



Développement Android

Partie 3



Les Intents

Principe des *Intents*

- ▶ Le passage d'une activité à l'autre se fait à l'aide d'*Intents*.
- ▶ Les *Intents* permettent de gérer l'envoi et la réception de messages afin de faire coopérer les applications.
- ▶ Le but des *Intents* est de déléguer :
 - ▶ une autre activité de l'application courante,
 - ▶ une action à un autre composant,
 - ▶ ou une autre application.
 - ▶ Par exemple, composer un numéro de téléphone
 - ▶ ou envoyer ou recevoir des messages (SMS/MMS), etc.

Principe des *Intents*

- ▶ Un objet *Intent* peut contenir les informations suivantes :
 - ▶ Le nom du composant ciblé (facultatif)
 - ▶ L'action à réaliser, sous forme de chaîne de caractères
 - ▶ Les données : contenu MIME et URI
 - ▶ Des données supplémentaires sous forme de paires clef/valeur
 - ▶ Une catégorie pour cibler un type d'application
 - ▶ Des drapeaux (informations supplémentaires)
- ▶ On peut envoyer des *Intents* informatifs pour faire passer des messages. Mais on peut aussi envoyer des *Intents* suivant à lancer une nouvelle activité.

URI : Uniform Resource Identifier

URL : Uniform Resource Locator

Lancement d'une activité

- ▶ Soit une application contenant deux activités : Activ1 et Activ2. La première lance la seconde par :

```
Intent intent = new Intent(Activ1.this,Activ2.class);  
startActivity(intent);
```

- ▶ L'instruction **startActivity** démarre **Active2**. Celle-ci se met devant **Activ1** qui se met alors en **sommeil**,
- ▶ Dans certain cas, il ne faut pas mettre **this** mais faire appel à **getApplicationContext()** si l'objet manipulant l'Intent n'hérite pas de **Context**.
- ▶ Ce bout de code est employé par exemple lorsqu'un bouton, un menu, etc. est cliqué. Seule contrainte : que ces **deux activités** soient **déclarées** dans **AndroidManifest.xml**.

Lancement d'une application Android

- ▶ S'il s'agit de passer la main à une autre application, on donne au constructeur de l'*Intent* les données et l'**URI** cible: l'OS est chargé de trouver une application pouvant répondre à l'*Intent*.
- ▶ Pour ouvrir le navigateur sur un URL spécifique :

```
String url = "http://www.ensaf.ac.ma/..";  
intent = new Intent (Intent.ACTION_VIEW, Uri.parse(url));  
startActivity(intent);
```

- ▶ L'action VIEW avec un URI est interprétée par Android, cela fait ouvrir automatiquement le navigateur.

Lancement d'une application Android

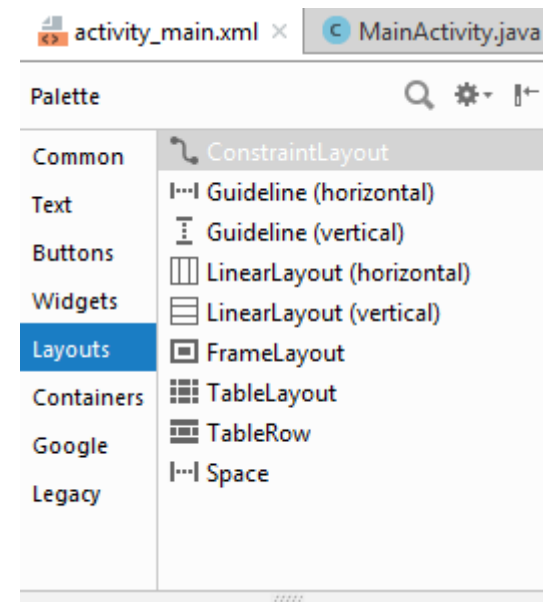
- ▶ Pour émettre un appel depuis votre application, il ne suffira pas uniquement de créer un objet *Intent* avec l'action *ACTION_CALL*. Il vous faudra également ajouter la permission *android.permission.CALL_PHONE* :
- ▶ Spécifier les permissions de l'application

Layouts prédéfinis

- ▶ Positions relatives
- ▶ Lignes ou colonnes
- ▶ Tableaux et « Grid »
- ▶ + autres plus ou moins utilisés

