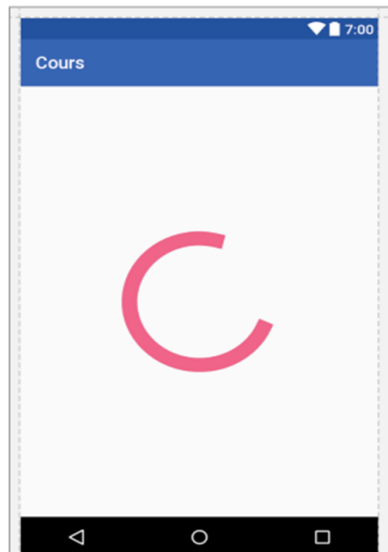
## ○ ImageButton



<b>ImageButton</b>	
srcCompat	<input type="text" value="&lt;code&gt;:drawable/ic_menu_camera&lt;/code&gt;"/>
contentDescription	<input type="text"/>
style	<input type="text" value="imageButtonStyle"/>
tint	<input type="text"/>
background	<input type="text"/>
backgroundTint	<input type="text" value="&lt;code&gt;d:color/holo_orange_light&lt;/code&gt;"/>
scaleType	<input type="text" value="center"/>
elevation	<input type="text" value=""/>
onClick	<input type="text" value="none"/>
adjustViewBounds	<input type="checkbox"/>
cropToPadding	<input type="checkbox"/>
visibility	<input type="text" value="none"/>

# QUELQUES COMPOSANTS ET PROPRIÉTÉS

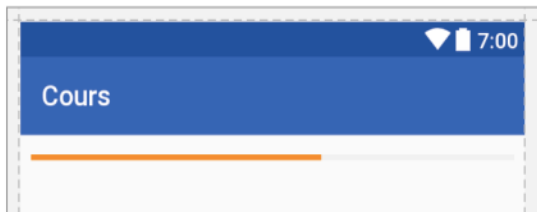
## ○ ProgressBar (circulaire)



ProgressBar	
style	<input type="text" value="..Material.Light.ProgressBar"/> ...
progressDrawable	<input type="text"/> ...
indeterminateDrawable	<input type="text"/> ...
progressTint	<input type="text"/> ...
indeterminateTint	<input type="text"/> ...
max	<input type="text"/> ...
progress	<input type="text"/> ...
visibility	<input type="text" value="none"/> ▼
⚙ visibility	<input type="text" value="none"/> ▼
indeterminate	<input checked="" type="checkbox"/>

# QUELQUES COMPOSANTS ET PROPRIÉTÉS

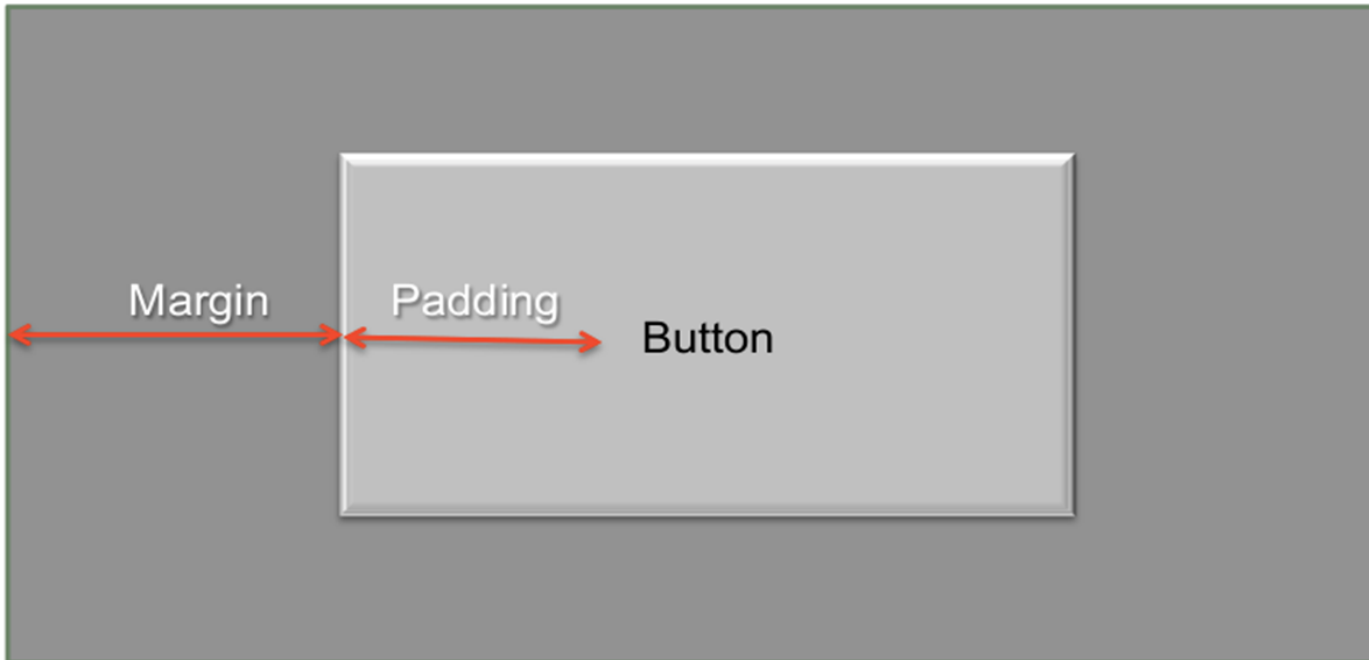
## ○ ProgressBar (Horizontal)



<b>ProgressBar</b>	
style	<input type="text" value="light.ProgressBar.Horizontal"/> ...
progressDrawable	<input type="text"/> ...
indeterminateDrawable	<input type="text"/> ...
progressTint	<input type="text" value="color/colo_holo_orange_dark"/> ...
indeterminateTint	<input type="text"/> ...
max	<input type="text" value="100"/> ...
progress	<input type="text" value="60"/> ...
visibility	<input type="text" value="none"/> ▼
visibility	<input type="text" value="none"/> ▼
indeterminate	<input type="checkbox"/>

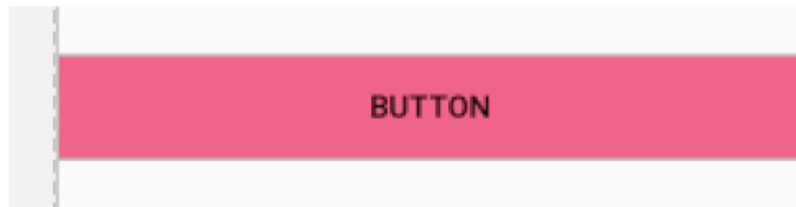
# PADDING & MARGIN

- Margin : Marges externes
- Padding : Marges internes

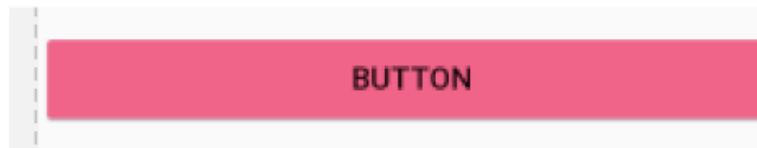


# BACKGROUND ET BACKGROUNTTINT

background



backgroundTint





# ViewGroup et View hiérarchie

# ViewGroup contient des vues « enfant »

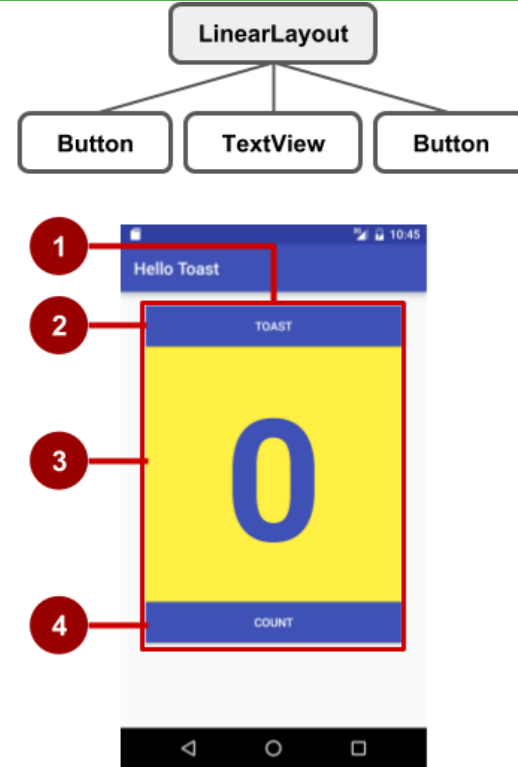
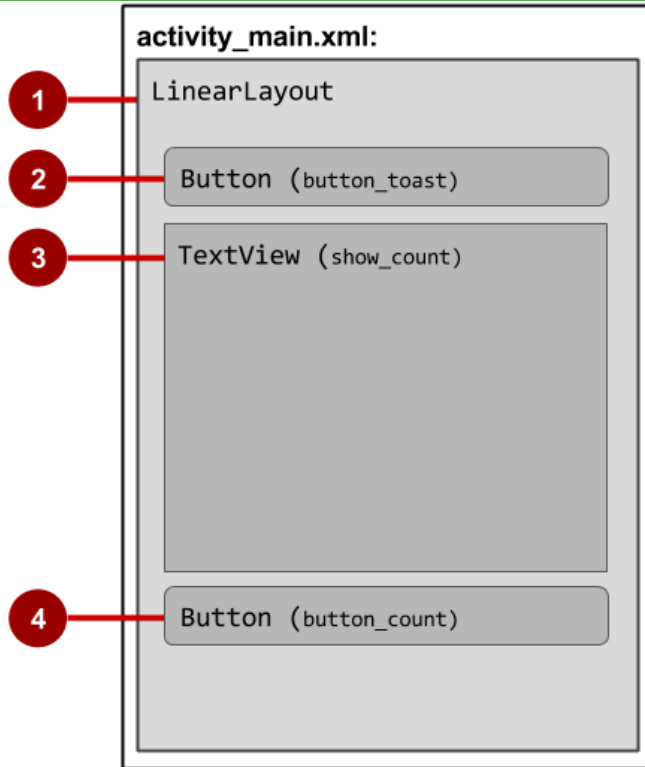
- [ConstraintLayout](#): positionne les éléments de l'interface utilisateur à l'aide de connexions de contrainte à d'autres éléments et aux bords d'un Layout
- [ScrollView](#) : contient un élément et permet le défilement
- [RecyclerView](#) : contient une liste d'éléments et permet le défilement en ajoutant et en supprimant des éléments de manière dynamique

