Programmation Mobile

UI/UX Design

2024 - 2025

Dr. Abdelkader Ouared

Agenda

Interface Utilisateur (UI)

Expérience Utilisateur

Utilisabilité

Android et les Design Patterns

Optimisation des performances de UI

Challenge

Agenda

Interface Utilisateur (UI)

Expérience Utilisateur

Utilisabilité

Android et les Design Patterns

Optimisation des performances de UI

Challenge

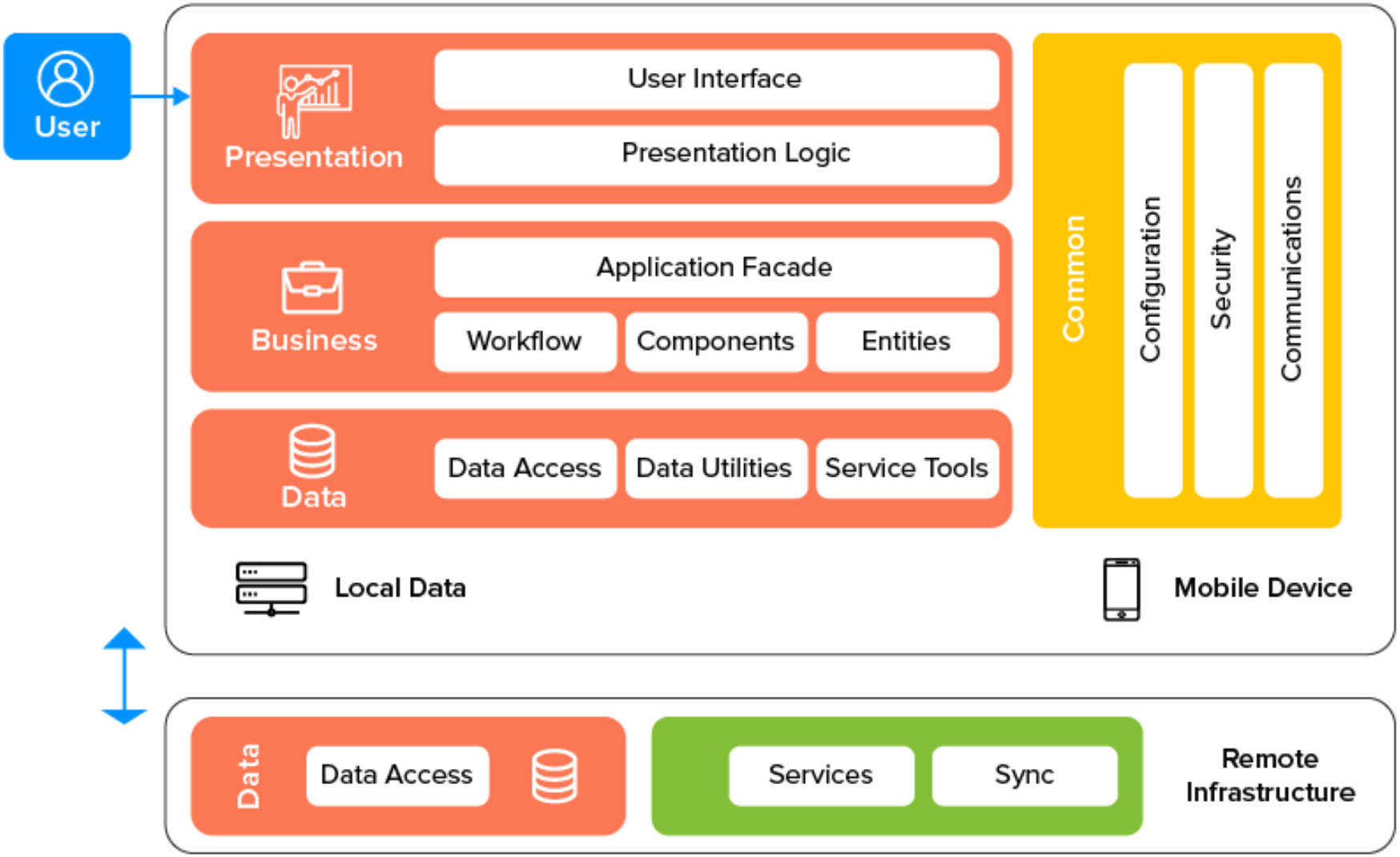
Agenda

Interface Utilisateur (UI)

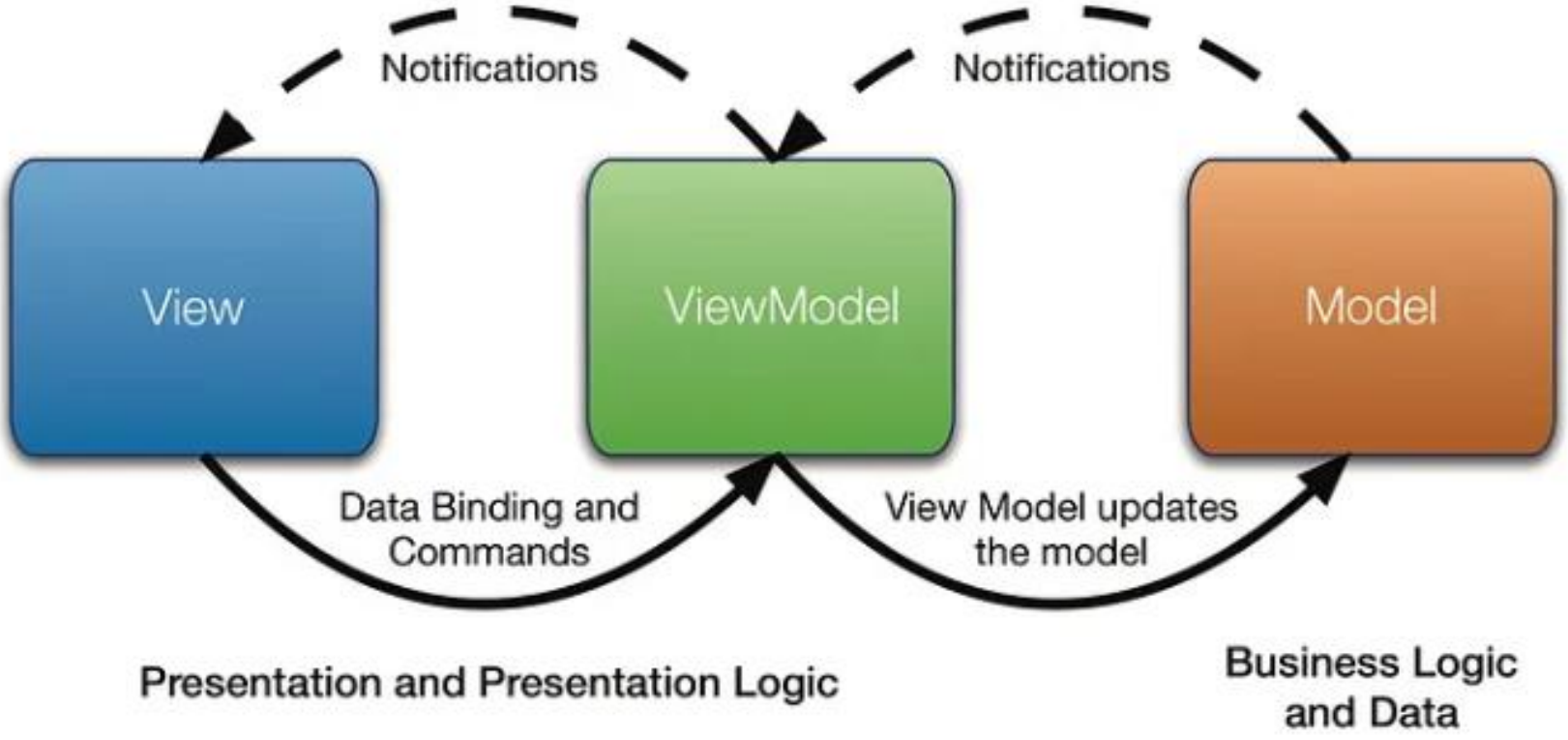
- Qu’est-ce que l’interface utilisateur Android ?
- Types d’interface utilisateur Android ?
- Qu'est-ce que ViewGroup et View ?
- Widgets
- Types de Layouts

Introduction

Qu’est-ce que l’architecture des applications mobiles ?

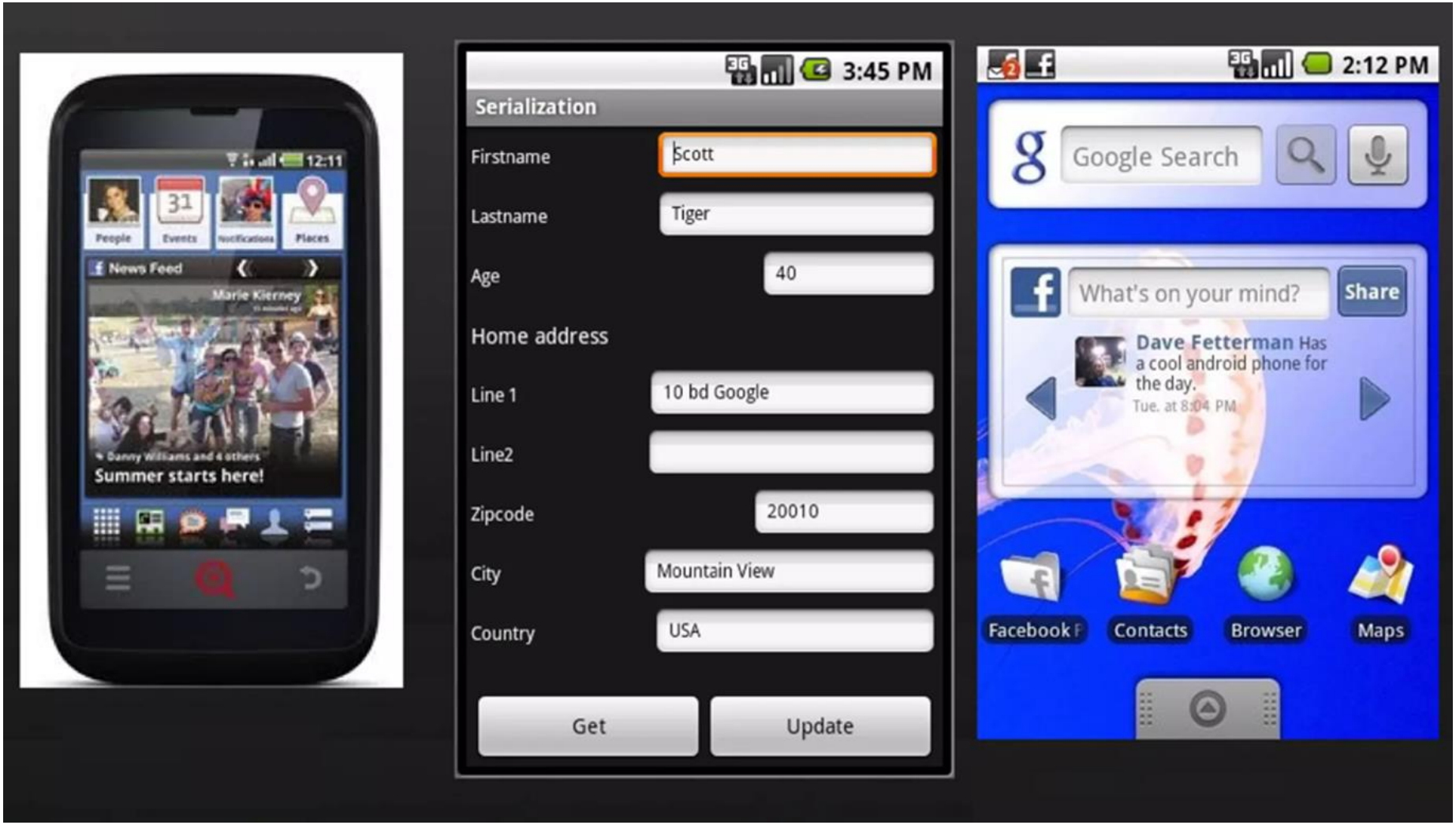


A mobile app architecture diagram of the MVP pattern



Qu’est-ce que l’interface utilisateur Android ?