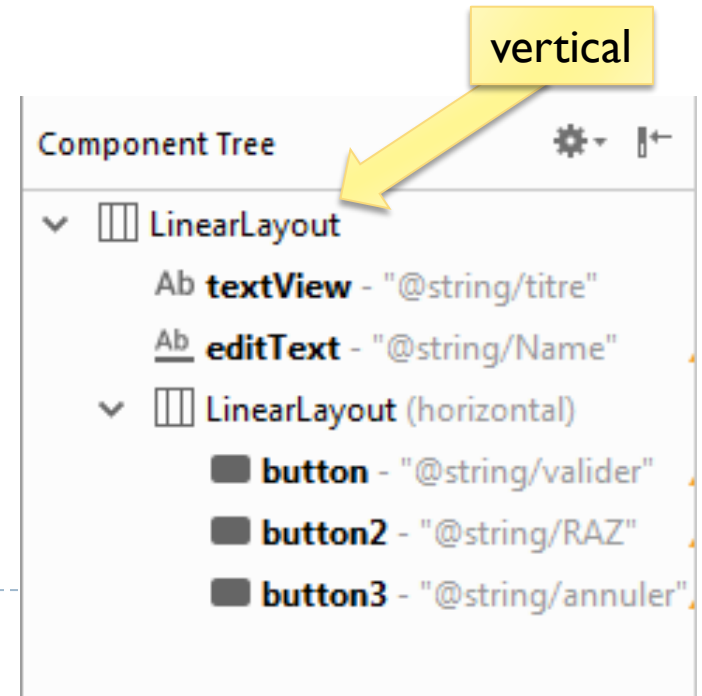
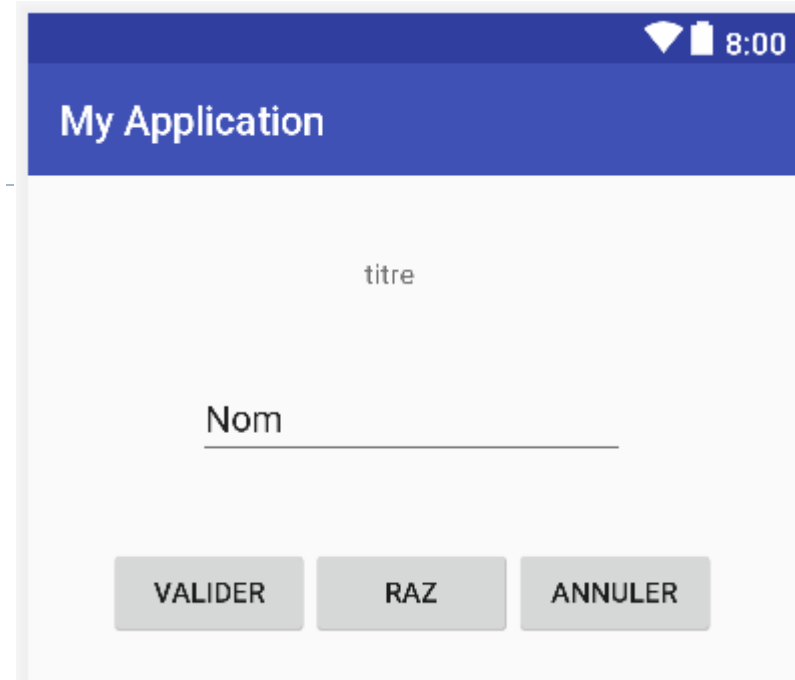
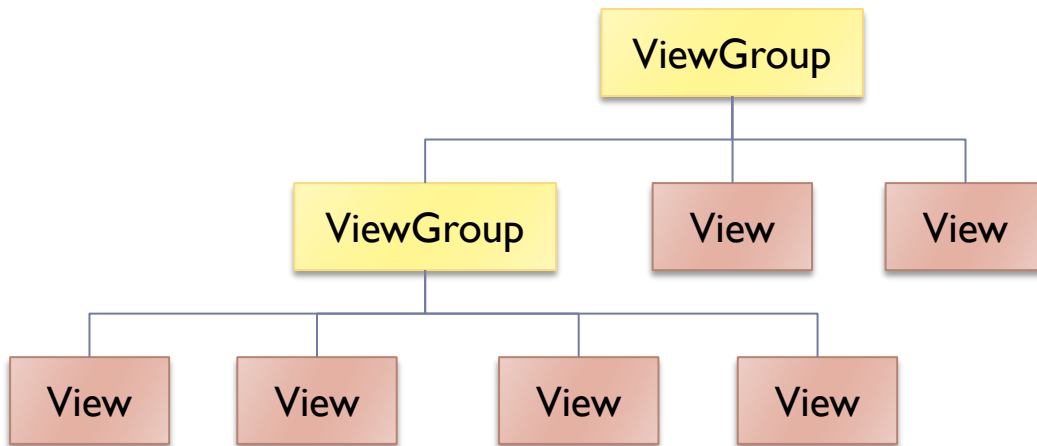
Structure d'une interface Android

- ▶ Un écran Android de type formulaire est généralement composé de plusieurs vues. Entre autres :
 - ▶ **TextView, ImageView** titre, image
 - ▶ **EditText** texte à saisir
 - ▶ **Button, CheckBox** bouton à cliquer, case à cocher
- ▶ Ces vues sont alignées à l'aide de **groupes** sous-classes de type **ViewGroup**, éventuellement imbriqués :
 - ▶ **LinearLayout** positionne ses vues en ligne ou colonne
 - ▶ **RelativeLayout** positionne ses vues l'une par rapport à l'autre
 - ▶ **TableLayout** positionne ses vues sous forme d'un tableau