# ViewGroups pour les Layouts

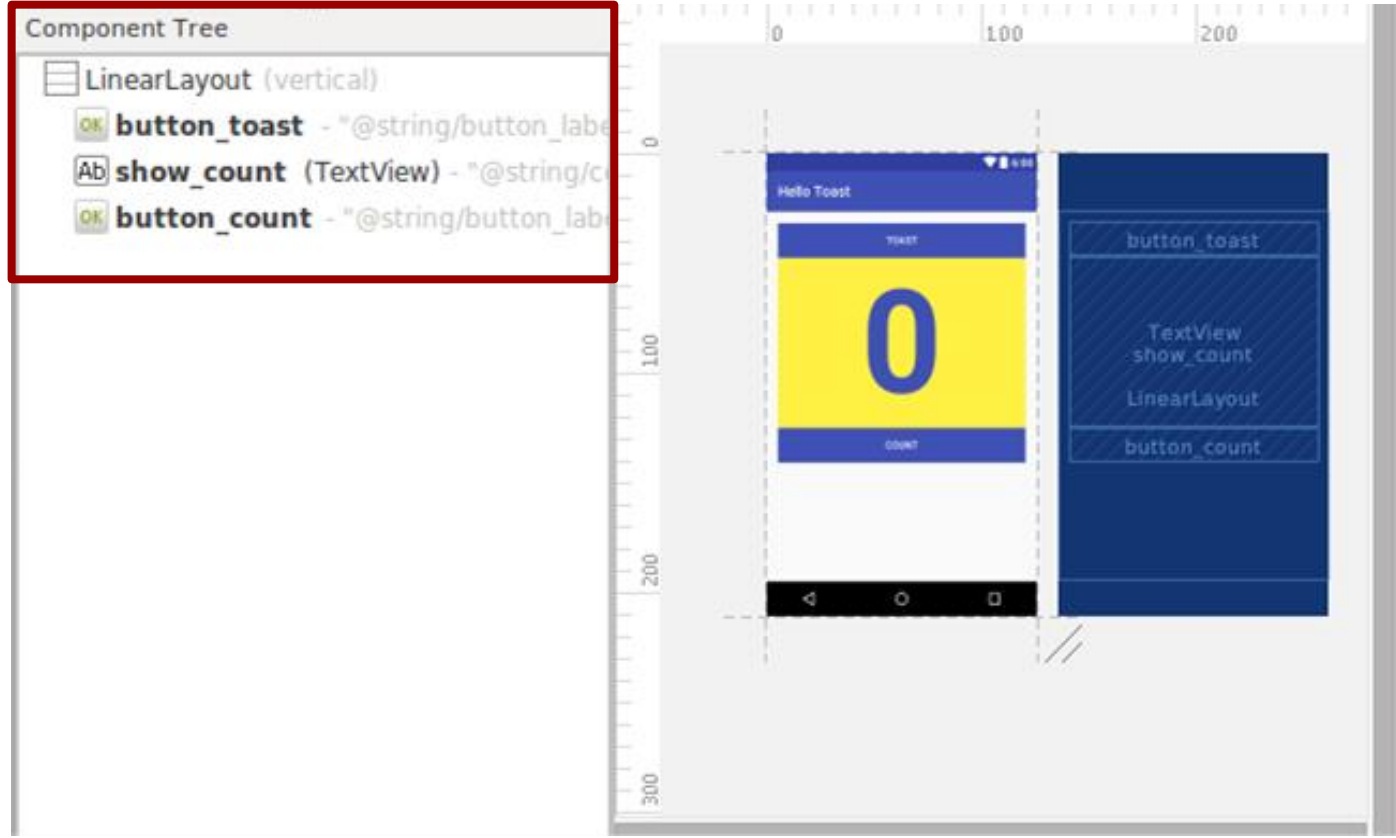
## Layouts

- sont des types spécifiques de ViewGroups (sous-classes de [ViewGroup](#))
- contient des vues enfants
- peut être dans une ligne, une colonne, une grille, un tableau, absolu

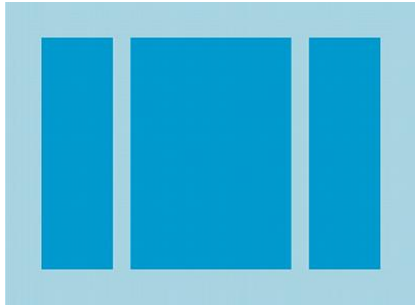
# La hiérarchie de vue et layout



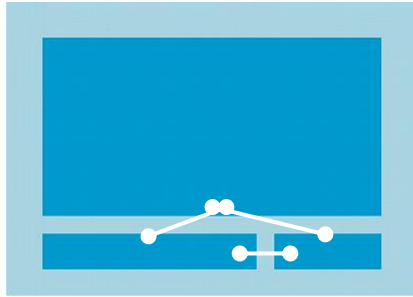
# La hiérarchie de vue dans l'éditeur de layout



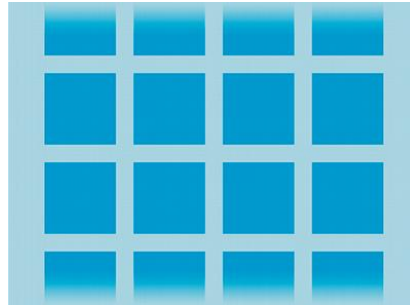
# Classes Layout courantes



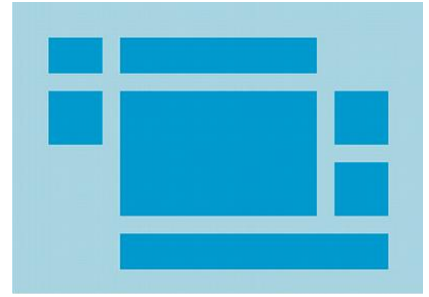
LinearLayout



ConstraintLayout



GridLayout



TableLayout

# LES DIFFÉRENTS TYPES DE LAYOUTS

- **LinearLayout** : permet d'aligner de gauche à droite ou de haut en bas les éléments qui y seront incorporés.
- **RelativeLayout** : ses enfants sont positionnés les uns par rapport aux autres, le premier enfant servant de référence aux autres.
- **FrameLayout** : c'est le plus basique des gabarits. Chaque enfant est positionné dans le coin en haut à gauche de l'écran et affiché par-dessus les enfants précédents, les cachant en partie ou complètement.
- **TableLayout** : permet de positionner vos vues en lignes et colonnes à l'instar d'un tableau.
- **ConstraintLayout** : permet de positionner vos vues en fonction de contraintes sur les positions fixées par l'utilisateur.

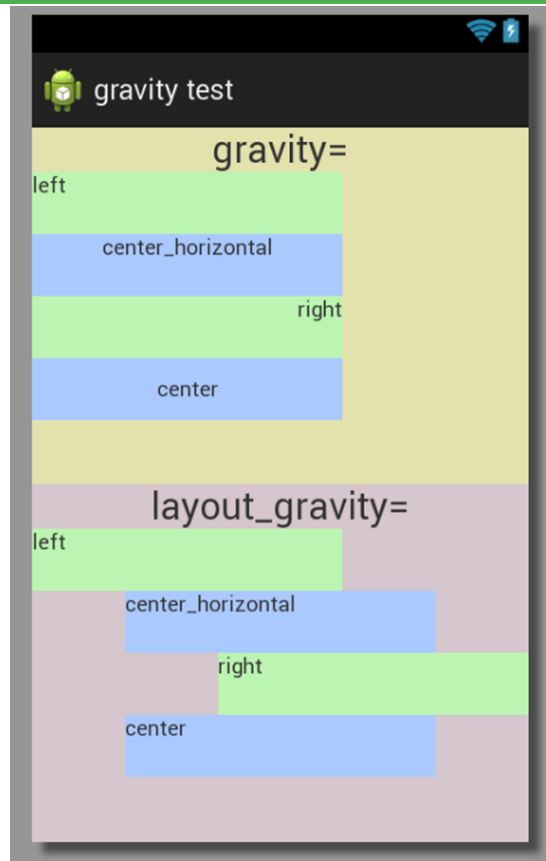
# LAYOUTS : PROPRIÉTÉS COMMUNES

- `layout_width` et `layout_height` :
  - a. Celles-ci permettent de spécifier le comportement du remplissage en largeur et en hauteur
  - b. ils peuvent contenir une taille (en pixels ou dpi)
  - c. ou les valeurs constantes suivantes :
    - i. **`match_parent`** : spécifie que le gabarit doit avoir la même dimension (largeur ou hauteur) de l'objet qui le contient
    - ii. **`wrap_content`** : affiche le gabarit tel quel et s'adapte à son contenu
    - iii. **Fixe (en dp)** : par exemple pour 200dp, la dimension sera exactement de 200 pixels
    - iv. **`match_constraint`** (0dp) : spécifie que le gabarit doit prendre toute la place disponible sur la largeur/hauteur tout en tenant compte des contraintes appliquées.



# LinearLayout

- Agencement des enfants dans une direction unique
- Attribut d'orientation :  
**LinearLayout.HORIZONTAL** et  
**LinearLayout.VERTICAL**
- Paramètres d'agencement :
  - **gravity**
  - **weight** pour la priorité pour l'agrandissement ou la réduction d'un composant (distribution de la différence d'espace proportionnelle au poids)



# Layout en XML

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        ... />
    <TextView
        ... />
    <Button
        ... />
</LinearLayout>
```

# Layout créé dans Java Activity

```
LinearLayout linearL = new LinearLayout(this);  
linearL.setOrientation(LinearLayout.VERTICAL);  
  
TextView myText = new TextView(this);  
myText.setText("Afficher ce texte!");  
  
linearL.addView(myText);  
setContentView(linearL);
```

# Définir la largeur et la hauteur dans le code Java