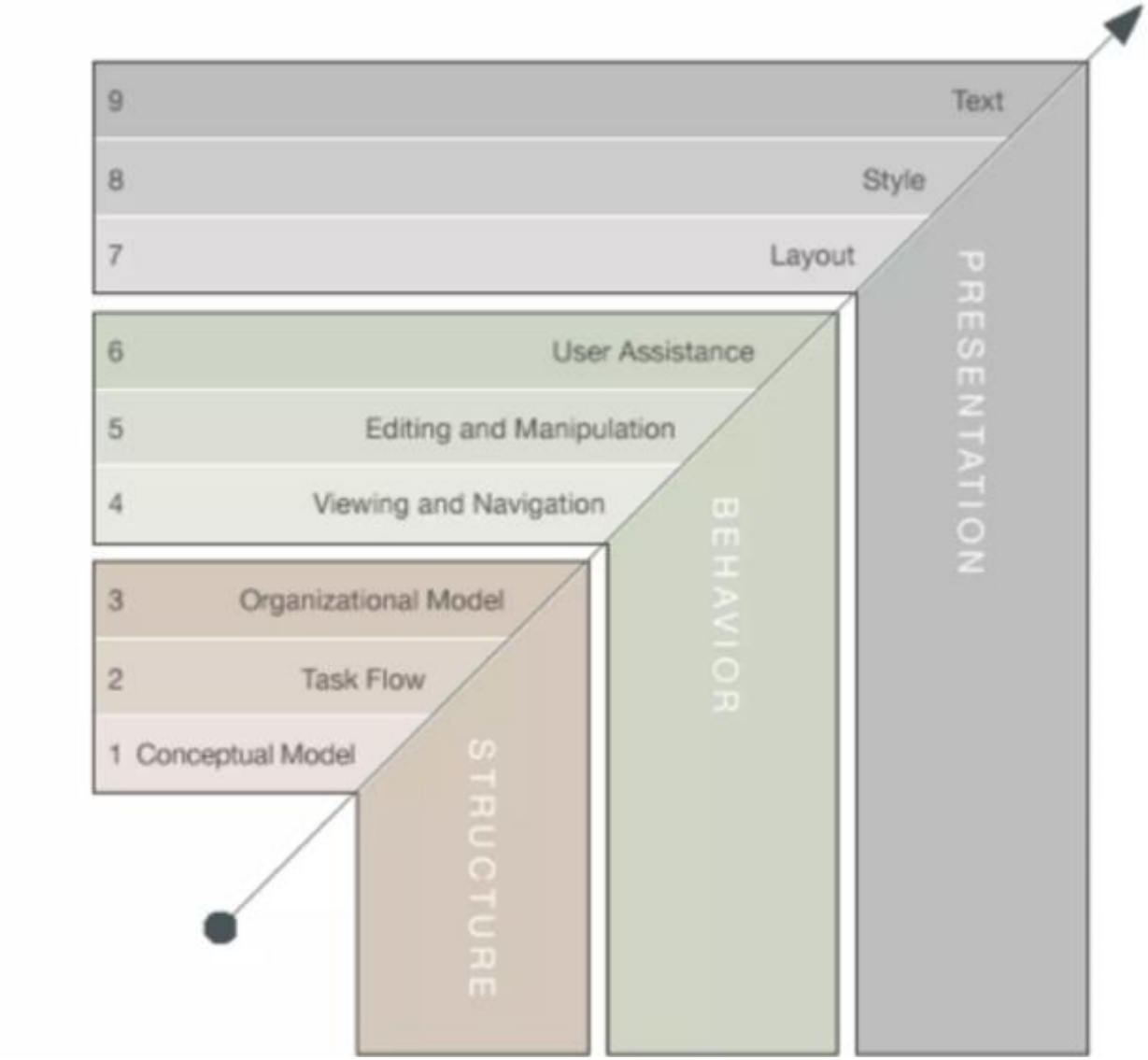
Interface utilisateur basée sur Android : quelques exemples



Qu'est-ce que l'interface utilisateur (UI) Android ?

- Une frontière ou surface commune entre le système interactif et l'utilisateur
- Tous les éléments qui assurent la communication entre le système interactif et l'utilisateur
- Interface utilisateur dans la plate-forme Android, tout comme les autres interfaces utilisateur basées sur Java

Éléments d'une interface utilisateur



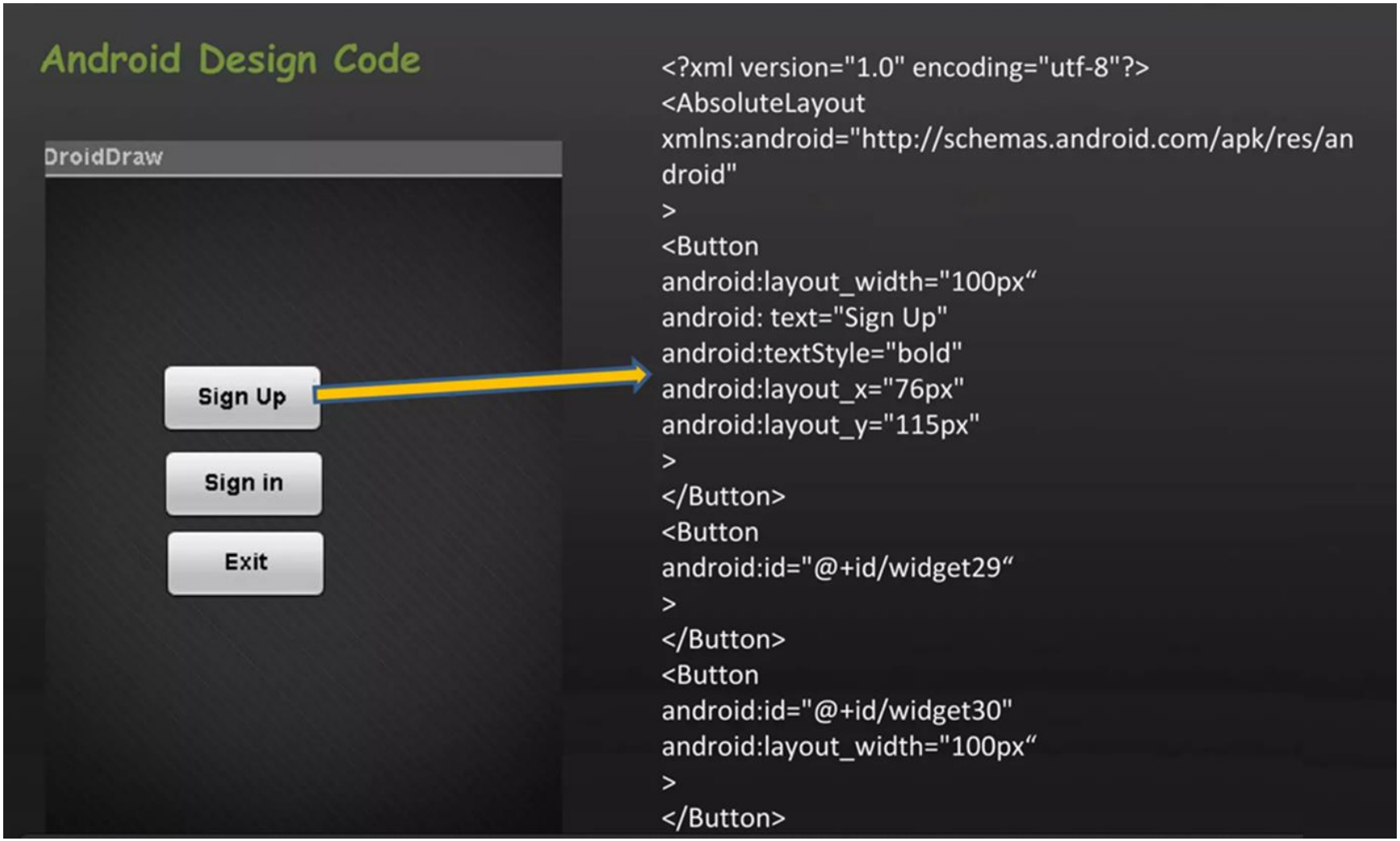
Création d'interface utilisateur des app mobile

- Création des interfaces utilisateur:
  - **Statique** [Drag and Drop]
  - **Dynamique** [Run time]

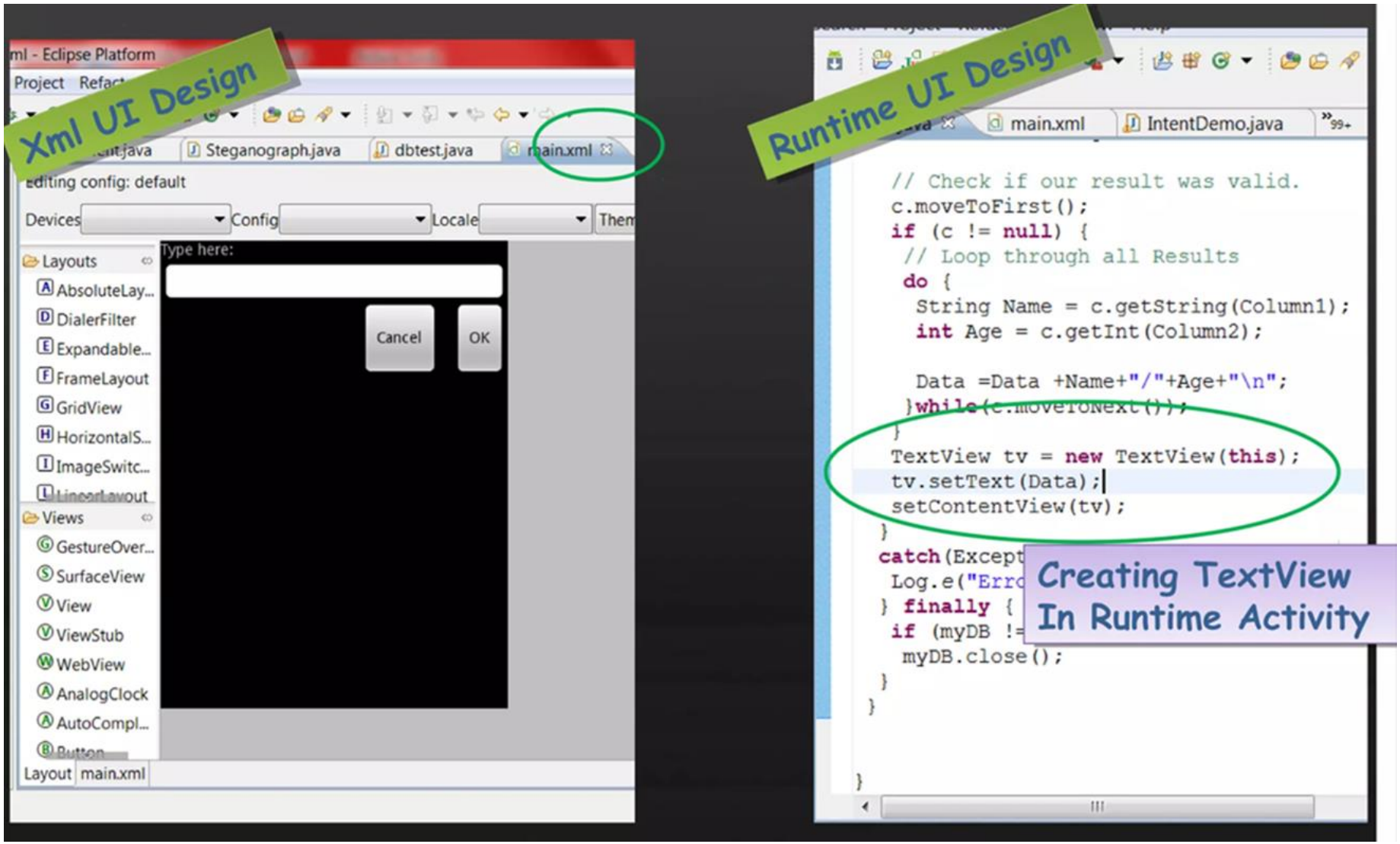
Interface utilisateur Java vs Interface utilisateur Android