Composants UI

- ▶ Éléments d'interface (UI) construits à base d'objets View et ViewGroup
- ▶ View = un élément (zone de texte, bouton, etc)
- ▶ Hiérarchie sous forme d'arbre



Représentation en XML

- ▶ Cet arbre s'écrit en XML :

```
<?xml version="1.0" encoding="utf-8"?>
<android.widget.LinearLayout android:id="@+id/groupe1" ... />

    <TextView android:id="@+id/textView" ... />
    <EditText android:id="@+id/editText" ... />

    <LinearLayout android:id="@+id/groupe2" ... />
        <Button android:id="@+id/button" ... />
        <Button android:id="@+id/button2" ... />
        <Button android:id="@+id/button3" ... />
    </LinearLayout>

</android.widget.LinearLayout>
```

Paramètres généraux

- ▶ Toutes les vues doivent spécifier ces deux attributs :
 - ▶ `android:layout_width` largeur de la vue
 - ▶ `android:layout_height` hauteur de la vue
- ▶ Ils peuvent valoir :
 - ▶ `"wrap_content"` : la vue est la plus petite possible
 - ▶ `"match_parent"` : la vue est la plus grande possible
 - ▶ `"valeur"` : une taille fixe, en dp, px, mm, in, ...
- ▶ **Attention** : changement dans Android 2.2 : `match_parent` si API ≥ 8 , `fill_parents` si API < 8

Unité de mesure

- ▶ Google préconise l'utilisation de l'unité **dp**
 - ▶ Appelé aussi **dip** (density independent pixel)
- ▶ Il existe aussi une métrique pour les polices de caractères
 - ▶ sp (our sip pour scale independent pixel)
- ▶ Il existe aussi :
 - ▶ px pour le pixel
 - ▶ in pour inch
 - ▶ mm pour millimètre
- ▶ Le dp est une unité de taille indépendante de l'écran. 100dp font 100 pixels sur un écran de 100 dpi (100 dots per inch) tandis qu'ils font 200 pixels sur un écran 200dpi. Ça fait la même taille apparente.

Bonnes pratiques

- ▶ UI en XML => séparation UI (l'apparence de l'application) et programmes Java (la logique de l'application)
- ▶ Unités : utiliser *Density-independent Pixels* (dp ou dip), pas de pixel
- ▶ Optimisation des interfaces : arbre peu profond
- ▶ Mieux vaut beaucoup de fils et une profondeur faible
- ▶ Un arbre profond augmente le temps de calcul pour le rendu du layout

Les Layouts

- ▶ Divers propriétés sont disponibles sur les layouts pour paramétrer l'affichage
 - ▶ voir en fonction des layouts
- ▶ Il faut penser en terme :
 - ▶ d'orientation
 - ▶ modèle de remplissage
 - ▶ poids
 - ▶ gravité
 - ▶ remplissage

Les Layouts

- ▶ Orientation ([android:orientation](#))
 - ▶ des layouts, comme [LinearLayout](#), présentent les widgets en ligne ou en colonne
 - ▶ l'orientation peut-être modifiée en cours d'exécution avec [setOrientation\(...\)](#)
- ▶ Poids ([android:layout_weight](#))
 - ▶ comment deux widgets se partagent l'espace disponible ?
 - ▶ [layout_weight](#) indique la proportion par widget
 - ▶ si 1 pour deux widgets l'espace libre est divisé en 2
 - ▶ si 1 pour un widget, et 2 pour le second, le second utilisera deux fois moins d'espaces que le premier

Les Layouts

▶ Gravité

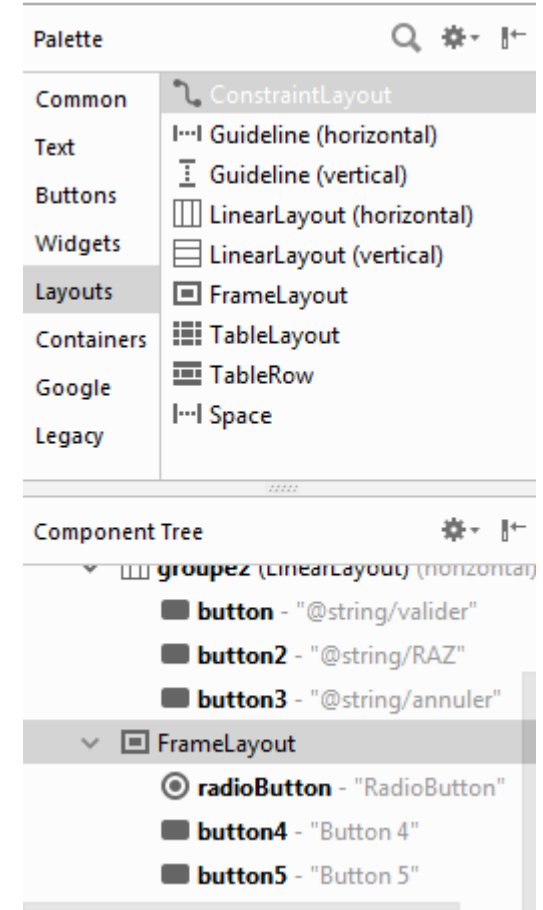
- ▶ `layout_gravity` gère le placement du widget dans le layout
 - ▶ indique au conteneur et au widget comment l'alignement doit être effectué
- ▶ `gravity` gère le placement du texte dans le widget