```
LinearLayout.LayoutParams layoutParams =  
    new LinearLayout.LayoutParams(  
        layoutParams.MATCH_PARENT,  
        layoutParams.WRAP_CONTENT);  
myView.setLayoutParams(layoutParams);
```

# Bonnes pratiques pour les hiérarchies de vues

- La disposition de la hiérarchie des vues affecte les performances de l'application
- Utiliser le plus petit nombre de vues les plus simples possible
- Maintenez la hiérarchie à plat : limitez l'imbrication des vues et des groupes de vues

# RelativeLayout

- Permet l'agencement de composants avec la spécification de contraintes relatives aux composant parent et aux composants frères (on place tel composant à gauche d'un autre, en occupant toute la hauteur du composant parent...)
- Très pratique pour la réalisation de formulaires (placement de champs d'édition, de boutons...)

# RelativeLayout

- Paramètres de positionnement relatif :
  - a. Verticalement : **below** (en bas d'un composant), **above** (au-dessus d'un composant)
  - b. Horizontalement :
    - i. **toLeftOf** (à gauche de),
    - ii. **toRightOf** (à droite de)
    - iii. **toStartOf** (avant),
    - iv. **toEndOf** (après)
- Paramètres d'alignement par rapport à un composant frère :
  - a. Verticalement : **alignTop**, **alignBottom**, **alignBaseline**
  - b. Horizontalement : **alignLeft**, **alignRight**, **alignStart**, **alignEnd**

# RelativeLayout

- Paramètres d'alignement par rapport au parent
  - a. Verticalement : `alignParentTop`, `alignParentBottom`
  - b. Horizontalement : `alignParentLeft`, `alignParentRight`, `alignParentStart`, `alignParentEnd`
  - c. Centrage : `centerHorizontal`, `centerVertical`, `centerInParent`



# TableLayout

- Contient des lignes de type TableRow (qui sont des sortes de LinearLayout)
- Similaire aux système de table en HTML (table, row, td)
- Paramètres d'agencement de TableRow :
  - a. **column** pour indiquer l'indice de départ de colonne
  - b. **span** pour indiquer le nombre de colonnes occupées

# GridLayout

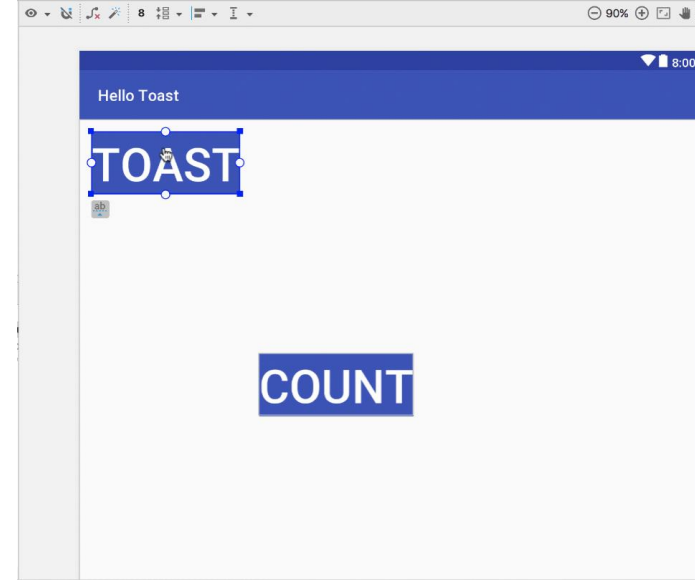
- Agencement sur une grille rectangulaire de N colonnes contrairement au TableLayout, les composants sont ajoutés directement avec leur paramètres d'agencement
- Propriétés de **GridLayout.LayoutParams** :
  - a. **column** et **columnSpan** : colonne de départ et nombre de colonnes occupées
  - b. **row** et **rowSpan** : ligne de départ et nombre de lignes occupées
  - c. **gravity** : emplacement de la vue enfant dans la cellule

# ConstraintLayout

- Layout permettant de définir des contraintes de placement : comparable au RelativeLayout mais avec plus de possibilités
- A pour vocation d'être un layout universel répondant à la plupart des besoins : usage par défaut lors de la création d'une activité Android avec Android Studio
- Layout en développement actif : il est conseillé d'utiliser une bibliothèque de rétrocompatibilité pour utiliser des fonctionnalités récentes sur des versions anciennes de l'API

# L'éditeur de Layout avec ConstraintLayout

- Connecter les éléments de l'interface utilisateur au Layout parent
- Redimensionner et positionner les éléments
- Aligner les éléments sur les autres
- Ajuster les marges et les dimensions
- Modifier les attributs



# ConstraintLayout

- Positionnement relatif possible par rapport à un autre composant (en indiquant son id) ou par rapport au parent (désigné par *parent*).
  - Sur l'axe horizontal pour aligner un côté d'un composant avec le côté d'un autre composant : `layout_constraint{Left, Right, Start, End}_to{Left, Right, Start, End}Of`
  - Sur l'axe vertical par rapport au bas, haut ou ligne de base (alignement de la base des lettres) des composants : `layout_constraint{Top, Bottom, Baseline}_to{Top, Bottom, Baseline}Of`
  - Sur un repère circulaire pour placer le centre d'un composant par rapport au centre d'un autre composant en indiquant le composant de référence au centre du cercle (`layout_constraintCircle`), le rayon du cercle (`layout_constraintCircleRadius`) et l'angle de placement (`layout_constraintCircleAngle`)

# ConstraintLayout

- Possibilité de définir des marges (comme pour la plupart des layouts) :
  - a. `layout_margin{Left, Right, Start, End, Top, Bottom}` par rapport à un composant visible

# UTILISATION DES COMPOSANTS GRAPHIQUES

# Identifiant et Fichier R.java

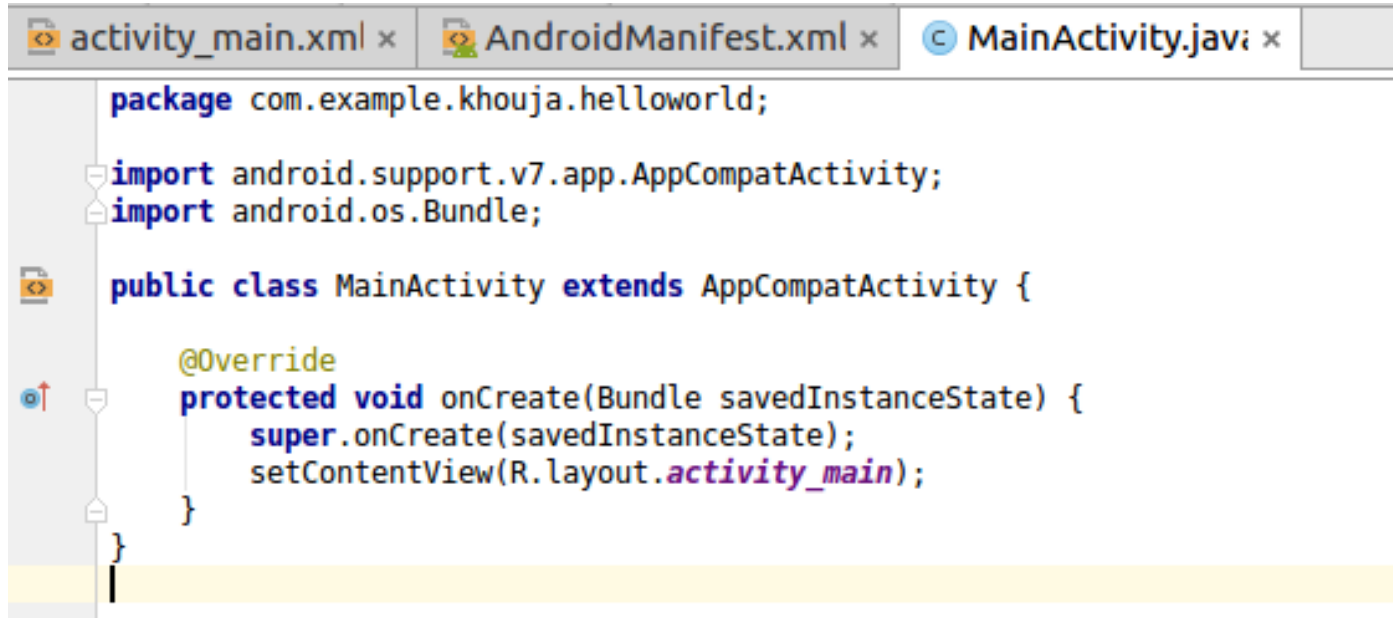
- . Pour pouvoir utiliser un composant graphique, il faut lui associer un **id**
  - a. Représenté dans le layout XML par l'attribut `android:id`
  - b. En général sous la forme: **@+id/nom\_id**
    - i. **@** : variable particulière, doit être parsée par le parseur XML
    - ii. **+** : un identifiant avec la valeur `nom_id` sera généré, et ajouté automatiquement à la classe R
- . Fichier R.java
  - a. Contient un ensemble de classes générées automatiquement à partir du répertoire `res` .
  - b. Chaque classe représente un ensemble de propriétés utilisables dans le code Java
  - c. Fichier généré, ne doit donc pas être modifié à la main!



# Code des Activités

- Les fichiers sources représentant le comportement de l'application sont définis dans le répertoire src, dans une arborescence de packages
- Chaque activité doit :
  - a. Être définie dans un fichiers .java à part
  - b. Hériter (directement ou indirectement) de la classe `android.app.Activity` ou (`AppCompatActivity`)
  - c. Implémenter la méthode `onCreate()` pour définir le comportement de l'activité à sa création
  - d. Appeler la méthode `setContentView()` dans le code de `onCreate()` pour définir le layout de cette activité
- L'implémentation des autres méthodes de transition ( `onPause`, `onDestroy`...) est optionnelle

# MainActivity.java

A screenshot of an IDE window showing the MainActivity.java file. The window has three tabs: 'activity\_main.xml', 'AndroidManifest.xml', and 'MainActivity.java'. The code is written in Java and includes package declarations, imports for AppCompatActivity and Bundle, and a MainActivity class that extends AppCompatActivity. The onCreate method is overridden to call super.onCreate and setContentView.

```
package com.example.khouja.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

# Comportement d'un Élément Graphique

- . Considérons l'objet de type *Button*
- . Objectif:

Afficher Hello < nom > en cliquant sur le bouton ( <nom> étant la chaîne saisie dans le EditText)

- . Il existe trois manières pour définir le comportement du bouton *button*

# Façon 1:

- Définir un objet Java de type `android.widget.Button`

```
public Button monButton ;
```

- Initialiser cet objet en l'associant à son équivalent dans le fichier XML

```
monButton = (Button) findViewById(R.id.button);
```

- Définir le comportement de ce bouton

```
monButton.setOnClickListener(new View.OnClickListener(){  
    @Override  
    public void onClick(View v) {  
        ...  
    }  
})
```

# Façon 1:

activity\_main.xml x AndroidManifest.xml x MainActivity.java x

```
package com.example.khouja.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    public Button monButton;
    public EditText monEditText;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        monButton = (Button) findViewById(R.id.button);
        monEditText = (EditText) findViewById(R.id.editText);
        monButton.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                Toast.makeText(MainActivity.this, "salut " + monEditText.getText() + "!", Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

# Façon 2:

- Définir un objet Java de type **android.widget.Button**
- Initialiser cet objet en l'associant à son équivalent dans le fichier XML
- Implémenter l'interface **android.view.View.OnClickListener**

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener
```

- Surcharger la méthode **onClick**

```
@Override
```

```
    public void onClick(View v) {  
        if (v.getId()==R.id.button) }
```

**Attention** :cette méthode sera commune à tous les éléments cliquables, il faut donc distinguer le comportement selon l'identifiant de l'élément cliqué

- Définir la classe en cours comme étant le Listener du bouton monButton

```
monButton.setOnClickListener(this);
```

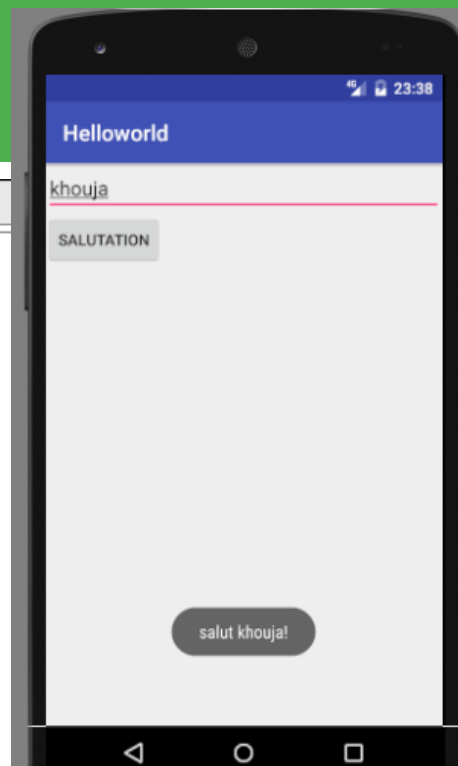
# Façon 2:

```
activity_main.xml x  AndroidManifest.xml x  MainActivity.java x
package com.example.khouja.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity implements View.OnClickListener{
    public Button monButton;
    public EditText monEditText;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        monButton = (Button) findViewById(R.id.button);
        monEditText = (EditText) findViewById(R.id.editText);
        monButton.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        if (v.getId()==R.id.button)
            Toast.makeText(this,"salut " + monEditText.getText() + "!", Toast.LENGTH_LONG).show();
    }
}
```



## Façon 3:

- Dans le fichier layout XML, associer au bouton le nom d'une méthode qui définira son comportement au clic

`android:onClick="salut"`

- Dans la classe Java de l'activité, créer et implémenter cette méthode en respectant les contraintes suivantes:
  1. Elle doit retourner void
  2. Elle doit porter le nom défini dans le fichier layout XML
  3. Elle doit définir un paramètre de type **android.view.View**

```
public void salut(View v) {  
    if (v.getId()==R.id.button)  
        ...  
}
```



# Façon 3:

activity\_main.xml x AndroidManifest.xml x MainActivity.java

```
package com.example.khouja.helloworld;
```

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    public EditText monEditText;
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
```

```
        monEditText = (EditText) findViewById(R.id.editText);
```

```
        public void salut(View v) {
```

```
            if (v.getId() == R.id.button)
```

```
                Toast.makeText(this, "salut " + monEditText.getText() + "!", Toast.LENGTH_LONG).show();
```

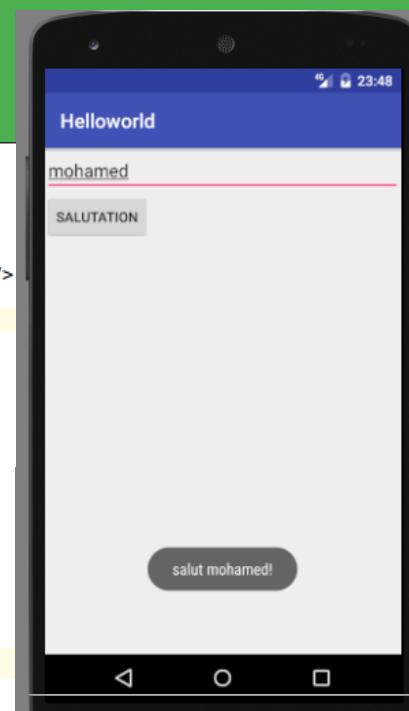
```
        }
```