Type d'application	Java [UI Design]	Android [UI Design]
Windows	Awt,Swings	XML JETPACK COMPOSE
Web based	Html,css,java script	

Création d'interface utilisateur des app mobile



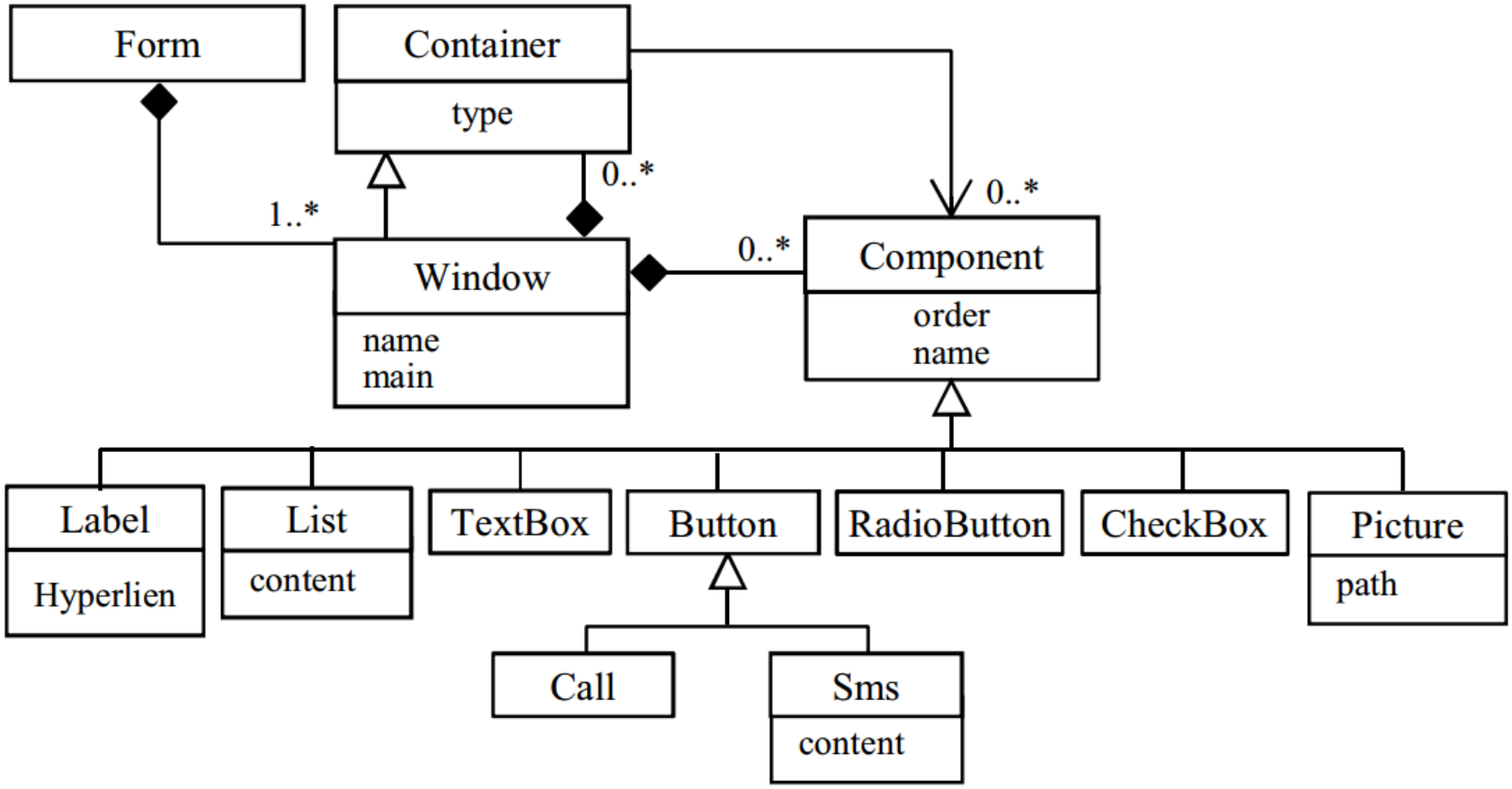
Création d'interface utilisateur des app mobile



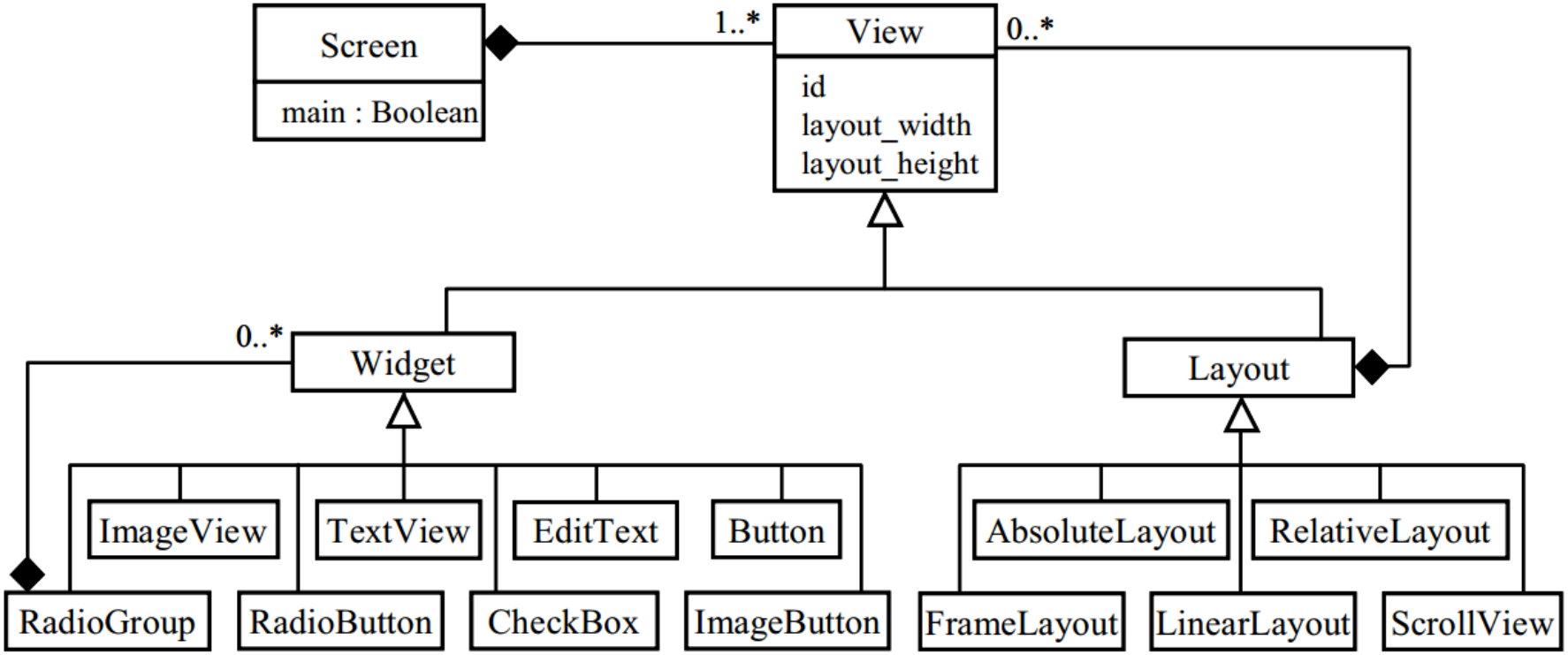
INTERFACE UTILISATEUR

- View/WIDGETS
- Layouts

Un métamodèle d'application mobile



Un extrait du métamodèle de l'application Android

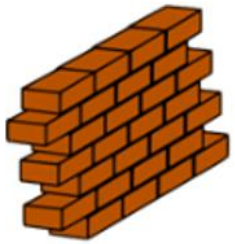


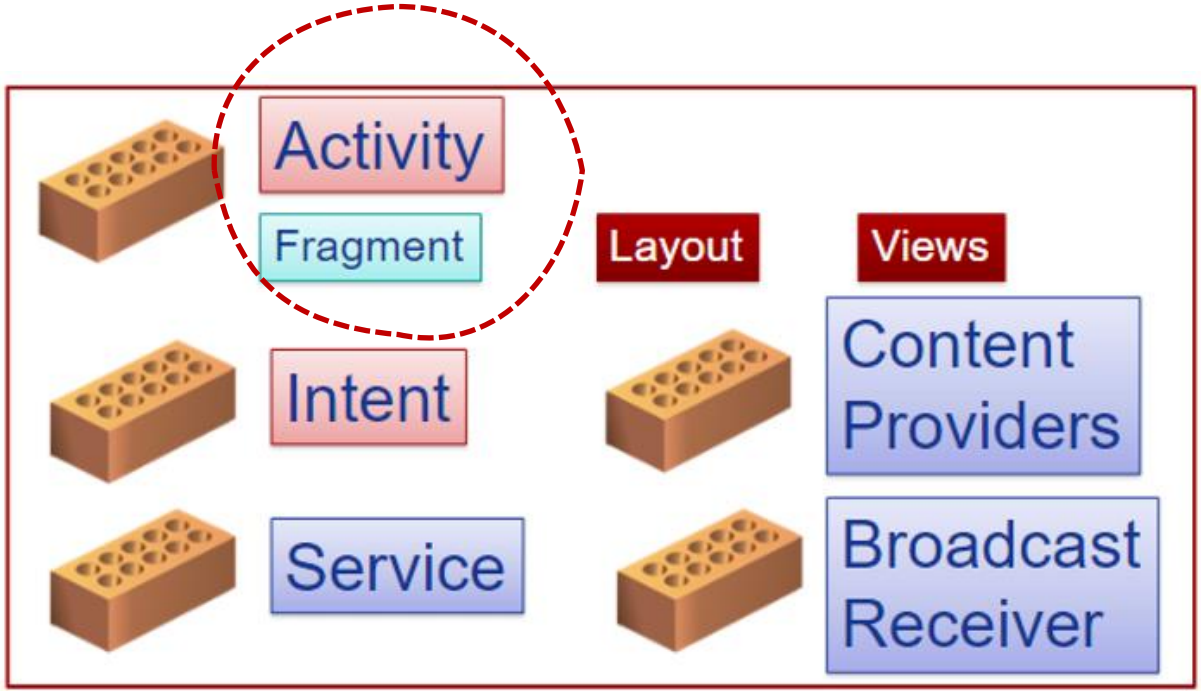
INTERFACE UTILISATEUR

- View/WIDGETS
- Layouts

Applications Android: Design

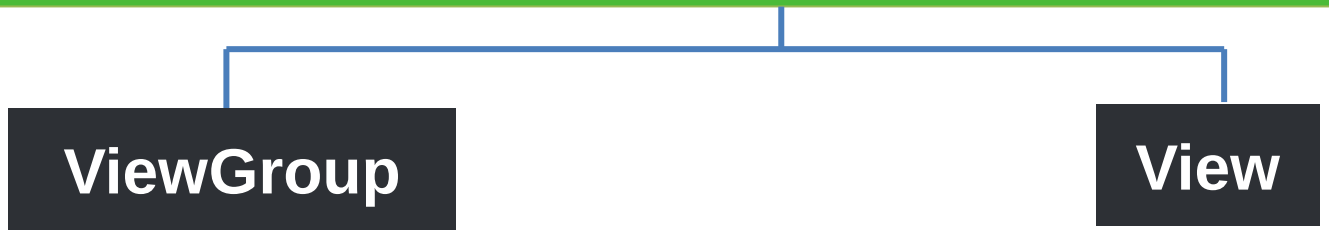
Développer une application Android, c'est utiliser de manière appropriée **les composants de base d'Android…**