▶ Remplissage

- ▶ les widgets sont par défaut serrés les uns contre les autres
 - ▶ `padding` gère les 4 zones
 - ▶ `paddingTop`, `paddingBottom`, `paddingLeft`, `paddingRight` gèrent le remplissage zone par zone

Les Layouts – FrameLayout (1/2)

- ▶ Le plus simple des conteneurs
 - ▶ Tous les éléments sont placés les uns au-dessus des autres à partir du coin haut-gauche
 - ▶ Le dernier élément graphique ajouté vient recouvrir les autres éléments



Les layouts – FrameLayout

(2/2)

```
<FrameLayout
```

```
    android:layout_width="match_parent"  
    android:layout_height="match_parent">
```

```
    <RadioButton
```

```
        android:id="@+id/radioButton"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_gravity="right|center_vertical"  
        android:text="RadioButton" />
```

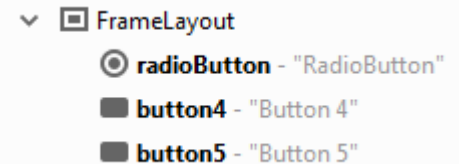
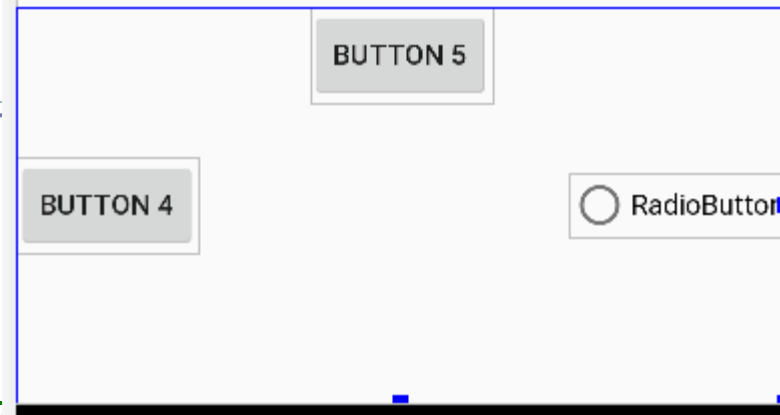
```
    <Button
```

```
        android:id="@+id/button4"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_gravity="left|center"  
        android:text="Button 4" />
```

```
    <Button
```

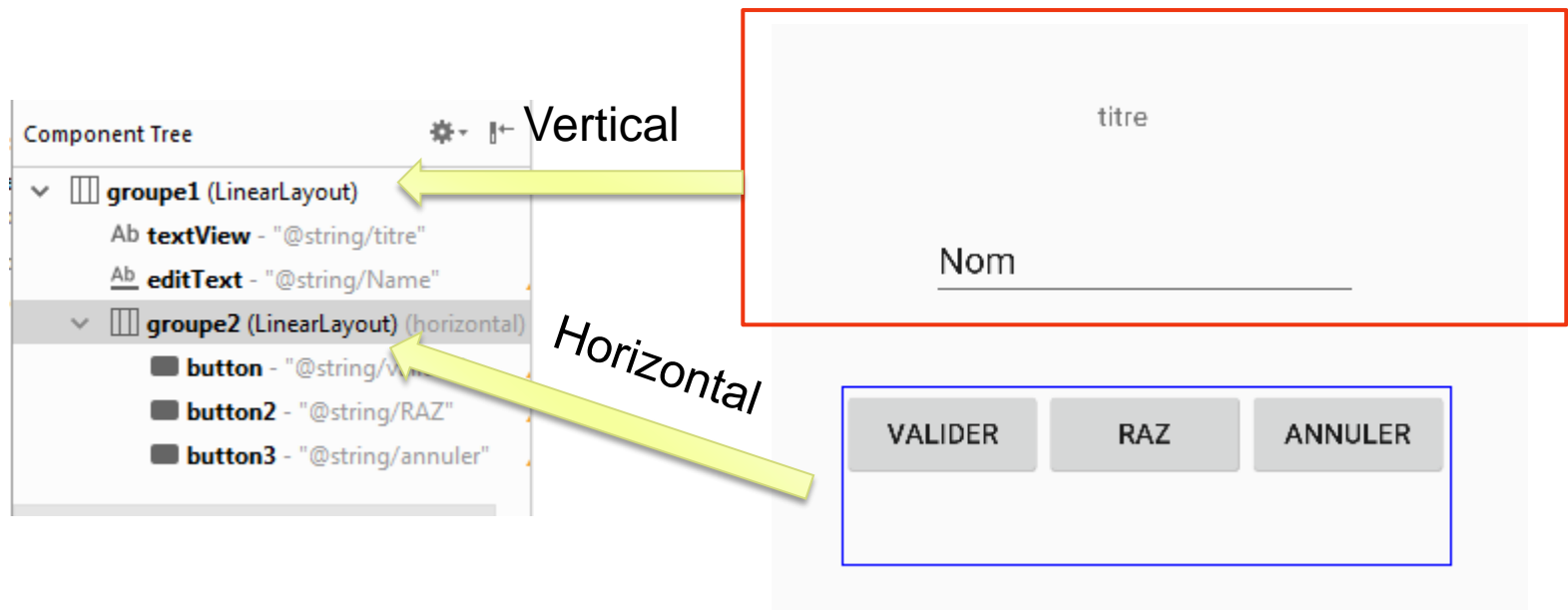
```
        android:id="@+id/button5"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_gravity="center_horizontal"  
        android:text="Button 5" />
```

```
</FrameLayout>
```