```
    }
```

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

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/salutation"
    android:id="@+id/button"
    android:onClick="salut"/>

</LinearLayout>
```



# Ressources et mesures

# Resources

- Séparez les données statiques du code dans vos Layouts.
- Chaînes, dimensions, images, texte de menu, couleurs, styles
- Utile pour la localisation

# Trouver les ressources dans le code

- Layout:

```
R.layout.activity_main  
setContentView(R.layout.activity_main);
```

- View:

```
R.id.recyclerview  
rv = (RecyclerView) findViewById(R.id.recyclerview);
```

- String:

```
En Java: R.string.title  
En XML: android:text="@string/title"
```

# Measures

- Density-independent Pixels (dp): Pour les vues
- Scale-independent Pixels (sp): pour le texte

N'utilisez pas d'unités dépendantes de l'appareil ou de la densité :

- ~~Pixels (px)~~
- ~~Mesure actuelle (in, mm)~~
- ~~Points - typography (pt)~~

# Text et scrolling views

# TextView

- **TextView** est la sous-classe View pour le texte à une ou plusieurs lignes
- **EditText** est une sous-classe TextView avec du texte modifiable
- Contrôlé avec des attributs de layout
- Définir le texte:
  - Statiquement à partir d'une ressource de chaîne en XML
  - Dynamiquement à partir de code Java et de n'importe quelle source

# Attributs TextView commun

android:text — texte à afficher

android:textColor — couleur du texte

android:textAppearance — style ou thème prédéfini

android:textSize — taille du texte en sp

android:textStyle — normal, bold, italic, ou bold | italic

android:typeface — normal, sans, serif, ou monospace

android:lineSpacingExtra — espacement supplémentaire entre les lignes en sp



# Formattage des liens html

```
<string name="article_text">...www.iset.com...</string>
```

```
<TextView  
    android:id="@+id/article"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:autoLink="web"  
    android:text="@string/article_text"/>
```

[autoLink](#) valeurs: "web", "email", "phone", "map", "all"

# Qu'en est-il des grandes quantités de texte ?

- Nouvelles, articles, etc...
- Pour faire défiler un TextView, incorporez-le dans un ScrollView
- Un seul élément View (généralement TextView) autorisé dans un ScrollView
- Pour faire défiler plusieurs éléments, utilisez un ViewGroup (tel que LinearLayout) dans le ScrollView

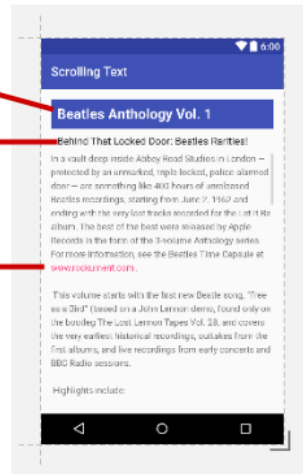
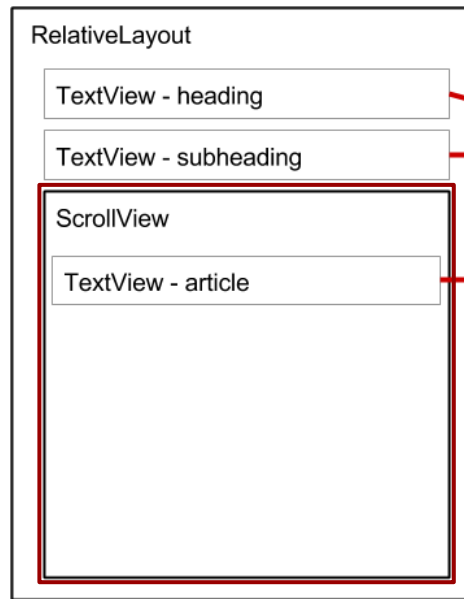
# ScrollView pour faire défiler le contenu

- **ScrollView** est une sous-classe de FrameLayout
- Contient tout le contenu en mémoire
- Pas bon pour les textes longs, les layouts complexes
- Ne pas imbriquer plusieurs vues de défilement
- Utiliser HorizontalScrollView pour le défilement horizontal
- Utiliser un RecyclerView pour les listes

# ScrollView layout avec un TextView

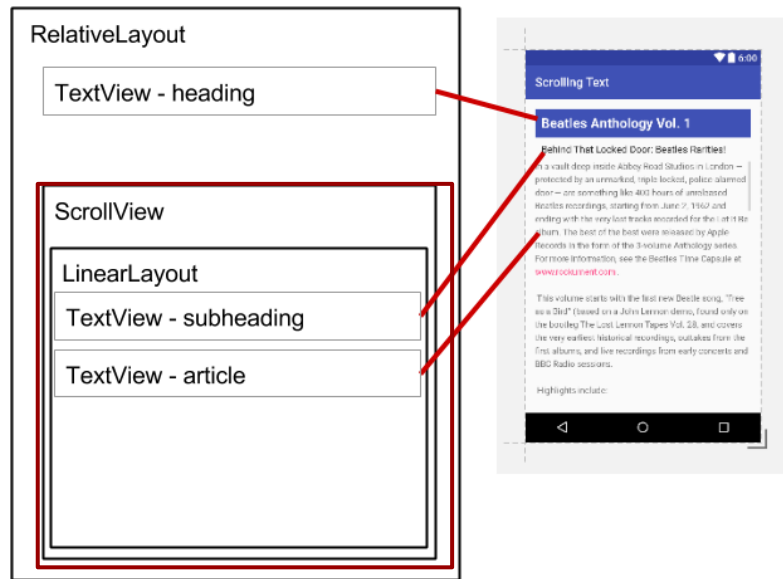
```
<ScrollView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

    android:layout_below="@id/article_subheading">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        .../>
</ScrollView>
```



# ScrollView layout avec un view group

```
<ScrollView ...  
    <LinearLayout  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:orientation="vertical">  
  
        <TextView  
            android:id="@+id/article_subheading"  
            .../>  
  
        <TextView  
            android:id="@+id/article" ... />  
        </LinearLayout>  
    </ScrollView>
```



# ScrollView avec image et bouton

```
<ScrollView...>
```

```
  <LinearLayout...>
```

```
    <ImageView.../>
```

```
    <Button.../>
```

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

```
  </LinearLayout>
```

```
</ScrollView>
```

← Un file du ScrollView  
est le layout

← Fils du layout

# Boutons et images cliquables

# Intéraction avec l'utilisateur



# Les utilisateurs s'attendent à interagir avec les applications

- Taper ou cliquer, tapoter, utiliser des gestes et parler
- Les boutons effectuent des actions
- D'autres éléments de l'interface utilisateur permettent la saisie de données et la navigation

# Conception d'interaction avec l'utilisateur

Important d'être évident, facile et cohérent :

- Réfléchissez à la manière dont les utilisateurs utiliseront votre application
- Réduire les étapes
- Utiliser des éléments d'interface utilisateur faciles d'accès, de compréhension et d'utilisation
- Suivez les bonnes pratiques Android
- Répondre aux attentes des utilisateurs

# Boutons

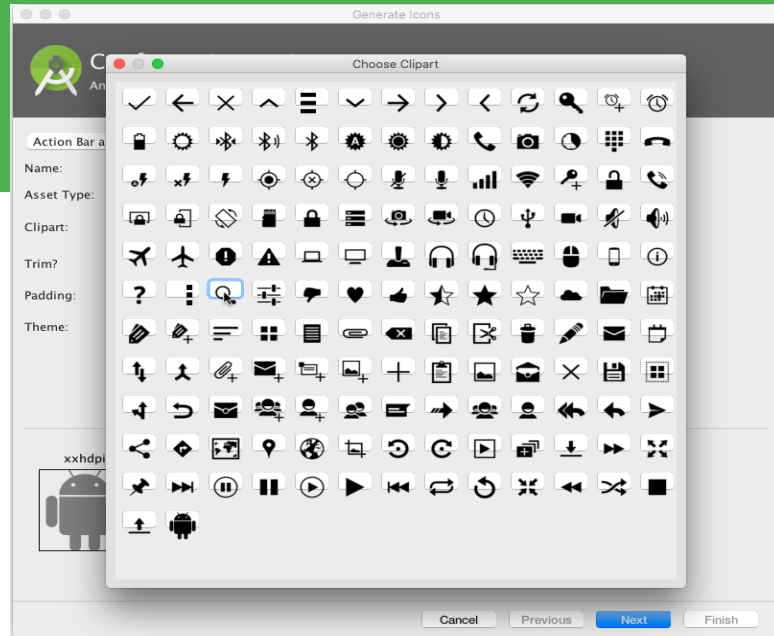
# Bouton

- Vue qui répond au tapotement ou à la pression
- Habituellement, le texte ou les visuels indiquent ce qui se passera lorsque vous appuyez sur
- État : normal, a le focus, désactivé, enfoncé, on/off



# Bouton : image assets

1. Cliquer droit app/res/drawable
2. Choisir **New > Image Asset**
3. Choisir **Action Bar and Tab Items**
4. Cliquer sur **Clipart**: image (logo Android)



alternative:

2. Choisir **New > Vector Asset**

# Réglage de l'écouteur avec le rappel onClick

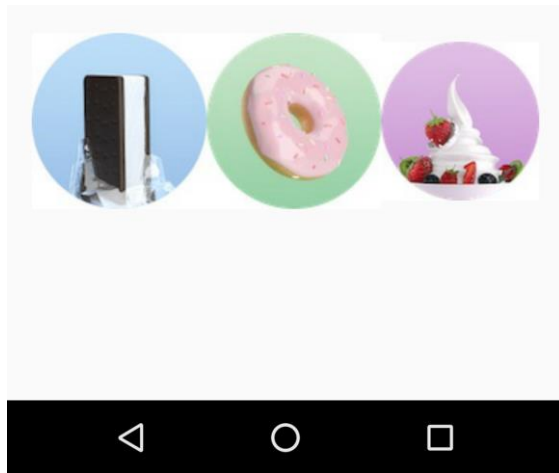
```
Button button = findViewById(R.id.button);

button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

# Images cliquables

# ImageView

- ImageView avec l'attribut `android:onClick`
- Image pour ImageView dans **`app>src>main>res>drawable`**





# Répondre aux cliques d'ImageView

- *Dans votre code*: utilise `OnClickListener`.
- *Dans XML*: utilise l'attribut `android:onClick` dans XML

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/donut_circle"  
    android:onClick="orderDonut"/>
```

android:onClick

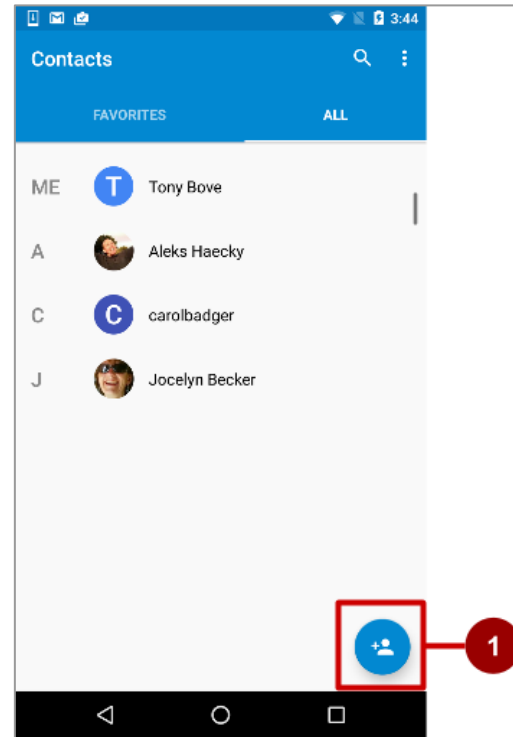
A diagram consisting of a rounded rectangular box containing the text 'android:onClick'. A vertical line extends downwards from the bottom center of the box, and a horizontal line extends to the left from the bottom of the vertical line, ending with an arrowhead pointing to the 'android:onClick' attribute in the XML code block to the left.