Android: Views & Layout

Composants de **l'interface utilisateur (UI)** d'une activité



- **Conteneur** de View.
- Responsable du **placement** d'autres View sur l'écran
- Chaque layout doit étendre un **ViewGroup**

- **Composant de base (UI)**
- Peut gérer/produire des **événements**
- Nouveau composant: extension de **View**

Fichier XML ayant LinearLayout

Linear layout is a View Group that displays child View elements in a linear direction, either vertically or horizontally.

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

   <!-- More GUI components go here  -->

</LinearLayout>
```

Main.xml

Drap and Drop « Views »

Fichier XML ayant LinearLayout

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

   <TextView android:id="@+id/text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="This is a TextView" />

   <Button android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="This is a Button" />

   <!-- More GUI components go here  -->

</LinearLayout>
```

Views

View Group

Une fois votre mise en page (**layout**) est créée, vous pouvez charger **ressource** de mise en page (**layout**) à partir du code de votre application, dans votre implémentation « **Activity.onCreate()** »

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

Android: Views & Layout

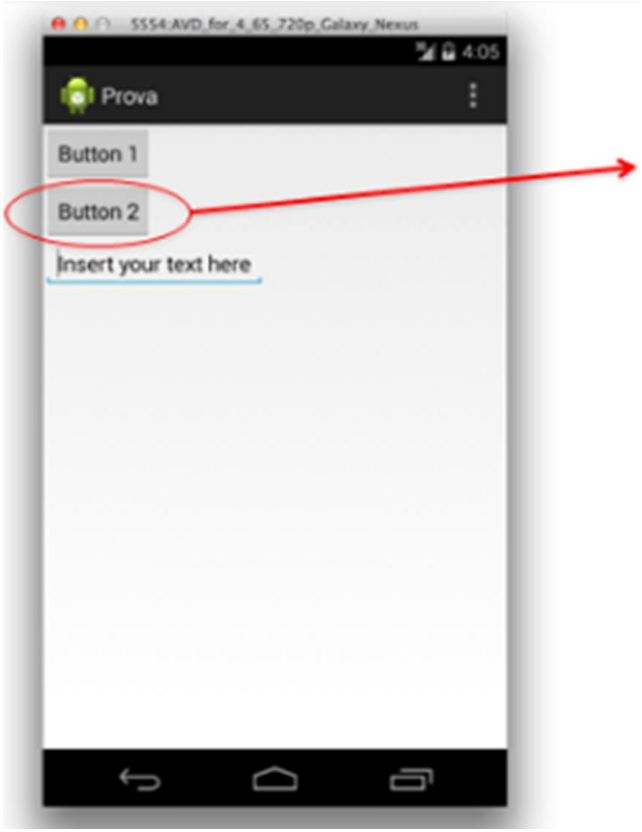
Composants de **l'interface utilisateur (UI)** d'une activité

- **Conteneur** de View.
- Responsable du **placement** d'autres View sur l'écran
- Chaque layout doit étendre un **ViewGroup**

- **Composant de base (UI)**
- Peut gérer/produire des **événements**
- Nouveau composant: extension de **View**

Android: Views objects

Views →éléments de base pour les composants de l'interface utilisateur



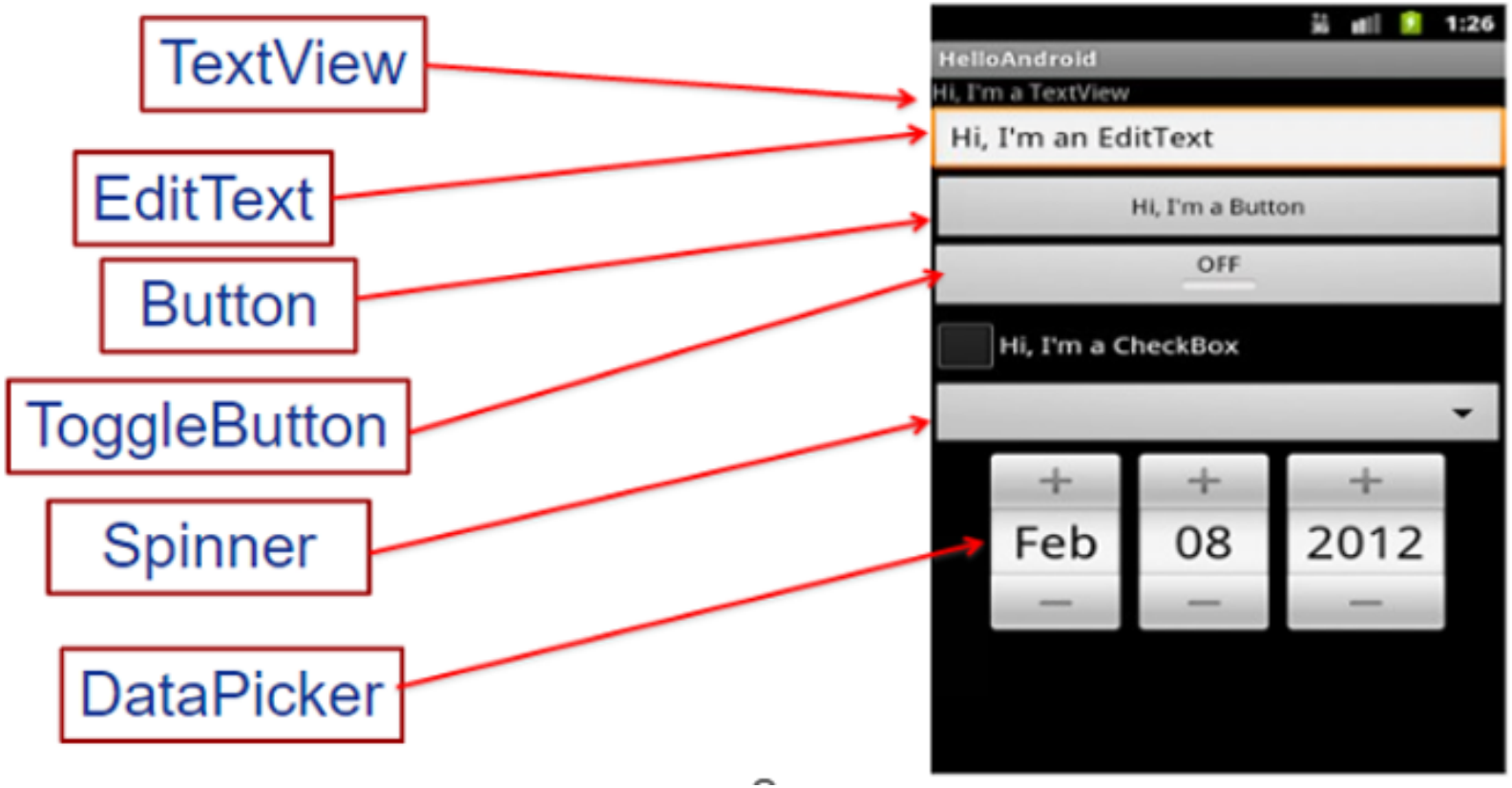
- ❖ Zone rectangulaire de l'écran
- ❖ Responsable du **dessin**
- ❖ Responsable de **la gestion des événements**

EXEMPLES d'objets **VIEWS** :

- GoogleMap
- WebView
- **Widgets** → sujet du jour
- User-defined Views

Android: Views objects

Widget → Composants d'interface utilisateur interactifs prédéfinis (android.view.widgets)



Widgets: code XML

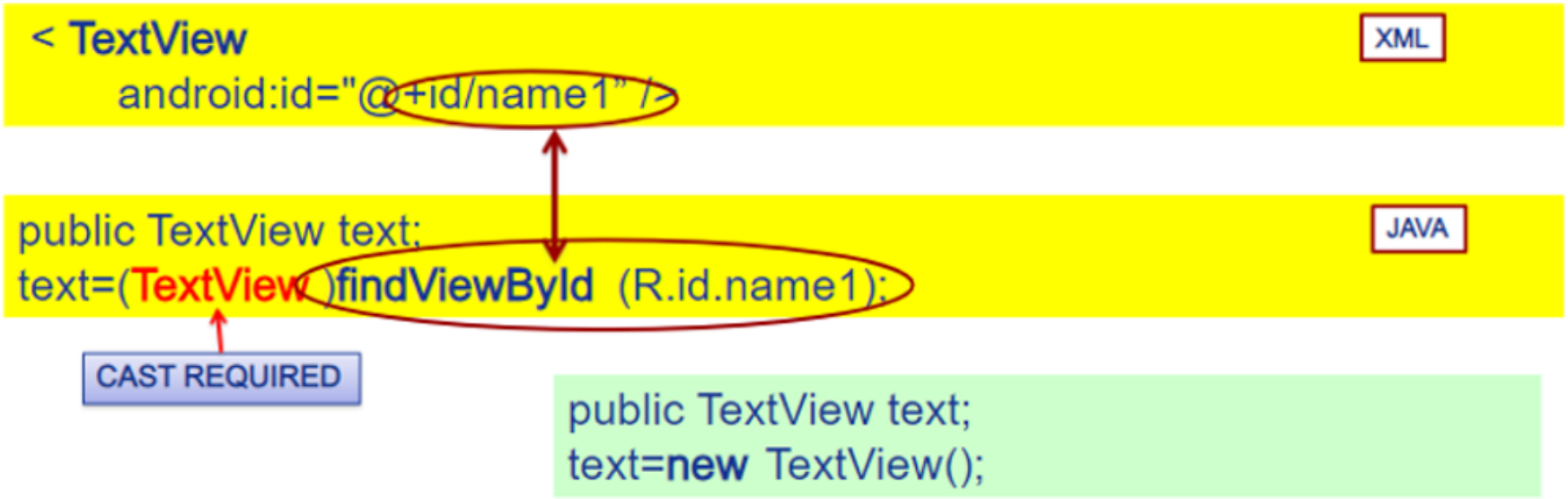
Widgets can be defined in the **XML layout files**

```
< TextView
    android:id="@+id/textLabel”
    android:layout_width="match_parent"
    android:layout_height="wrap_content”
    android:visibility="visible”
    android:enabled="true”
    android:scrollbars="vertical”
    android:text=“Hello World”
/>
```