Les Layouts - LinearLayout

- ▶ Les composants sont placés les uns après les autres, selon l'attribut d'orientation
 - ▶ Verticalement : les uns sous les autres
 - ▶ Horizontalement : les uns après les autres, à la droite du précédent.



```
<?xml version="1.0" encoding="utf-8"?>
```

```
<android.widget.LinearLayout
```

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

```
xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
xmlns:tools="http://schemas.android.com/tools"
```

```
android:id="@+id/groupe1"
```

```
android:layout_width="match_parent"
```

```
android:layout_height="match_parent"
```

```
android:gravity="center_horizontal"
```

```
android:orientation="vertical" tools:context=".MainActivity">
```

```
<TextView
```

```
    android:id="@+id/textView"
```

```
    android:layout_width="48dp"
```

```
    android:layout_height="wrap_content"
```

```
    android:layout_alignParentTop="true"
```

```
    android:layout_centerHorizontal="true"
```

```
    android:layout_marginTop="40dp"
```

```
    android:text="@string/titre" />
```

```
<EditText
```

```
    android:id="@+id/editText"
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:layout_alignParentTop="true"
```

```
    android:layout_centerHorizontal="true"
```

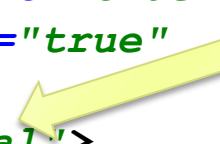
```
    android:layout_marginTop="40dp"
```

```
    android:inputType="textPersonName"
```

```
    android:text="@string/Name" />
```

```
<LinearLayout
```

```
    android:id="@+id/groupe2"  
    android:layout_width="match_parent"  
    android:layout_height="89dp"  
    android:layout_alignParentStart="true"  
    android:layout_alignParentTop="true"  
    android:layout_margin="40dp"  
    android:orientation="horizontal">
```

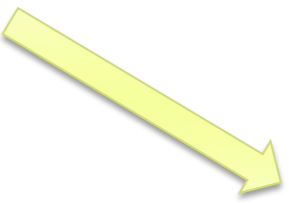


```
<Button
```

```
    android:id="@+id/button"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_weight="1"  
    android:text="@string/valider" />
```

```
<Button
```

```
    android:id="@+id/button2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_weight="1"  
    android:text="@string/RAZ" />
```



```
<Button
```

```
    android:id="@+id/button3"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_weight="1"  
    android:text="@string/annuler" />
```

```
</LinearLayout> </android.widget.LinearLayout>
```

Les Layouts - RelativeLayout

- ▶ C'est le plus complexe à utiliser mais il donne de bons résultats. Il permet de spécifier la position relative de chaque vue à l'aide de paramètres complexes : (**LayoutParams**)
- ▶ Tel bord aligné sur le bord du parent ou centré dans son parent :
 - ▶ **layout_alignParentTop** : haut du widget aligné avec celui du parent
 - ▶ **layout_alignCenterVertical** : centré verticalement par rapport au parent
- ▶ Tel bord aligné sur le bord opposé d'une autre vue :
 - ▶ android:**layout_toRightOf**, android:**layout_above**, android:**layout_below**...
- ▶ Tel bord aligné sur le même bord d'une autre vue :
 - ▶ android:**layout_alignLeft**,
 - ▶ android:**layout_alignTop**...

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"    android:layout_height="match_parent"
    android:paddingLeft="16dp"    android:paddingRight="16dp" >
```

```
<EditText
```

```
    android:id="@+id/name"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="@string/reminder" />
```