# Floating action button

# Floating Action Buttons (FAB)

- Surélevé, circulaire, flotte au-dessus du layout
- Action principale pour un écran
- Un par écran

Par exemple: Bouton Ajouter un contact dans l'application Contacts



# Utiliser les FABs

- Par défaut dans le template de Basic Activity
- Layout:

```
<android.support.design.widget.FloatingActionButton  
    android:id="@+id/fab"  
    android:layout_gravity="bottom|end"  
    android:layout_margin="@dimen/fab_margin"  
    android:src="@drawable/ic_fab_chat_button_white"  
.../>
```

# Taille FAB

- 56 x 56 dp par défaut
- Taille minimale (30 x 40 dp) avec l'attribut `app:fabSize` :
  - `app:fabSize="mini"`
- Mettre à 56 x 56 dp (défaut):
  - `app:fabSize="normal"`

# Gestes communs

# Gestes tactiles

Les gestes tactiles incluent :

- touche longue
- tapez deux fois
- balancer
- glisser
- faire défiler
- pincer

# Détecter les gestes

Des classes et des méthodes sont disponibles pour vous aider à gérer les gestes.

- Classe **GestureDetectorCompat** pour les gestes courants
- Classe **MotionEvent** pour les événements de mouvement



# Détecter tous types de gestes

1. Recueillez des données sur les événements tactiles.
2. Interprétez les données pour voir si elles répondent aux critères de l'un des gestes pris en charge par votre application.

En savoir plus sur la façon de gérer les gestes dans [Android developer documentation](#)

# ADAPTATEURS

# Layout Dynamique

- Si le contenu du Layout est dynamique, ou non pré-déterminé, utiliser un layout qui hérite de la classe *AdapterView*
  - Permet de remplir le layout avec des vues à l'exécution
- Utilisation d'un Adapter dans cette sous-classe pour relier les données au layout
  - Permet d'extraire les données d'une source (tableau ou requête de base de données)
  - Convertit chaque entrée en une vue
  - Ajoute la vue au layout
- Il existe plusieurs types d'adaptateurs..

# ArrayAdapter

- Utiliser quand la source de données retourne un tableau
- D'abord créer l'adaptateur: prend trois paramètres :
  - Contexte: instance de l'activité où il évolue
  - Layout avec un *textView* pour chaque chaîne du tableau: par ex: ressource prédéfinie: **android.R.layout.simple\_list\_item\_1**
  - Tableau ou liste : éléments à insérer

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,  
android.R.layout.simple_list_item_1, mesDonnees);
```

- Associer ensuite l'adaptateur à la liste (élément graphique de type ListView)

```
maliste=(ListView) findViewById(R.id.maliste);  
maliste.setAdapter(adapter);
```

# ArrayAdapter

```
activity_main.xml x  AndroidManifest.xml x  MainActivity.java x

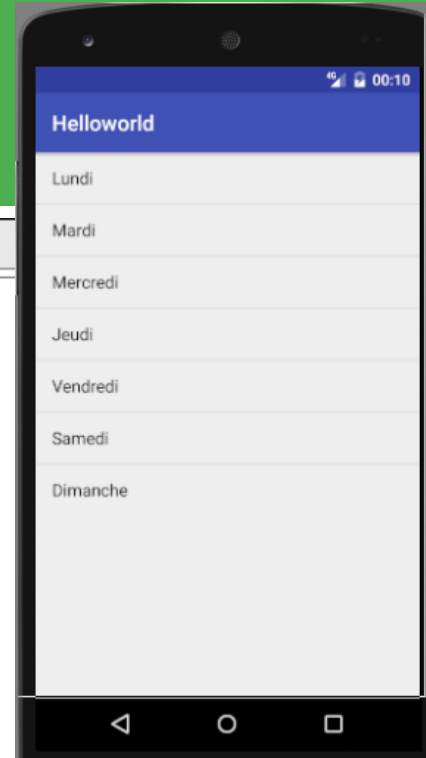
package com.example.khouja.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends AppCompatActivity {

    ListView maliste;
    String[] mesDonnees = {"Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        maliste=(ListView) findViewById(R.id.maliste);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,mesDonnees);
        maliste.setAdapter(adapter);
    }
}
```



# ArrayAdapter

- Associer un comportement au clic sur un élément de la liste:
  - Créer une classe anonyme dans laquelle définir le comportement au clic de l'élément de la liste
  - Définir cette classe comme **ItemClickListener** de votre liste

```
private OnItemClickListener mMessageClickedHandler= new  
OnItemClickListener(){  
    public void onItemClick(AdapterView parent, View  
v, int position, long id){  
        ...  
    }  
}
```

# ArrayAdapter

```

activity_main.xml x  AndroidManifest.xml x  MainActivity.java x
package com.example.khouja.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.*;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

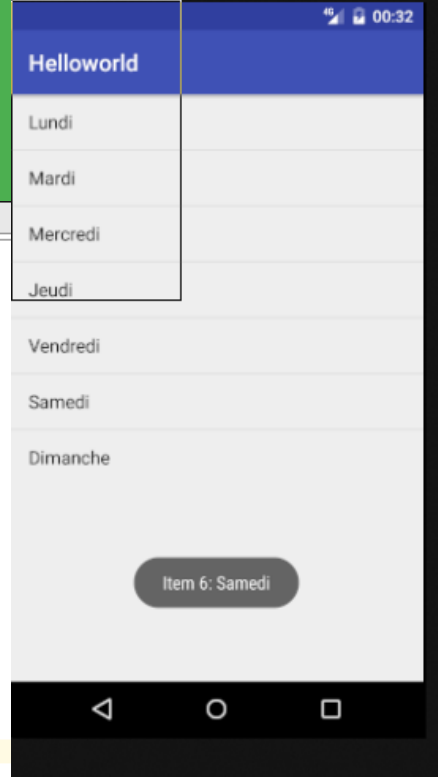
public class MainActivity extends AppCompatActivity {

    ListView maliste;
    String[] mesDonnees = {"Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        maliste = (ListView) findViewById(R.id.maliste);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, mesDonnees);
        maliste.setAdapter(adapter);
        maliste.setOnItemClickListener(mMessageClickedHandler);
    }

    private OnItemClickListener mMessageClickedHandler= new OnItemClickListener(){
        public void onItemClick(AdapterView parent, View v, int position, long id){
            Toast.makeText(MainActivity.this, "Item "+(position +1)+": "+((TextView)v).getText(), Toast.LENGTH_LONG).show();
        }
    };
}

```



# Liste Déroulante

- Appelées aussi Spinner
- Offre la même fonctionnalité qu'une Liste, mais en prenant moins d'espace
- Remplacer l'élément prédéfini `android.R.layout.simple_list_item_1` par `android.R.layout.simple_spinner_item` par exemple.
- Associer un layout à utiliser quand la liste de choix apparaît avec `setDropDownViewResource`
- Pour spécifier le comportement à la sélection d'un élément dans la liste déroulante, implémenter les méthodes `onItemSelected` et `onNothingSelected` de la classe `OnItemSelectedListener`



# Liste Déroulante

```

activity_main.xml x  AndroidManifest.xml x  MainActivity.java x
package com.example.khouja.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.*;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

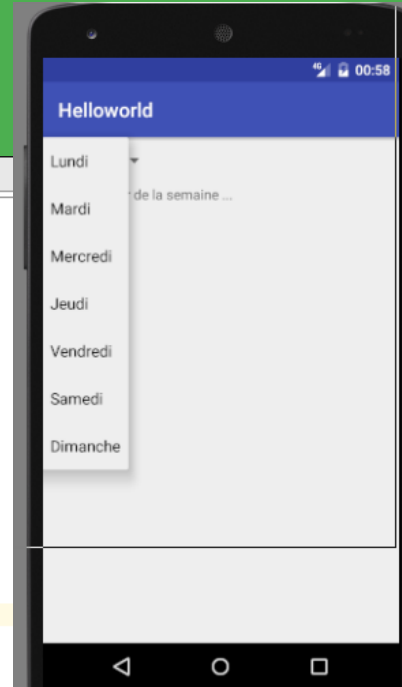
    String[] mesDonnees = {"Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche"};
    Spinner maSpinner;
    TextView choixText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        maSpinner = (Spinner) findViewById(R.id.spinner);
        choixText = (TextView) findViewById(R.id.choixText);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_dropdown_item, mesDonnees);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        maSpinner.setAdapter(adapter);
        maSpinner.setOnItemSelectedListener(mMessageSelectedHandler);

        private OnItemSelectedListener mMessageSelectedHandler= new OnItemSelectedListener(){
            public void onItemSelected(AdapterView parent, View v, int position, long id){
                Toast.makeText(MainActivity.this, "Item "+(position +1)+": "+((TextView)v).getText(), Toast.LENGTH_LONG).show();
            }
        }

        @Override
        public void onNothingSelected (AdapterView<?> parent){
            choixText.setText("Choisir un jour de la semaine ...");
        }
    }
}

```



# AutoCompleteTextView

- Utilise des données prédéfinies pour auto-compléter la chaîne entrée par l'utilisateur par des suggestions
- Les suggestions sont présentées dans une liste de sélections, comme pour le Spinner
- Sous classe de EditText:
  - On peut la paramétrer de la même manière
  - Contient un attribut supplémentaire : **android:completionThreshold** : indique le nombre minimum de caractères qu'un utilisateur doit entrer pour que les suggestions apparaissent

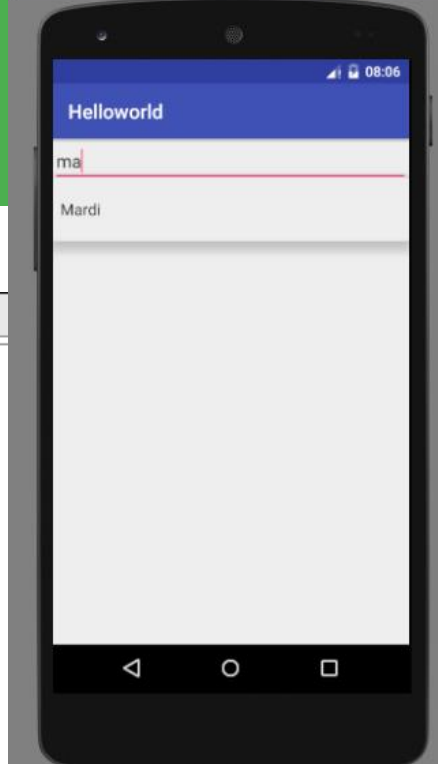
# AutoCompleteTextView