Propriétés: définies via Android : … attributes

Widgets: Code Java et XML

Les **widgets** peuvent être définis en **XML** et accessibles depuis **Java/Koltlin**



Widgets: Code Java et XML

- Chaque widget peut avoir un **focus** et une **visibilité**, basés sur l'interaction de l'utilisateur.
  - L'utilisateur peut forcer le focus sur un composant spécifique via la méthode **requestFocus().**
  - L'utilisateur peut modifier la visibilité d'un composant spécifique via la méthode **setVisibility(int).**

```
public TextView text;
text=(TextView) findViewById(R.id.name1);
text.setVisibility(true)
text.requestFocus();
```

Widgets: TextView

- **XML tags: <TextView> </TextView>**
  - Peut être rempli de chaînes ou de **balises** HTML
  - Non directement modifiable par les utilisateurs
  - Habituellement utilisé pour afficher des informations **statiques**

```
<TextView
        android:text=”@string/textWelcome"
        android:id="@+id/textLabel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content”
/>
```

Widgets: Méthods TextView

- **Méthodes pour placer certains textes dans un TextView**
  - public void **setText**(CharSequence text)
  - public **CharSequence getText**()
  - public void **setSingleLine**(boolean singleLine)
  - public void **setHorizontallyScrolling**(boolean enable)
  - public void **setLines**(int lines)
  - public void **setEllipsize**(TextUtils.TruncateAt where)
  - public void **setHints**(CharSequence hints)

- TextUtils.TruncateAt.**END**
- TextUtils.TruncateAt.**MARQUEE**
- TextUtils.TruncateAt.**MIDDLE**
- TextUtils.TruncateAt.**START**

Widgets: EditText

- **XML** tags: **<EditText> </EditText>**
  - Similaire à un TextView, mais **modifiable** par les utilisateurs
  - Un **clavier** approprié sera affiché

```
<EditText
        android:text=”@string/textDefault"
        android:id="@+id/editText”
        android:inputType= “textCapSentences” |
                           “textCapWords” |
                 “textAutoCorrect” |
         “textPassword” |
“textMultiLane” />
```

Widgets: Button

❖ **XML** tags: **< Button></Button>**

- Sous-classe d'un TextView, mais non directement **modifiable** par les utilisateurs
- Peut générer des événements liés au clic, au clic long, etc.

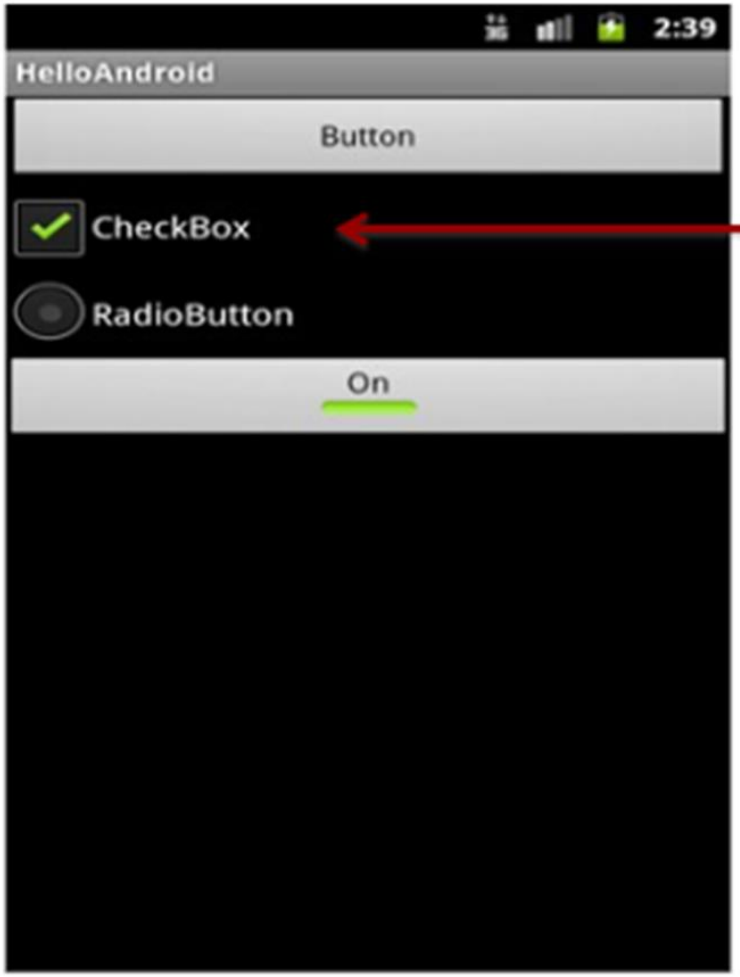
```
<Button
        android:text="@string/textButton"
        android:id="@+id/idButton"
        android:background="@color/blue”
/>
```

```
<selector>
<item
android:color="#ff819191”
android:state_pressed="true”>
</item>
</selector>
```

res/color/blue.xml

❖ **CompoundButton**: Button + *state* (checked/unchecked)

Widgets: Button and CompoundButton

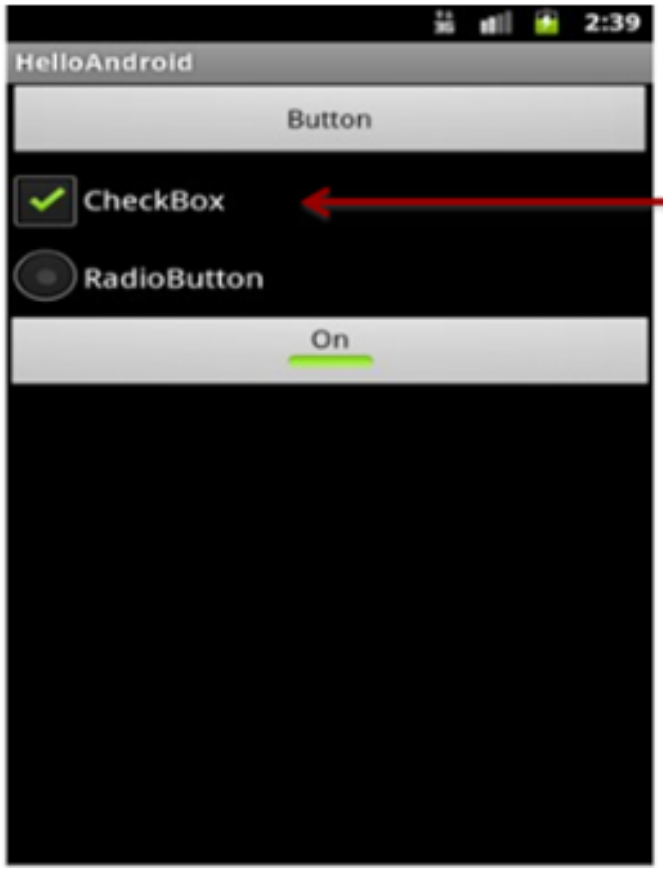


checkBox CompoundButton

XML tags: <**checkBox** >

```
<checkBox
android:layout_width="wrap_content”
android:layout_height="wrap_content”
android:id="@+id/buttonCheck”
android:text="CheckBox”
android:checked="true”
/>
```

Widgets: Button and CompoundButton



checkBox CompoundButton

- public boolean **isChecked**(): Returns true if the button is checked, false otherwise.
- public boolean **setChecked**(bool)

Listener:
onCheckedChangeListener

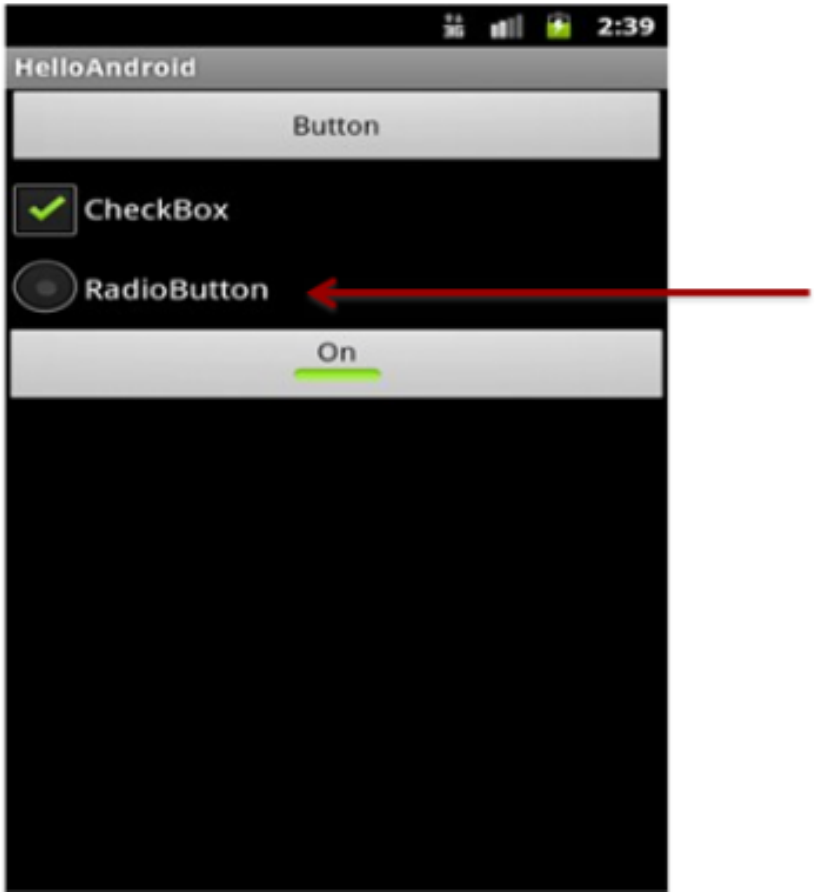
Widgets: Button and CompoundButton

radioButton CompoundButton

XML tags: <**RadioButton** >

```
<RadioButton
android:layout_width="wrap_content”
android:layout_height="wrap_content”
android:id="@+id/buttonRadio”
android:text="ButtonRadio”
android:checked="true”
/>
```

Widgets: Button and CompoundButton

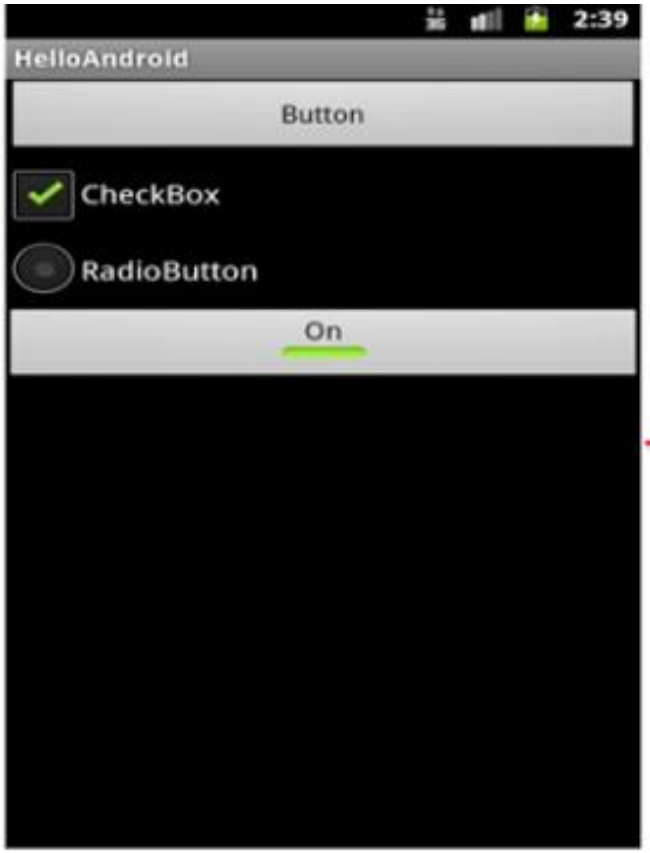


radioButton CompoundButton

- Define multiple (mutual-exclusive) options through a **<RadioGroup >** tag.
- Only one button can be checked within the same **RadioGroup** .