```
<Spinner
```

```
    android:id="@+id/dates"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/name"
    android:layout_alignParentLeft="true"
    android:layout_toLeftOf="@+id/times" />
```

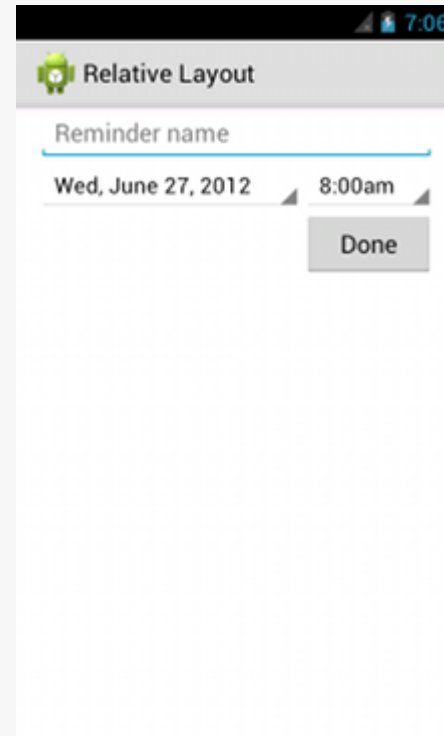
```
<Spinner
```

```
    android:id="@id/times"
    android:layout_width="96dp"
    android:layout_height="wrap_content"
    android:layout_below="@id/name"
    android:layout_alignParentRight="true" />
```

```
<Button
```

```
    android:layout_width="96dp"
    android:layout_height="wrap_content"
    android:layout_below="@id/times"
    android:layout_alignParentRight="true"
    android:text="@string/done" />
```

```
</RelativeLayout>
```

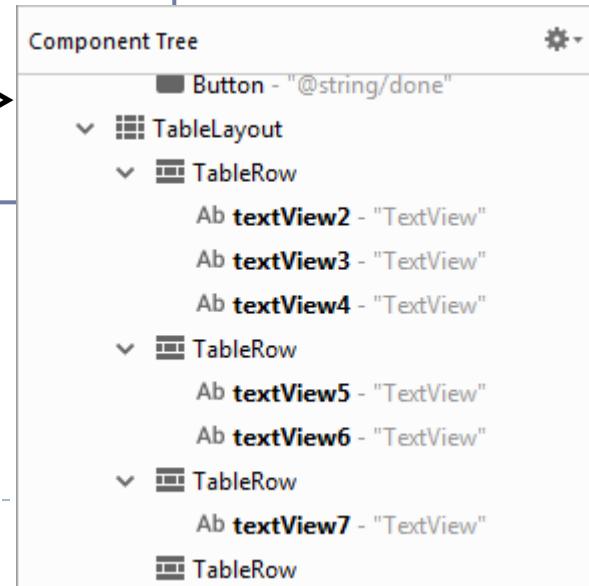


TableLayout

- ▶ Dérive de LinearLayout
- ▶ Éléments TableLayout, TableRow

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</TableLayout>
```

TextView	TextView	TextView	
TextView	TextView		
TextView			



GridLayout

- ▶ Similaire à `TableLayout` mais plus simple car inutile de spécifier toutes les “cellules”
- ▶ Intéressant en termes de performances
- ▶ Attribut `columnCount` pour spécifier le nombre de colonnes (donc attribut `rowCount` à priori inutile)
- ▶ Contenu : `layout_column` et `layout_row` pour spécifier la cellule à remplir `layout_rowSpan` et `layout_columnSpan` (“span” du contenu)
- ▶ `gravity` à utiliser, `weight` non utilisé par `GridLayout`



Composants

TextView

- Le plus simple, il affiche un texte statique, comme un titre. Son libellé est dans l'attribut android:text.

```
<TextView  
android:id="@+id/tvtitre"  
android:text="@string/titre"  
... />
```

- On peut le changer dynamiquement :

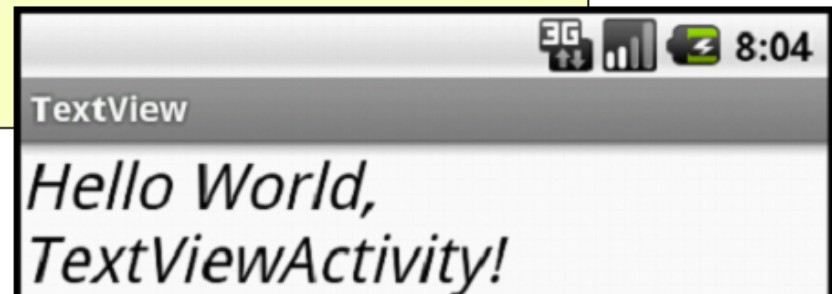
```
TextView tvTitre = (TextView) findViewById(R.id.tvtitre);  
tvTitre.setText("blablabla");
```

TextView

- Affichage d'un texte
 - Principales propriétés
 - `autoLink` : convertit les liens HTTP et adresses mails en liens cliquables
 - `ellipsize` : règle d'affichage du texte si plus long que la zone de visualisation
 - `height`, `width`, `maxHeight`, `maxWidth` : hauteur et largeur
 - `lines`, `minLines`, `maxLines` : nombre de lignes
 - `text` : texte à afficher
 - `textColor`, `textSize`, `textStyle` : caractéristiques du texte
 - `gravity` : endroit où est afficher le texte dans le conteneur si la zone d'affichage du texte est plus petite que le conteneur
 - `drawableTop`, `drawableBottom`, `drawableLeft`, `drawableRight`: permet la gestion de l'affichage d'une ressource *drawable* à côté du texte