```
activity_main.xml x  AndroidManifest.xml x  MainActivity.java x
package com.example.khouja.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;

public class MainActivity extends AppCompatActivity {
    AutoCompleteTextView jour;
    String[] mesDonnees = {"Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        jour = (AutoCompleteTextView) findViewById(R.id.jour);
        jour.setAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_dropdown_item_1line, mesDonnees));
    }
}
```



# Grille

- GridView: élément graphique qui permet d'afficher les éléments dans une grille de défilement à deux dimensions
- Les éléments sont automatiquement insérés dans la grille grâce à un ListAdapter
- Déterminer les caractéristiques de la grille dans son élément XML :
  - **android:numColumns** : Nombre de colonnes
  - **android:horizontalSpacing** : Espacement horizontal
  - **android:verticalSpacing** : Espacement vertical
  - **android:columnWidth** : Nombre de pixels défini pour la largeur de la colonne
  - **android:stretch** : indique, pour les grilles avec un nombre de colonnes défini à **auto\_fit**, ce qui doit arriver pour l'espace disponible, non occupé par les colonnes ou les espaces.
    - S'il est égal à **columnWidth**, les colonnes seront étirées pour combler l'espace vide.
    - S'il est égal à **spacingWidth**, les espaces entre les colonnes seront étirés pour combler l'espace vide.

# Exemple de GridView

```
package com.example.khouja.helloworld;

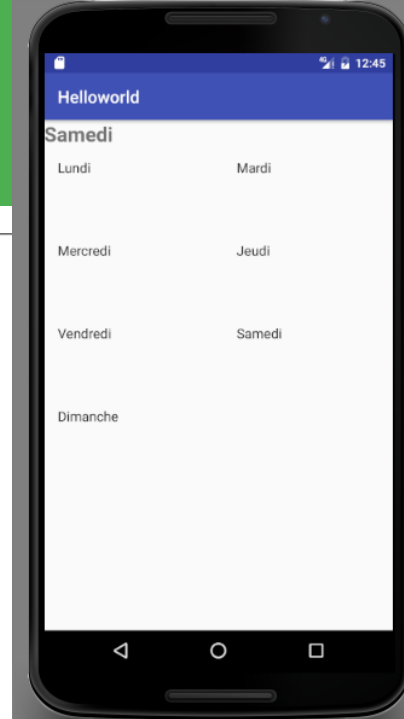
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.GridView;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    GridView grille;
    TextView selection;
    String[] mesDonnees = {"Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        selection = (TextView) findViewById(R.id.selection);
        grille = (GridView) findViewById(R.id.grille);
        ArrayAdapter<String> aa=new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, mesDonnees);
        grille.setAdapter(aa);
        grille.setOnItemClickListener(new AdapterView.OnItemClickListener(){
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                selection.setText(mesDonnees[position]);
            }
        });
    }
}
```

```
<TextView
    android:text="@string/textview"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/selection"
    android:textStyle="bold"
    android:textSize="24sp" />

<GridView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/grille"
    android:columnWidth="200dp"
    android:horizontalSpacing="5dp"
    android:verticalSpacing="50dp"
    android:numColumns="auto_fit"
    android:stretchMode="spacingWidth" />
</LinearLayout>
```



# Listes Personnalisées

- Permettent de gérer les situations où les listes correspondent aux situations suivantes:
  - Les lignes ne suivent pas la même disposition
  - Contiennent des éléments graphiques qui varient d'une case à l'autre
- Il faut:
  - Définir la ligne individuelle dans un layout XML à part
  - Créer vos propres adaptateurs, en surchargeant la méthode `getView` du `ArrayAdapter` pour construire vos propres lignes
    - `getView` retourne une ligne associée à la position fournie
  - Utiliser un `LayoutInflater`
    - Permet de convertir un layout en XML vers le vrai arbre de vues que cet XML représente.

# Exemple de Liste Personnalisée (layouts XML)

```

activity_main.xml x  ma_ligne.xml x  AndroidManifest.xml x  Mai
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    tools:context="com.example.khouja.helloworld.MainActivity"
    tools:ignore="RtlHardcoded">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/selection"
        android:textStyle="bold"
        android:textSize="20dp"
        android:gravity="center"
        android:hint="Sélectionner un item"/>

    <ListView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@android:id/list"/>

</LinearLayout>

```

```

activity_main.xml x  ma_ligne.xml x  AndroidManifest.xml x  MainA
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:layout_width="40dp"
        android:layout_height="42dp"
        app:srcCompat="@drawable/pass"
        android:id="@+id/imageView"
        android:nestedScrollingEnabled="false"
        android:longClickable="false"/>

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:id="@+id/label"
        android:gravity="center"/>

</LinearLayout>

```

# Exemple de Liste Personnalisée (Code Java et Rendu)

```

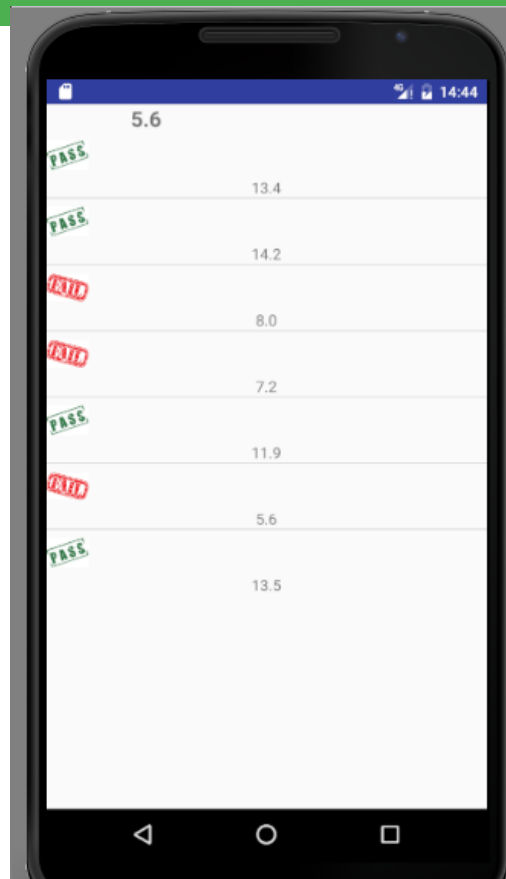
import android.widget.ListView;
import android.widget.TextView;

public class MainActivity extends ListActivity {
    TextView selection;
    String[] mesDonnees = {"13.4", "14.2", "8.0", "7.2", "11.9", "5.6", "13.5"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        selection = (TextView) findViewById(R.id.selection);
        setListAdapter(new IconicAdapter(this));
    }

    @Override
    protected void onListItemClick(ListView l, View view, int position, long id){
        selection.setText(mesDonnees[position]);
    }

    class IconicAdapter extends ArrayAdapter<String>{
        Activity context;
        IconicAdapter(Activity context){
            super(context, R.layout.ma_ligne, mesDonnees);
            this.context=context;
        }
        @Override
        public View getView(int position, View convertView, ViewGroup parent){
            LayoutInflater inflater=context.getLayoutInflater();
            View ligne=inflater.inflate(R.layout.ma_ligne,null);
            TextView label = (TextView) ligne.findViewById(R.id.label);
            ImageView icone = (ImageView) ligne.findViewById(R.id.imageView);
            label.setText(mesDonnees[position]);
            if(Float.valueOf(mesDonnees[position]) >=10){
                icone.setImageResource(R.drawable.pass);
            }
            else{
                icone.setImageResource(R.drawable.fail);
            }
            return ligne;
        }
    }
}

```



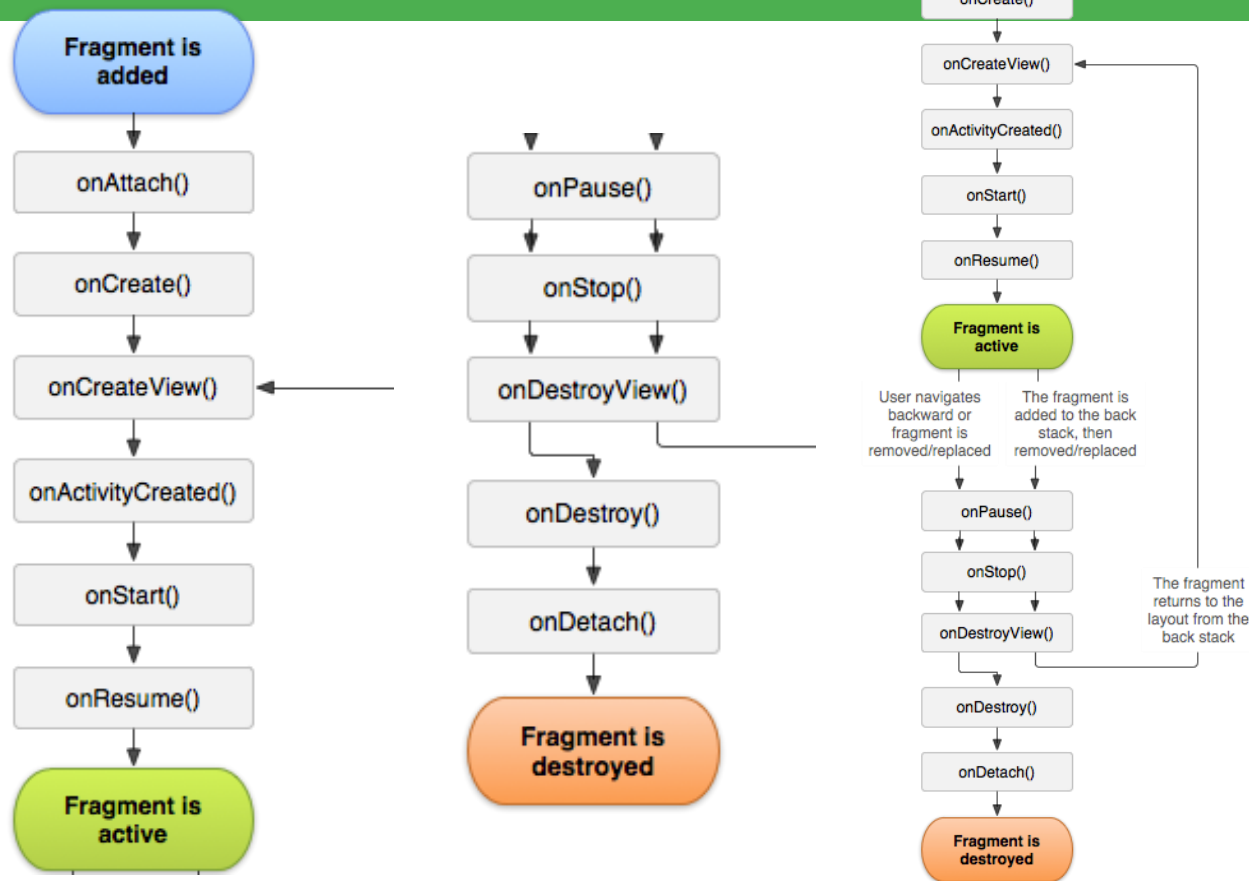


# FRAGMENTS

# FRAGMENTS

- . Un Fragment représente un comportement ou une portion d'interface utilisateur dans une activité
- . Plusieurs fragments peuvent être combinés dans une seule activité pour construire une interface à plusieurs volets ( *multi-pane* )
- . Un fragment peut être réutilisé dans plusieurs activités
- . Un fragment :
  - . a son propre cycle de vie, mais est directement affecté par celui de son hôte
  - . reçoit ses propres entrées
  - . peut être ajouté ou enlevé dynamiquement pendant que l'activité s'exécute
- . Vit à l'intérieur d'un ViewGroup dans la hiérarchie de son Activity
- . Peut être ajouté à l'activité:
  - . Dans le fichier Layout XML, comme `<fragment>`
  - . Dans le code de l'application en l'ajoutant à un conteneur (*ViewGroup*) existant

# Fragments: Cycle de Vie



# Fragments: Cycle de Vie

- Dans *onAttach*, on récupère un pointeur vers l'activité contenant, elle est invoquée quand le fragment est associé à une activity
- Dans *onCreate*, on instancie les objets non graphiques ;
- Dans *onCreateView*, on instancie la vue et les composants graphiques, elle est invoquée pour créer sa propre interface utilisateur pour la première fois
- Dans *onActivityCreated*, on récupère un pointeur sur l'activité (qui est construite), on lance les initialisations qui ont besoin de cette activité, on restaure le contexte des fragments et utilisateur, elle est invoquée quand la méthode onCreate() de l'activity se termine.
- Dans *onStart*, on lance les traitements ;
- Dans *onResume*, on s'abonne et on remet le contexte utilisateur ;
- Dans *onPause*, on se désabonne et on enregistre le contexte utilisateur ;
- Dans *onStop*, on arrête les traitements et on désalloue les objets ;
- Dans *onDestroyView*, la vue est détachée de celle de l'activité, on peut délivrer la mémoire des objets graphiques ;
- *onDestroy*
- *OnDetach* : invoquée quand le fragment est dissocié de l' Activity.

# Création d'un Fragment

• Pour créer un fragment, il faut créer une classe qui hérite de *android.app.Fragment* (ou une de ses sous-classes)

• Sous-classes utiles de Fragment:

– **DialogFragment**

• Affiche une boîte de dialogue flottante, au lieu d'utiliser des méthodes dans votre activité, car il peut être rajouté à la pile des fragments de l'activité

– **ListFragment**

• Affiche une liste d'éléments gérés par un adaptateur

• Fournit plusieurs méthodes pour gérer une ListView

– **PreferenceFragment**

• Affiche une hiérarchie d'objets *Preference* comme liste

• Permet de créer une activité « *settings* » pour votre application

# Création de l'Interface du Fragment

- Définir une interface pour le fragment dans un fichier Layout séparé
- Implémenter la méthode de transition `onCreateView` pour construire l'interface du fragment
  - Doit retourner un objet *View* représentant la racine du fragment
  - Utiliser un *LayoutInflater* pour construire l'interface à partir des ressources XML