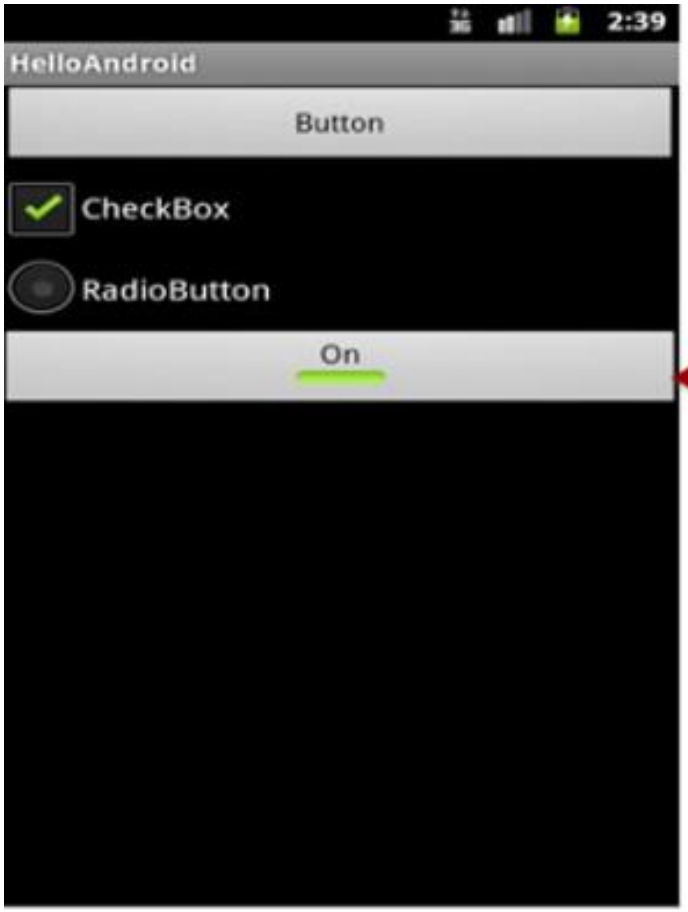
Listener:
OnCheckedChangeListener

Widgets: Button and CompoundButton




```
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/buttonRadio1"
        android:text="Option 1"
        android:checked="true" />
    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/buttonRadio2"
        android:text="Option 2" />
</RadioGroup >
```

Widgets: Button and CompoundButton



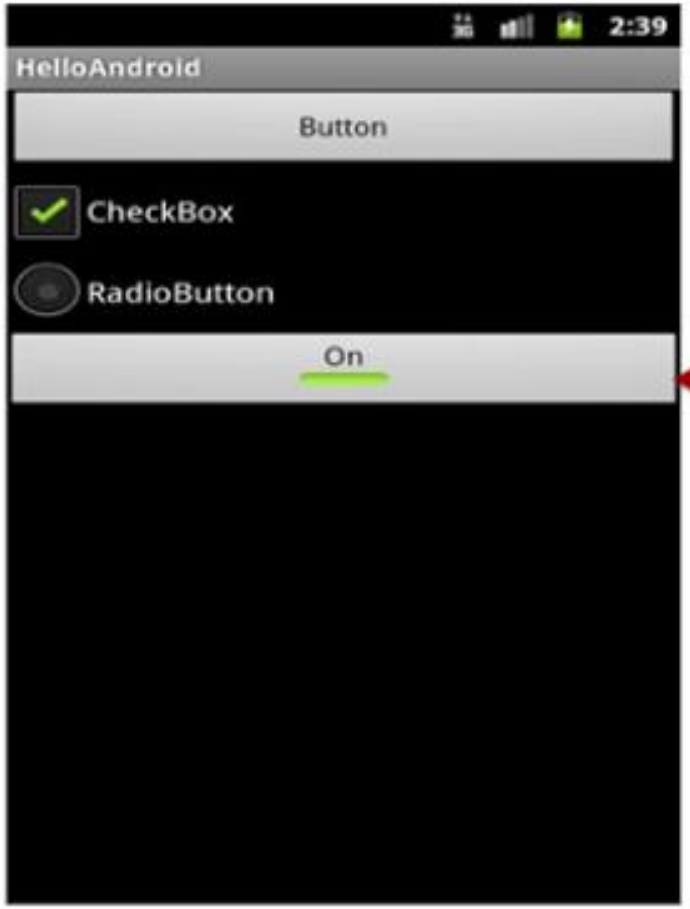
toggleButton CompoundButton

- It can assume only 2 states: checked/unchecked
- Different labels for the states with: **android:textOn** and **android:textOff** XML attributes.

Listener:

- OnCheckedChangeListener

Widgets: Button and CompoundButton

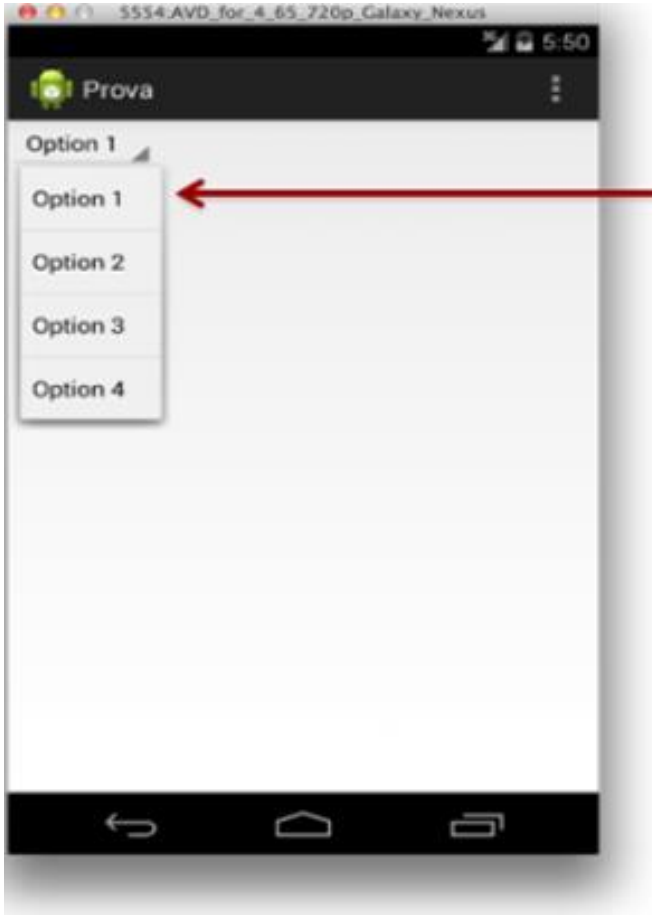


toggleButton CompoundButton

XML tags: **<ToggleButton>**

```
<ToggleButton
android:layout_width="wrap_content”
android:layout_height="wrap_content”
android:id="@+id/toggleButtonId”
android:textOn="Button ON”
android:textOff="Button OFF”
android:checked="false”
/>
```

Widgets: Spinners



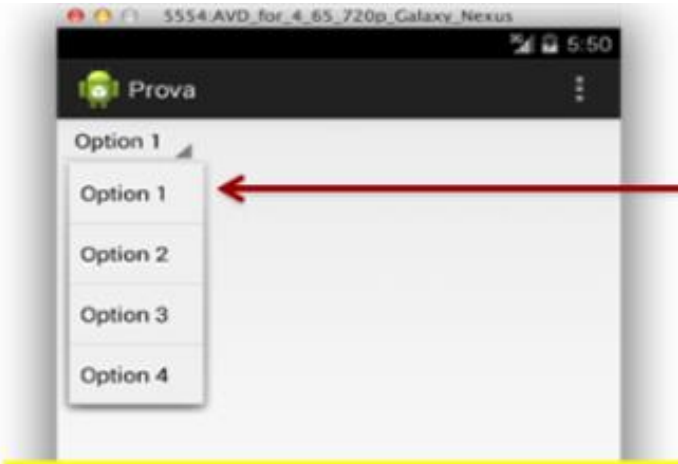
Spinner component

XML tags: **<Spinner>**

- Provides a quick way to select values from a specific set.
- The spinner value-set can be defined in XML (through the **entries** tag) or through the *SpinnerAdapter* in Java

Listener:
OnItemSelectedListener

Widgets: Spinners



Spinner component

XML tags: **<Spinner>**

```
<Spinner
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/spinnerId"
    android:entries="@array/stringOptions">
/>
```

```
<resources>
  <string-array  name="stringOptions">
        <item>Option 1</item>
        <item>Option 2</item>
        <item>Option 3</item>
        <item>Option 4</item>
</string-array>
</resources>
```

res/values.xml

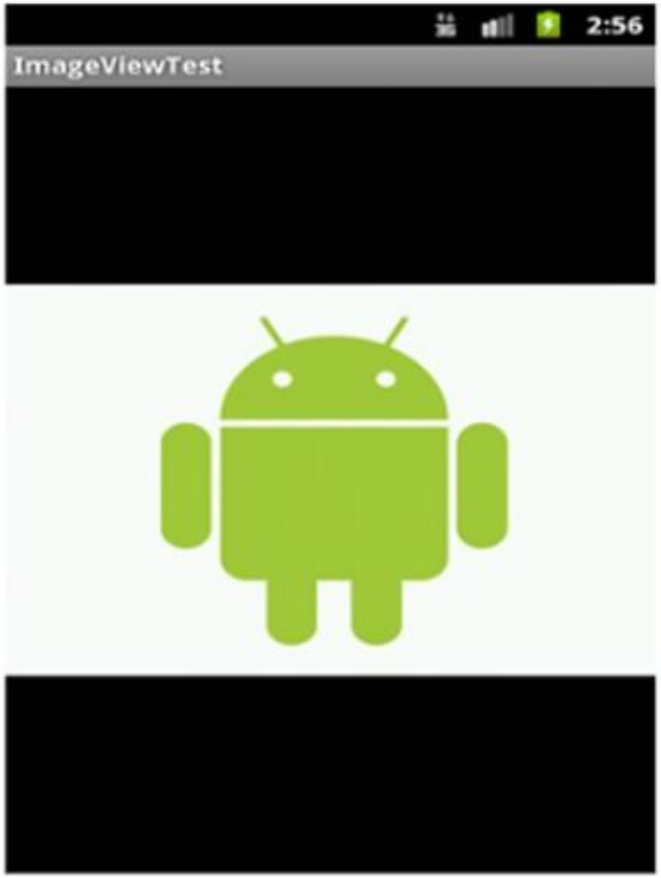
Widgets: Button and CompoundButton

DataPicker component

XML tags: <DatePicker>

```
<DatePicker
android:layout_width="wrap_content”
android:layout_height="wrap_content”
android:id=“@+id/datePickerId”
android:endYear="1990”
android:startYear="2014”
android:maxDate="10/10/2014”
/>
```