TextView: exemple

```
<?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:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:background="#FFF"
        android:textColor="#000"
        android:textSize="25sp"
        android:textStyle="italic"
        android:text="@string/hello" />
</LinearLayout>
```



EditText

- Un **EditText** permet de saisir un texte,
- Hérite de TextView
 - même propriétés

```
<EditText  
android:id="@+id/email_address"  
android:inputType="textEmailAddress"  
... />
```

- Principales propriétés supplémentaires
- L'attribut `android:inputType` spécifie le type de texte :
adresse, téléphone, etc. ça définit le clavier qui est proposé
pour la saisie.
- `scrollHorizontally` : défilement horizontal du texte
 - La méthode `getText()` permet de récupérer le
texte saisi

EditText

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

...

<EditText
    android:id="@+id/nom"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="textPersonName" />

...

<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="textPassword" >

</EditText>


...

<EditText
    android:id="@+id/email"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="textEmailAddress" >

</EditText>

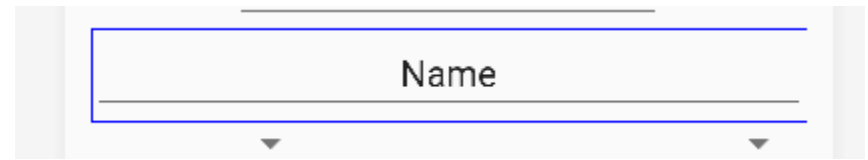
</LinearLayout>
```

Listing incomplet



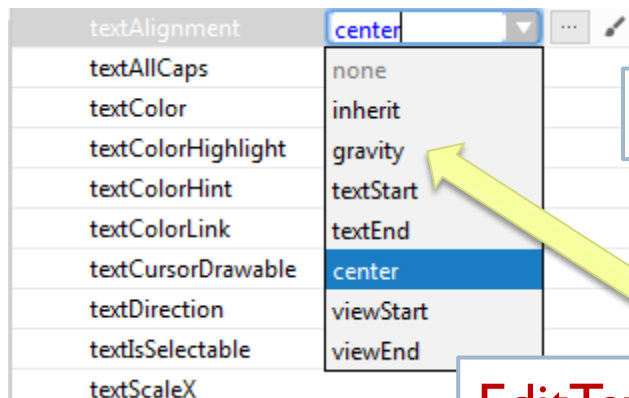
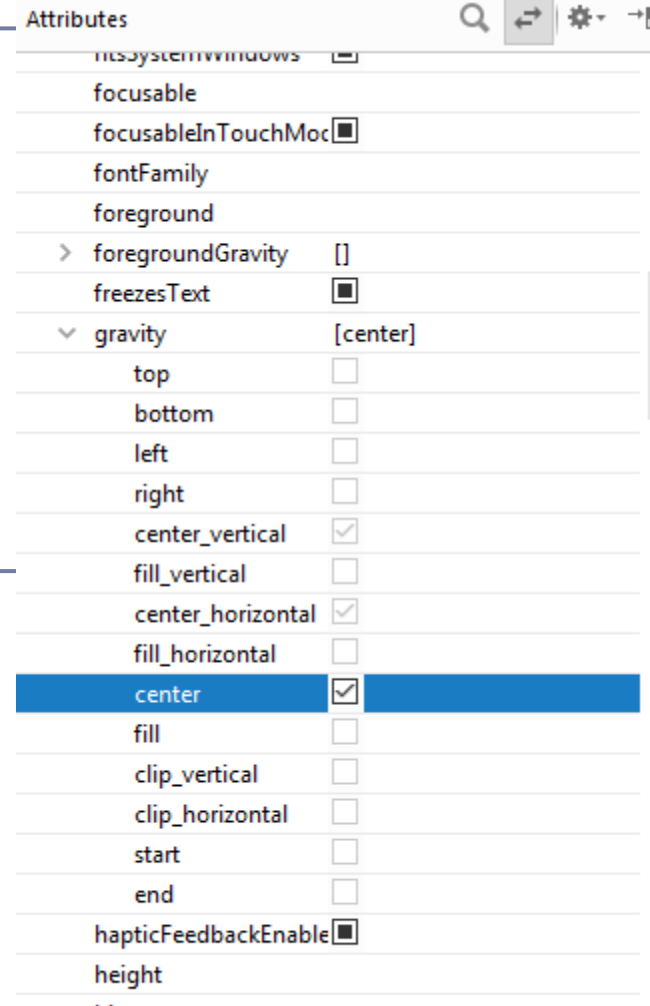
The screenshot shows a mobile application interface with a black background. At the top, there is a status bar with icons for 'AB', '3G', signal strength, battery, and the time '8:27'. Below the status bar, the title 'EditText' is displayed in a grey header. The main content area contains three input fields, each with a label to its left: 'Nom :' followed by a white text field with an orange border; 'Mot de passe' followed by a white password field; and 'Email :' followed by a white email address field.

EditText : gravity



<EditText