```
public static class PlaceholderFragment extends Fragment {  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
                             Bundle savedInstanceState) {  
        View rootView = inflater.inflate(R.layout.fragment_, container, false);  
        return rootView;  
    }  
}
```

Conteneur de l'activité parente, dans lequel le fragment sera inséré

ID du Layout à parcourir

# Ajout d'un Fragment à une Activité

- Soit déclarer le fragment dans le fichier layout XML de l'activité grâce à l'élément XML `<fragment>`

-android:name      classe du fragment à  
instancier dans le layout



```
<fragment android:name="com.example.news.ArticleReaderFragment"
    android:id="@+id/viewer"
    android:layout_weight="2"
    android:layout_width="0dp"
    android:layout_height="match_parent" />
```

- Soit ajouter le fragment à un élément ViewGroup dans le code

- Créer une transaction de fragment (ajout, suppression...) grâce au FragmentManager

- Ajouter un fragment en utilisant la méthode `add(<conteneur>, <fragment>)`

```
public class FragmentActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                .add(R.id.container, new PlaceholderFragment())
                .commit();
        }
    }
}
```

# Ajout d'un Fragment sans UI

- Un fragment peut être utilisé pour fournir un comportement en arrière plan pour l'activité sans présenter d'interface supplémentaire
- Pour cela, ajouter le fragment à l'activité en utilisant *add(Fragment, <tag\_unique\_du\_frag> )*
- Il est inutile d'implémenter *onCreateView*



# FragmentManager

- Le *FragmentManager* permet de gérer les fragments de votre activité
- Appeler *getFragmentManager()* à partir de votre activité
- Il permet de:
  - Manipuler des fragments existants dans votre application grâce à *findFragmentById()* ou *findFragmentByTag()*
  - Gérer la pile accessible via le bouton Back de votre appareil:
    - Dépiler un fragment de la pile des fragments, avec *popBackStack()*
    - Associer un Listener aux changement dans la pile avec *addOnBackStackChangeListener()*

# Gérer les Transactions

- Possibilité d'ajouter, supprimer, remplacer... des fragments dans une activité, en réponse à une interaction utilisateur
- Chacun de ces changements est appelé Transaction
- Utilisation de *FragmentTransaction*
- Il est possible de sauvegarder chaque transaction dans la pile de retour ( Back Stack ) de l'activité, pour permettre à l'utilisateur de revenir en arrière

# Exemple Complet : FragmentOne

```
package android.ieee.fragment2;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class FragmentOne extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {

        return inflater.inflate(
            R.layout.fragment_one, container, false);
    }
}
```

FragmentOne.java

fragment\_one.xml

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#00ffff">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="This is fragment No.1"
        android:textStyle="bold" />

</LinearLayout>
```

# Exemple Complet : FragmentTwo

```
package android.ieee.fragment2;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class FragmentTwo extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {

        return inflater.inflate(
            R.layout.fragment_two, container, false);
    }
}
```

FragmentTwo.java

fragment\_two.xml

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#ffff00">

    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="This is fragment No.2"
        android:textStyle="bold" />

</LinearLayout>
```

# Exemple Complet: Activité

```

public class MyFragmentActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_my_fragment);
    }

    public void selectFrag(View view) {
        Fragment fr;

        if (view == findViewById(R.id.button2)) {
            fr = new FragmentTwo();
        } else {
            fr = new FragmentOne();
        }

        FragmentManager fm = getFragmentManager();
        FragmentTransaction fragmentTransaction = fm.beginTransaction();
        fragmentTransaction.replace(R.id.fragment_place, fr);
        fragmentTransaction.commit();
    }
}

```

MyFragmentActivity.java

activity\_my\_fragment.xml

```

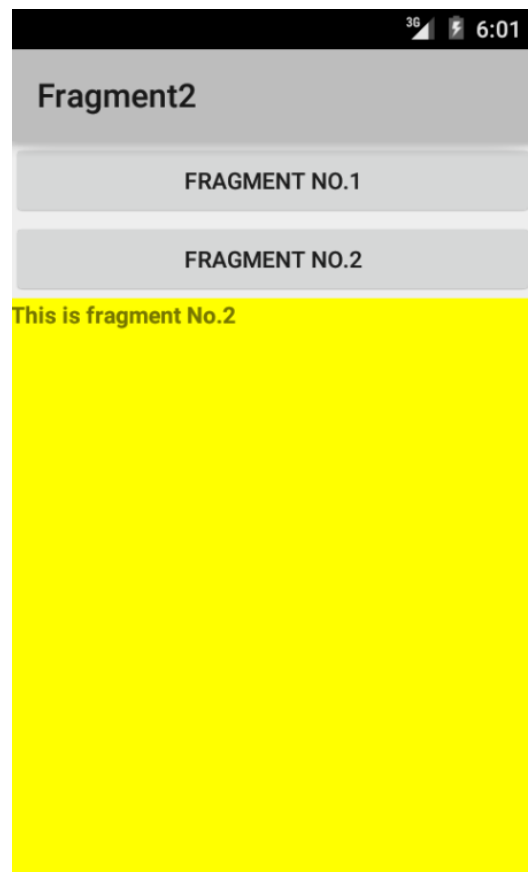
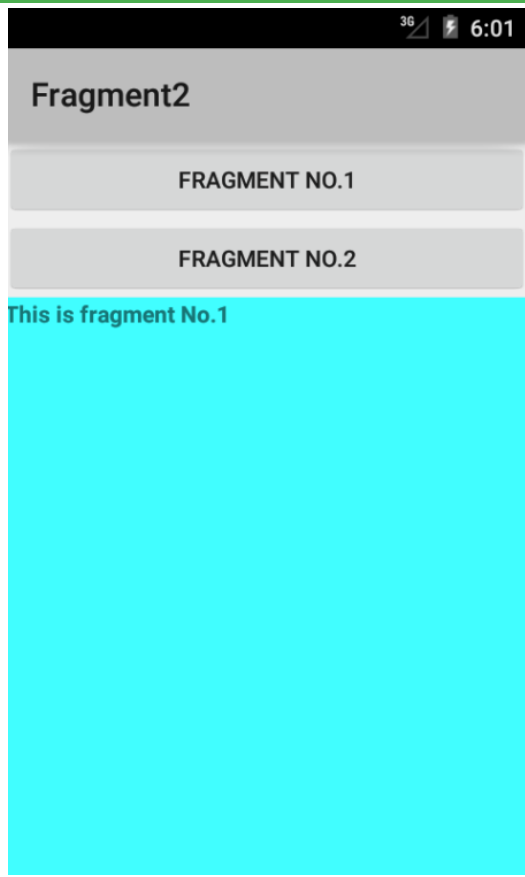
<Button
    android:id="@+id/button1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Fragment No.1"
    android:onClick="selectFrag" />

<Button
    android:id="@+id/button2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="selectFrag"
    android:text="Fragment No.2" />

<fragment
    android:name="android.support.design.widget.Snackbar"
    android:id="@+id/fragment_place"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

```

# Exemple Complet: Résultat



# ACTIONBAR ET TOOLBAR

# Menu d'Options et Action Bar

- Les APIs *Menu* étaient à la base destinées à définir le comportement (entre autres) du bouton *Menu* du téléphone, qui, depuis la version 3.0, est devenu obsolète

-Plusieurs terminaux récents sous Android ne possèdent plus ce bouton

- Les fonctionnalités du menu d'options sont désormais définies dans le *Action Bar*

-1. *App Icon*

-2. *Action Items*

-3. *Action Overflow*





# Action Bar

- La barre d'action (Action Bar) est une caractéristique qui identifie l'emplacement de l'utilisateur et lui fournit des actions et des modes de navigation
- Fournit:
  - Un espace dédié pour donner à votre application une identité et indiquer où se trouve l'utilisateur dans l'application
  - Des actions importantes et accessibles (recherche, par ex)
  - Navigation consistante et la possibilité de changer de vue dans une application (grâce aux tabulations ou liste déroulantes)

# Action Bar

- Les actions de la barre sont définies dans un fichier XML se trouvant dans le répertoire `res/menu`
- Chaque action est représentée par un item, avec de préférence trois attributs: ID, Icône et Titre
- Dans le fichier Manifest, associer un thème à votre application supportant cette barre:  
1. `android:theme="@style/Theme.AppCompat.Light"`
- Faire hériter votre activité de la classe `ActionBarActivity`
- Implémenter les méthodes (générées automatiquement):
  - *`onCreateOptionsMenu`* : construit le menu en appelant le menu XML
  - *`onOptionsItemSelected`* : définit le comportement de chacune des actions de la barre

# Action Bar

- Une ActionBar peut se composer de 4 parties :
  - L'icone de l'application : Etablie l'identité visuelle de l'application et permet aussi de naviguer dans l'application (remonter à la vue mère par exemple).
  - Dropdown Menu : Permet d'afficher le titre de la vue actuelle ainsi que naviguer dans l'application.
  - Actions principales : Définie les actions principales de votre application.
  - Autres actions : Permet d'accéder aux fonctionnalités moins importantes de l'application.

# Action Bar

- L'implémentation d'une ActionBar s'effectue dans le dossier menu et ressemble énormément à l'implémentation d'un menu.
- Afin de gérer la compatibilité avec des versions d'Android ne possédant pas d'ActionBar (entre 1.6 et 2.3.x), il est conseillé de remplacer son implémentation par des menus et cela en créant plusieurs dossiers :
  - fichier styles.xml dans le dossier values: Permet de spécifier le thème utilisé par l'application et donc un thème n'utilisant pas de barre d'action (conversion automatique en menu).
  - fichier styles.xml dans le dossier values-v11 : Permet de spécifier le thème utilisé (thème Honeycomb) utilisant les ActionBar fournies par cette version d'Android.
  - fichier styles.xml dans le dossier values-v14 : Permet de spécifier le thème utilisé (thème ICS / JellyBean) utilisant les ActionBar fournies par cette version d'Android.
  - Le chiffre se situant après la lettre v correspond au numéro de l'API Android cible

# Exemple

- L'exemple abordé dans cette partie servira à implémenter une ActionBar pour les appareils 3.x et supérieur et un menu pour les autres.
- Nous allons créer une application Android possédant les actions suivantes (accessible via ActionBar ou via menu) :
  - Recherche
  - Rafraichir
  - Aide
  - A propos
  - Paramètres

# Exemple

- Dans le dossier menu présent dans le dossier res, créez un fichier XML représentant votre menu

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

    <item
        android:id="@+id/menu_search"
        android:icon="@drawable/ic_action_search"
        android:showAsAction="ifRoom"
        android:title="@string/menu_search"/>

    <item
        android:id="@+id/menu_refresh"
        android:icon="@drawable/ic_action_refresh"
        android:showAsAction="ifRoom"
        android:title="@string/menu_refresh"/>

    <item
        android:id="@+id/menu_help"
        android:icon="@drawable/ic_action_help"
        android:showAsAction="ifRoom"
        android:title="@string/menu_help"/>

    <item
        android:id="@+id/menu_about"
        android:icon="@drawable/ic_action_about"
        android:showAsAction="never"
        android:title="@string/menu_about"/>

    <item
        android:id="@+id/menu_settings"
        android:icon="@drawable/ic_action_settings"
        android:showAsAction="never"
        android:title="@string/menu_settings"/>

</menu>
```