Widgets: ImageView



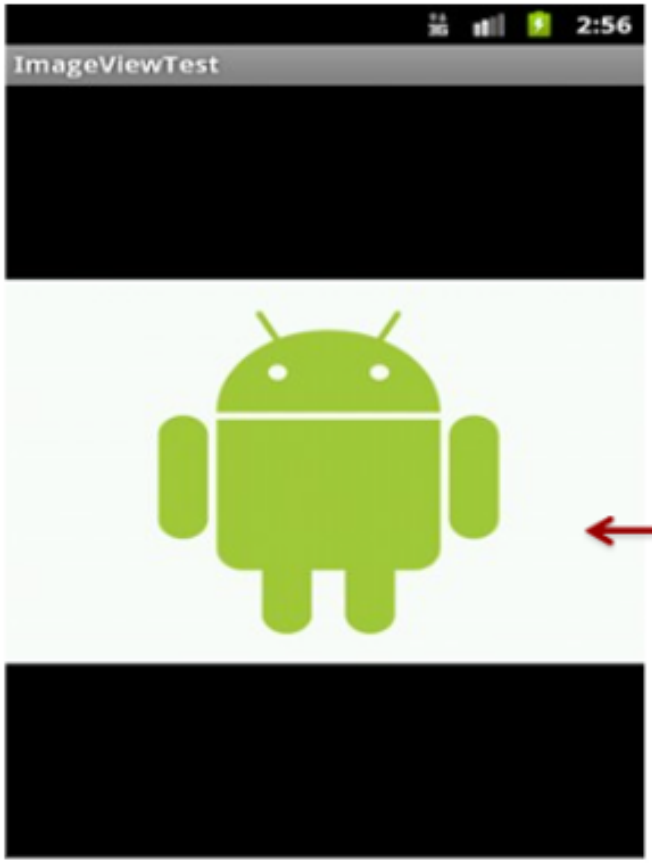
ImageView: subclass of View object**.**

Some methods to manipulate an image:

- void **setScaleType** (enum scaleType)
- void **setAlpha** (double alpha)
- void **setColorFilter** (ColorFilter color)

CENTER, CENTER_CROP, CENTER_INSIDE,
FIT_CENTER, FIT_END, FIT_START, FIT_XY, MATRIX

Widgets: ImageView



ImageView component

XML tags: <ImageView>

```
<ImageView
android:layout_width="wrap_content”
android:layout_height="wrap_content”
android:id="@+id/imageId”
android:src="@drawable/android”
/>
```

Source: **android** .jpg in drawable/

View & Events

View & Events

Les vues/Widgets sont des composants **interactifs** → Lors de certaines actions de l'utilisateur, un événement approprié sera déclenché

clic, clic long, focus, éléments sélectionnés, éléments cochés, glisser,…

Comment gérer les événements générés par une View ?

1. Directement depuis **XML**
2. Via **Event Listeners** (général, recommandé)

View and Events

Pour un ensemble limité de composants, il est possible de gérer les événements via des **rappels** indiqués dans la mise en page XML.

```
<Button
android:text="@string/textButton”
android:id="@+id/idButton”
android:onClick=”doSomething”
/>
```

XML Layout File

Java code

```
public void doSomething(View w) {
   // Code to manage the click
event
}
```

View & Events

Les vues/Widgets sont des composants **interactifs** → Lors de certaines actions de l'utilisateur, un événement approprié sera déclenché

clic, clic long, focus, éléments sélectionnés, éléments cochés, glisser,…

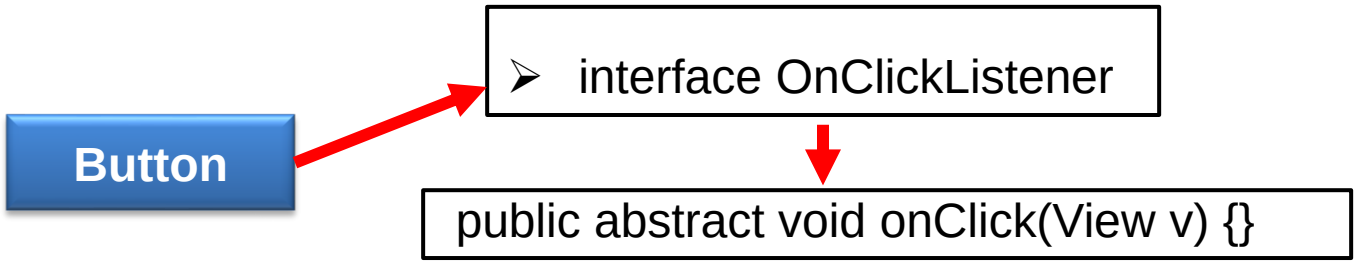
Comment gérer les événements générés par une View ?

1. Directement depuis **XML**
2. Via **Event Listeners** (général, recommandé)

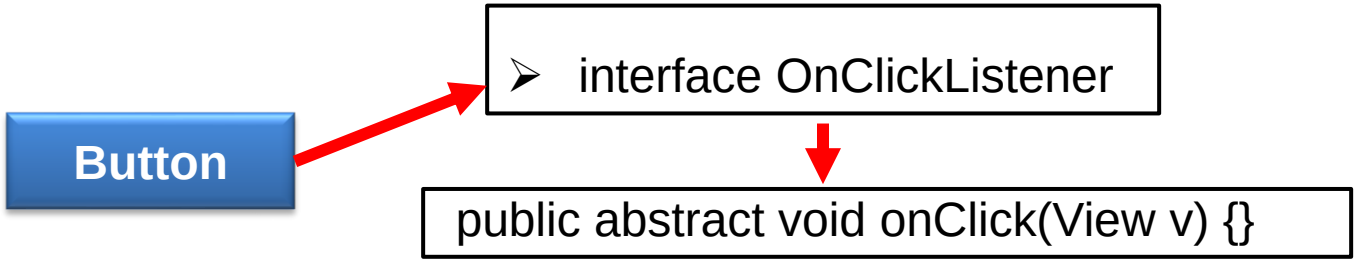
View & Events

Chaque View contient une collection d'interfaces imbriquées (listeners).

- Chaque listener gère un seul **event** …
- Chaque listener contient une seule méthode de rappel…
- Le rappel est invoqué lors de l'apparition de l'événement.



View & Events




```
…
Button button=(Button)findViewById(R.id.buttonNext);
button.performClick();
…
```

```
// Callback method
public void onClick(View v) {
        ….
}
```

View & Events: ActionListener

LISTE DES INTERFACES ACTIONLISTENER

- interface OnClickListener
  abstract method: *onClick()*
- interface OnLongClickListener
  abstract method: *onLongClick()*
- interface OnFocusChangeListener
  abstract method: *onFocusChange()*
- interface OnKeyListener
  abstract method: *onKey()*
- interface OnCheckedChangeListener
  abstract method: *onCheckedChanged()*

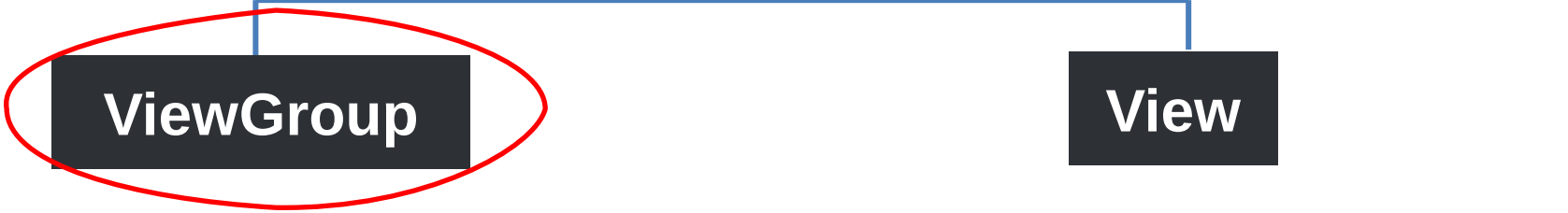
- interface OnItemSelectedListener
  abstract method: *onItemSelected()*
- interface OnCheckedChangeListener
  abstract method: *onCheckedChanged()*
- interface OnItemSelectedListener
  abstract method: *onItemSelected()*
- interface OnTouchListener
  abstract method: *onTouch()*
- interface OnCreateContextMenuListener
  abstract method: *onCreateContextMenu()*

INTERFACE UTILISATEUR

- View/WIDGETS
- Layouts

Android: Views & Layout

Composants de **l'interface utilisateur (UI)** d'une activité



- **Conteneur** de View.
- Responsable du **placement** d'autres View sur l'écran
- Chaque layout doit étendre un **ViewGroup**

- **Composant de base (UI)**
- Peut gérer/produire des **événements**
- Nouveau composant: extension de **View**

Android: ViewGroups

ViewGroup → définir **l'emplacement** des vues.

- Défini en code Java/kotlin (activity file)

- Défini en XML (layout file)

↓

- Utiliser des **balises XML** pour placer un ViewGroup
- Placez une View dans ViewGroup en utilisant ces **attributs XML**

android:layout_width

android:layout_height

} **match_parent I wrap_content**

Android: ViewGroups

Liste des **Layouts** les plus courantes fournies par Android

Nom	XML Tag	Description
LinearLayout	<LinearLayout> </LinearLayout>	arrange Views by aligning them on a single row or column
RelativeLayout	<RelativeLayout> </RelativeLayout>	arrange Views through relative positions
TableLayout	<TableLayout> </TableLayout>	arrange Views into rows and columns
FrameLayout	<FrameLayout> </FrameLayout>	arrange a single View within a Layout
AbsoluteLayout	<AbsoluteLayout> </AbsoluteLayout>	arrange Views through absolute positions
ConstraintLayout	<ConstraintLayout> </ConstraintLayout>	arrange views through constraints in ConstraintLayout:

- Un Layout peut être déclaré dans un autre layout

Android: ViewGroups

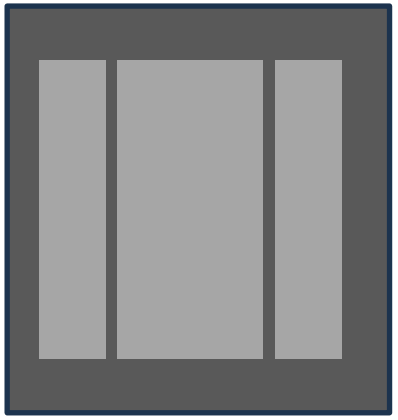
Liste des **Layouts** les plus courantes fournies par Android

Nom	XML Tag	Description
LinearLayout	<LinearLayout> </LinearLayout>	arrange Views by aligning them on a single row or column
RelativeLayout	<RelativeLayout> </RelativeLayout>	arrange Views through relative positions
TableLayout	<TableLayout> </TableLayout>	arrange Views into rows and columns
FrameLayout	<FrameLayout> </FrameLayout>	arrange a single View within a Layout
AbsoluteLayout	<AbsoluteLayout> </AbsoluteLayout>	arrange Views through absolute positions
ConstraintLayout	<ConstraintLayout> </ConstraintLayout>	arrange views through constraints in ConstraintLayout:

- Un Layout peut être déclaré dans un autre layout

Android: ViewGroups

LinearLayout → view group that aligns all children in a single direction, vertically or horizontally