# Exemple

- Chaque élément du menu correspond à la déclaration d'une balise item
- Chaque élément possède un identifiant, une icône (nom conseillé : `ic_action_*`), un titre
- L'attribut `showAsAction` permet de spécifier le comportement de l'élément dans une ActionBar, il peut posséder les valeurs suivantes :
  - **ifRoom** : L'élément sera ajouté aux actions principales de l'ActionBar si une place est disponible
  - **never** : Ne jamais rajouter l'action aux actions principales de l'ActionBar
  - **always** : Toujours rajouter l'action aux actions principales de l'ActionBar. Cette valeur n'est pas conseillé car peut entrainer une superposition d'élément (si nombre de place disponible < nombre d'élément ajouté à l'ActionBar), préférez la valeur `ifRoom`.
  - **withText** : Toujours afficher le texte représentant l'action

# Exemple

- Implémenter les fichiers styles.xml présent dans les dossiers (values / values-v11 / values-v14)

```
<resources>
```

```
    <style name="AppTheme" parent="android:Theme.Light" />
```

```
</resources>
```

- Implémenter la méthode onCreateOptionsMenu afin de charger le menu crée précédemment.

@Override

```
public boolean onCreateOptionsMenu(Menu menu) {  
    getMenuInflater().inflate(R.menu.activity_main, menu);  
    return true;  
}
```

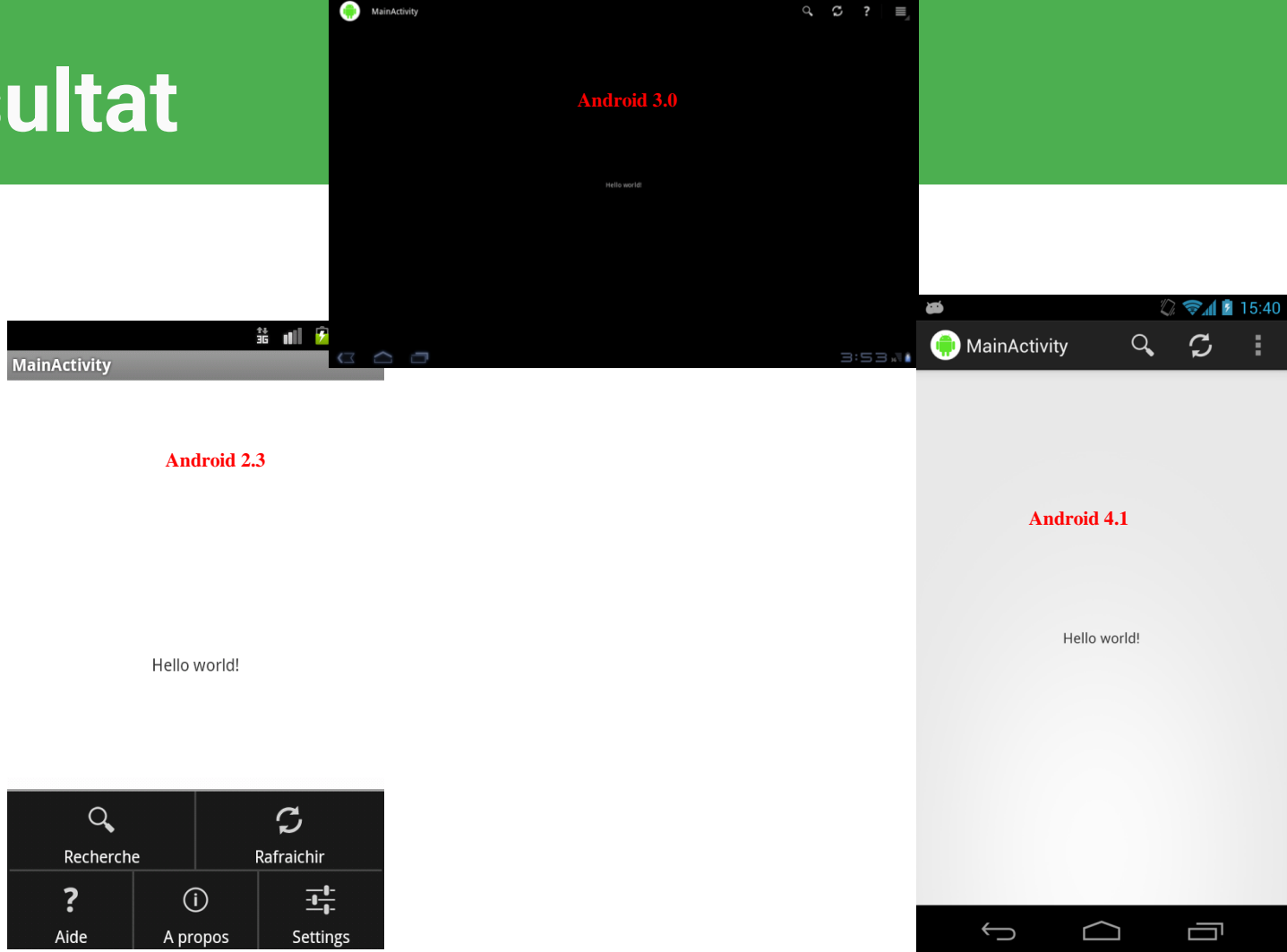


# Exemple

Afin de gérer le clic sur les différents éléments du menu, implémenter la méthode **onOptionsItemSelected** afin de spécifier l'action à exécuter en fonction du bouton cliqué :

```
public boolean onOptionsItemSelected(Menuitem item) {  
    switch (item.getItemId()) {  
        case R.id.menu_about: return true; // Comportement du bouton "A Propos"  
        case R.id.menu_help: return true; // Comportement du bouton "Aide"  
        case R.id.menu_refresh: return true; // Comportement du bouton "Rafrachir"  
        case R.id.menu_search: return true; // Comportement du bouton "Recherche"  
        case R.id.menu_settings: return true; // Comportement du bouton "Paramètres"  
        Default: return super.onOptionsItemSelected(item);  
    }  
}
```

# Résultat



# Toolbar

- La version 5 d'android (Lollipop) introduit une nouveauté nommé Toolbar. Ce nouveau composant devrait remplacer la barre d'action (ActionBar) d'une application.
- Ce composant possède plusieurs avantage par rapport à une action bar.
  - Il est déclaré dans le layout de votre activité : Ce qui permet de facilement personnalisé sa taille, son contenu, sa couleur, sa position
- Une toolbar peut contenir les différents éléments suivants :
  - Un bouton de navigation (ouverture d'un menu ou retour à l'activité parente).
  - Le logo / image identifiant l'application.
  - Une ou plusieurs vues personnalisées
  - Des actions

# Remplacer l'ActionBar par la Toolbar

Pour remplacer une ActionBar par une toolbar, il suffit de suivre les étapes suivantes :

- Modifier le thème de l'application
- Déclarer la Toolbar dans votre vue
- Remplacer l'action bar par la toolbar dans le code de l'activité
- Personnaliser la Toolbar

# Modifier le thème de l'application

- La première étape consiste à modifier le thème de l'application afin de rajouter les deux attributs `windowActionBar` (indique si l'application utilise une actionbar).
- Ce qui donnera pour le fichier `styles.xml` :

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">  
  
<item name="windowActionBar">false</item>  
  
</style>
```

# Déclarer la Toolbar dans votre vue

L'étape suivante consiste à déclarer la Toolbar dans un fichier de votre vue. Ce qui donnera pour notre exemple :

```
<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"
    tools:context=".MyActionBarActivity">
    <android.support.v7.widget.Toolbar
xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:layout_alignParentTop="true"
        android:background="#58FA58"
        android:minHeight="?android:attr/actionBarSize"
    "/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_below="@id/toolbar"
        android:layout_marginTop="10dp"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
</RelativeLayout>
```

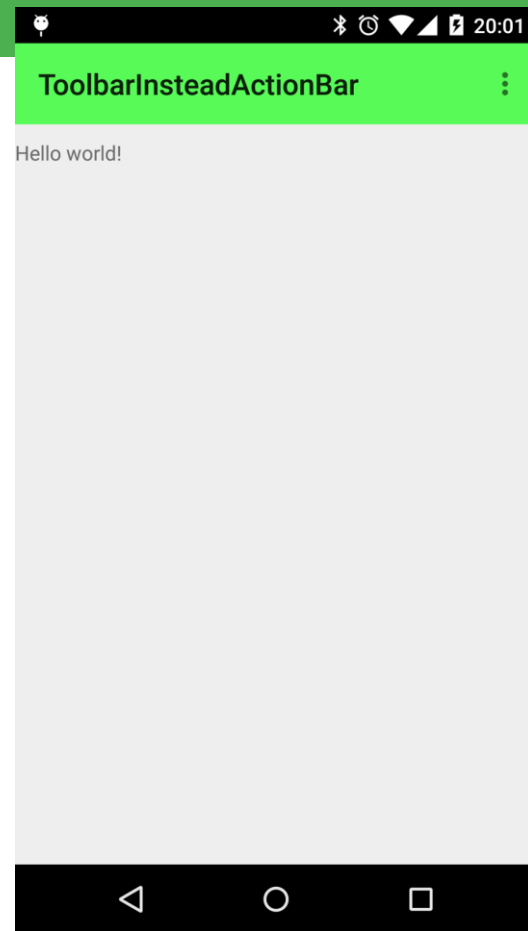
# Déclarer la Toolbar dans votre vue

- La vue contient :
  - Un RelativeLayout : servant de conteneur à la vue
  - La Toolbar : se trouvant toujours en haut de la vue (*alignParentTop*) et possédant un background et une taille (*minHeight*)
  - Une zone de texte
  - La valeur “?android:attr/actionBarSize” signifie que la hauteur de la toolbar sera celle fixée par la version d’Android. Vous pouvez remplacer par une valeur personnalisé mais cette valeur est conseillée car s’adaptera aux recommandations Google.

# Remplacer l'action bar par la toolbar dans le code de l'activité

- Indiquer à l'activité qu'elle utilise une toolbar au lieu d'une action bar et cela à l'aide de la méthode *setSupportActionBar*

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);  
setSupportActionBar(toolbar);
```





# Personnaliser la Toolbar

- Pour personnaliser une Toolbar, vous disposez des mêmes méthodes que celle disponible pour une barre d'action.
- Par exemple pour afficher le bouton de navigation up, il faut utiliser la méthode `setDisplayHomeAsUpEnabled`

```
ActionBar actionBar = getSupportActionBar();
```

```
//Permet d'afficher le bouton de navigation up sur l'application
```

```
actionBar.setDisplayHomeAsUpEnabled(true);
```