- **Orientation** can be declared through XML tag

  android:orientation= HORIZONTAL|VERTICAL

- **Orientation** can also be declared in Java through

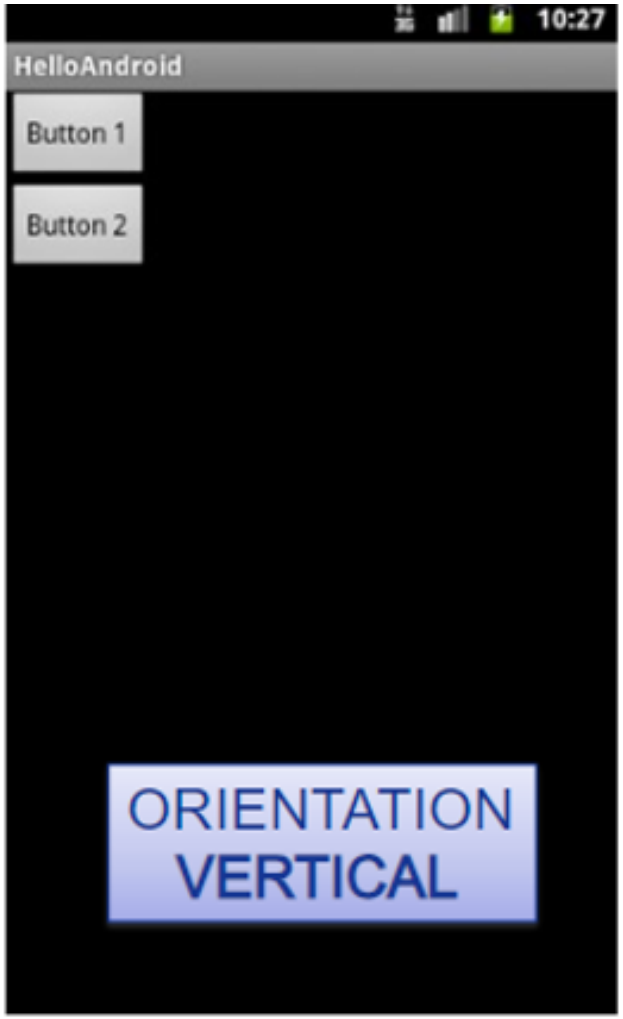
  setOrientation(int orientation)

- Views has also other two attributes:

  gravity → Align the View with its parent

  weigth → How much space is assigned to the View

Android: ViewGroups



Android: ViewGroups

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >        <!-- Also horizontal →
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/buttonString1" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/buttonString2" />
</LinearLayout>
```

Android: ViewGroups

❖ For each **View** in a LinearLayout, we can set the

android:weight XML property (integer value)

Importance of a View, how much it can **expand**

```
<LinearLayout>
<Button
…android:layout_width=match_parent
  android:weight=1/>
<Button …
  android:layout_width=match_parent
  android:weight=1/>
</LinearLayout>
```

Android: ViewGroups

❖ For each **View** in a LinearLayout, we can set the **layout_gravity** XML property

Align a View with its parent (LinearLayout)

```
<LinearLayout>
<Button  …
android:layout_width=match_parent
android:weigth=1/>
<Button …
android:layout_width=match_parent
android:layout_gravity=“center_vertical”
android:weight=2/>
</LinearLayout>
```

Android: ViewGroups

Liste des **Layouts** les plus courantes fournies par Android

Nom	XML Tag	Description
LinearLayout	<LinearLayout> </LinearLayout>	arrange Views by aligning them on a single row or column
RelativeLayout	<RelativeLayout> </RelativeLayout>	arrange Views through relative positions
TableLayout	<TableLayout> </TableLayout>	arrange Views into rows and columns
FrameLayout	<FrameLayout> </FrameLayout>	arrange a single View within a Layout
AbsoluteLayout	<AbsoluteLayout> </AbsoluteLayout>	arrange Views through absolute positions
ConstraintLayout	<ConstraintLayout> </ConstraintLayout>	arrange views through constraints in ConstraintLayout:

- Un Layout peut être déclaré dans un autre layout

Android: ViewGroups

RelativeLayout → View group that displays all **Views** based in **relative** positions.

RELATIVE= **COMPARED** to the **PARENT LAYOUT**

```
android:alignParentBottom="true|false"
android:alignParentTop="true|false"
android:alignParentLeft="true|false"
android:alignParentRight="true|false"
android:alignParentStart="true|false"
android:alignParentEnd="true|false"
...
```

Android: ViewGroups

RelativeLayout → View group that displays all **Views** based in relative positions.

RELATIVE= **COMPARED** to **SIBLING VIEWs**

```
android:toLeftOf=   ID
android:toRightOf=  ID
android:toStartOf=  ID
android:toEndOf=    ID
...
```

XML Identifier
of the View
used as reference
point

Android: ViewGroups

ACTIVITY_MAIN.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" >

    <EditText
        android:id="@+id/username"        android:text="username"
        android:inputType="text"
        android:layout_width="wrap_content"        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_toRightOf="@+id/usernameLabel" >
    </EditText>

    <TextView
        android:id="@+id/usernameLabel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/username"
        android:text="Username" />
```

CONTINUE

Android: ViewGroups

```
<EditText
    android:id="@+id/password"        android:text="password"
    android:inputType="textPassword"
    android:layout_below="@+id/username"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/username"
    android:layout_alignParentRight="true"
    android:layout_toRightOf="@+id/usernameLabel" >
</EditText>

<TextView
    android:id="@+id/passwordLabel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/password"
    android:text="Password" />
</RelativeLayout>
```

ACTIVITY_MAIN.XML

Android: ViewGroups

Liste des **Layouts** les plus courantes fournies par Android

Nom	XML Tag	Description
LinearLayout	<LinearLayout> </LinearLayout>	arrange Views by aligning them on a single row or column
RelativeLayout	<RelativeLayout> </RelativeLayout>	arrange Views through relative positions
TableLayout	<TableLayout> </TableLayout>	arrange Views into rows and columns
FrameLayout	<FrameLayout> </FrameLayout>	arrange a single View within a Layout
AbsoluteLayout	<AbsoluteLayout> </AbsoluteLayout>	arrange Views through absolute positions
ConstraintLayout	<ConstraintLayout> </ConstraintLayout>	arrange views through constraints in ConstraintLayout:

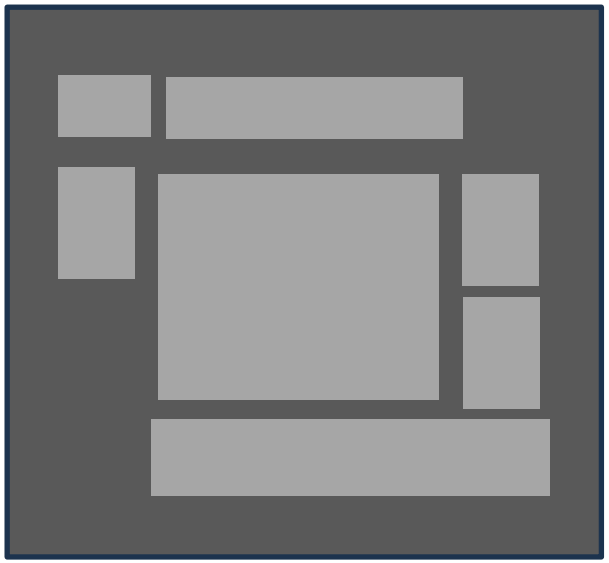
- Un Layout peut être déclaré dans un autre layout

Android: ViewGroups

TableLayout → View group that arranges all **Views** into rows and columns (as an HTML table)

PROPERTIES

1.Consists of a list of **TableRaw** objects, each defining a raw of the Table.
2.The **width** of a column is defined by the row with the **widest** cell in that column.
3.Cells can be **empty**, or can **span** multiple columns (like in HTML).
4.Border lines of the cells are not displayed.
PROPERTIES Android: ViewGroups



Android: ViewGroups

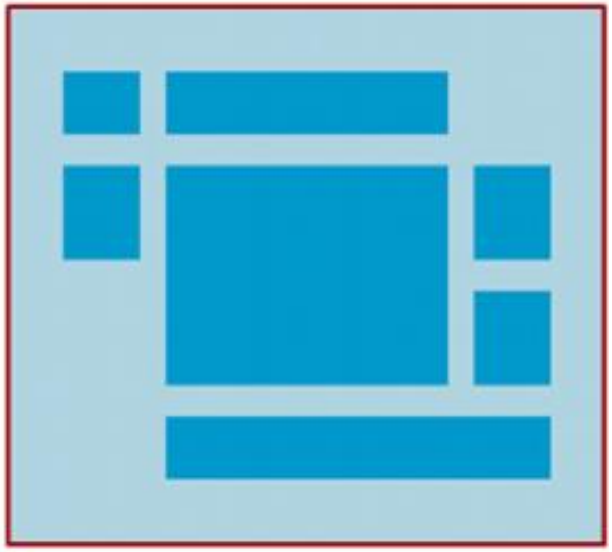
TableLayout → View group that arranges all **Views** into rows and columns (as an HTML table)

ADDITIONAL XML attributes

```
android:layout_column
android:layout_span
android:collapsecolumn
android:shrinkColumns
android:stretchColumns
```

Apply only to **width**

`layout_width` is always **WRAP_CONTENT**
`layout_height` is **WRAP_CONTENT** (default)



Android: ViewGroups

ACTIVITY_MAIN.XML

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout android:layout_width="fill_parent"
    android:layout_height="fill_parent" xmlns:android=schemas.android.com/apk/res/android
     android:id="@+id/tableLayout">
    <TableRow android:layout_width="wrap_content" android:layout_height="wrap_content" android:id="@+id/firstRow">
    <Button
          android:id="@+id/button1"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
               android:text="Button" />
    <Button android:id="@+id/button2"
               android:layout_width="match_parent"
               android:layout_height="match_parent"
               android:text="Button" />
    <Button android:id="@+id/button3"
               android:layout_width="match_parent"
               android:layout_height="match_parent"
               android:text="Button" />
    </TableRow>
```

CONTINUE

Android: ViewGroups

```
<TableRow
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/secondRow">

            <Button
            android:layout_column="1"
            android:layout_span="2"
                android:id="@+id/button4"
             android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Button">
            </Button>
</TableRow>
</TableLayout>
```

ACTIVITY_MAIN.XML

Android: ViewGroups

Liste des **Layouts** les plus courantes fournies par Android

Nom	XML Tag	Description
LinearLayout	<LinearLayout> </LinearLayout>	arrange Views by aligning them on a single row or column
RelativeLayout	<RelativeLayout> </RelativeLayout>	arrange Views through relative positions
TableLayout	<TableLayout> </TableLayout>	arrange Views into rows and columns
FrameLayout	<FrameLayout> </FrameLayout>	arrange a single View within a Layout
AbsoluteLayout	<AbsoluteLayout> </AbsoluteLayout>	arrange Views through absolute positions
ConstraintLayout	<ConstraintLayout> </ConstraintLayout>	arrange views through constraints in ConstraintLayout:

- Un Layout peut être déclaré dans un autre layout

Android: ViewGroups

FrameLayout → Block out an area on the screen to display a single item (i.e. a single View).

- It should be used to display a **single View** within the Layout
- Multiple views can be controlled through **android:layout_gravity**

AbsoluteLayout → Arrange Views on the screen by specifying absolute x-y positions of each View.

- **Deprecated**, since it is dependant of the **screen resolution**

Agenda

Interface Utilisateur (UI)

Expérience Utilisateur

Utilisabilité

Android et les Design Patterns

Optimisation des performances de UI

Challenge

Thank you !

Questions ? abdelkader.ouared@univ-tiaret.dz

www. .univ-tiaret.dz