```
android:id="@+id/name"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:ems="10"
android:gravity="right"
android:inputType="textPersonName"
android:text="Name"
android:textAlignment="center" />
```

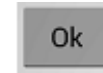


$gravity \simeq text-align$

EditText prend en considération l'attribut *gravity* une fois l'attribut *text-align* prend la valeur « gravity »

Button (1 / 6)

- L'une des vues les plus utiles est le Button:



```
<Button  
android:id="@+id/btn_ok"  
android:text="@string/ok"  
... />
```

- En général, on définit un identifiant pour chaque vue active, ici :
android:id="@+id/btn_ok"
- Son titre est dans l'attribut android:text,
- Hérite de TextView
- propriétés utiles
 - onClick : nom de la méthode de l'activité à exécuter
 - la signature de la méthode doit être:
 - public void nomMethode(View vue);

Button (2/6) : Méthode 1

- ▶ On peut assigner la méthode du bouton de Layout XML, utilisant l'attribut `android:onClick` :

```
<Button
    android:id="@+id/btnDisplay"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="bValider"
    android:text="@string/btn_display" />
```

- ▶ Quand l'utilisateur clique sur le bouton, le système Android appelle la méthode `bValider(View)`. La méthode doit être publique et accepte `View` comme paramètre unique.

```
nom = (TextView) findViewById(R.id.nom);
btnValider = (Button) findViewById(R.id.btnValider);

bValider((Button) btnValider);

}

public void bValider(View view) {
    Toast.makeText(getApplicationContext(), text: "Bonjour " + nom.getText(), Toast.LENGTH_SHORT).show();
}
```

A mon avis : Ne pas mélanger la logique de l'application avec l'apparence de l'application

Button (3/6) : Résultat

Khamlichi

VALIDER

Bonjour Khamlichi

Rendu du Toast

Button (4/6) : Méthode 2

écoutateur privé anonyme

► Exécution de l'action par utilisation d'un listener

```
<Button  
    android:id="@+id/btnValider"  
    ...  
    android:text="@string/valider" />
```

Il s'agit d'une classe qui est définie à la volée, lors de l'appel à `setOnClickListener`. Elle ne contient qu'une seule méthode.

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    Button b = (Button) findViewById(R.id.btnDisplay);  
    b.setOnClickListener(new View.OnClickListener() {  
        @Override  
        public void onClick(View v) {  
            EditText nom = (EditText) findViewById(R.id.nom);  
            Toast.makeText(context MainActivity.this, text: "Bonjour " + nom.getText(), Toast.LENGTH_LONG).show();  
        }  
    });  
}
```

Button (5/6) : Méthode 3

écoutateur privé

Cela consiste à définir une classe privée dans l'activité ; cette classe implémente l'interface OnClickListener ; et à en fournir une instance en tant qu'écouteur.

```
1 package ma.ensaf.myapplication;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.view.View;
6 import android.widget.Button;
7 import android.widget.TextView;
8 import android.widget.Toast;
9 public class MainActivity extends AppCompatActivity implements View.OnClickListener {
10     private Button btnValider;
11     private TextView nom;
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_main);
16
17         nom = (TextView) findViewById(R.id.nom);
18         btnValider = (Button) findViewById(R.id.btnValider);
19
20         btnValider.setOnClickListener(new btnValider() {
21             @Override
22             public void onClick(View v) {
23                 Toast.makeText(getApplicationContext(), text: "Bonjour " + nom.getText(), Toast.LENGTH_SHORT).show();
24             }
25         });
26     }
27 }
28
```

Button (6/6) : Méthode 3

L'activité elle-même en tant qu'écouteur

Il suffit de mentionner **this** comme écouteur et d'indiquer qu'elle implémente l'interface `OnClickListener`

```
1 package ma.ensaf.myapplication;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.view.View;
6 import android.widget.Button;
7 import android.widget.TextView;
8 import android.widget.Toast;
9 public class MainActivity extends AppCompatActivity implements View.OnClickListener {
10     private Button btnValider;
11     private TextView nom;
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_main);
16
17         nom = (TextView) findViewById(R.id.nom);
18         btnValider = (Button) findViewById(R.id.btnValider);
19
20         btnValider.setOnClickListener(this);
21     }
22
23     @Override
24     public void onClick(View v) {
25         Toast.makeText(getApplicationContext(), text: "Bonjour " + nom.getText(), Toast.LENGTH_SHORT).show();
26     }
27 }
28
```

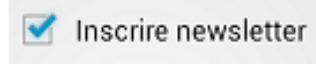
Distinction des émetteurs

- ▶ Dans le cas où le même écouteur est employé pour plusieurs vues, il faut les distinguer en se basant sur leur identifiant obtenu avec getId() :

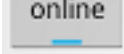
```
public void onClick(View v) {  
    switch (v.getId()) {  
        case R.id.btn_valider:  
            ...  
            break;  
        case R.id.btn_effacer:  
            ...  
            break;  
    }  
}
```


CheckBox (Bascules)

- Les **CheckBox** sont des cases à cocher :

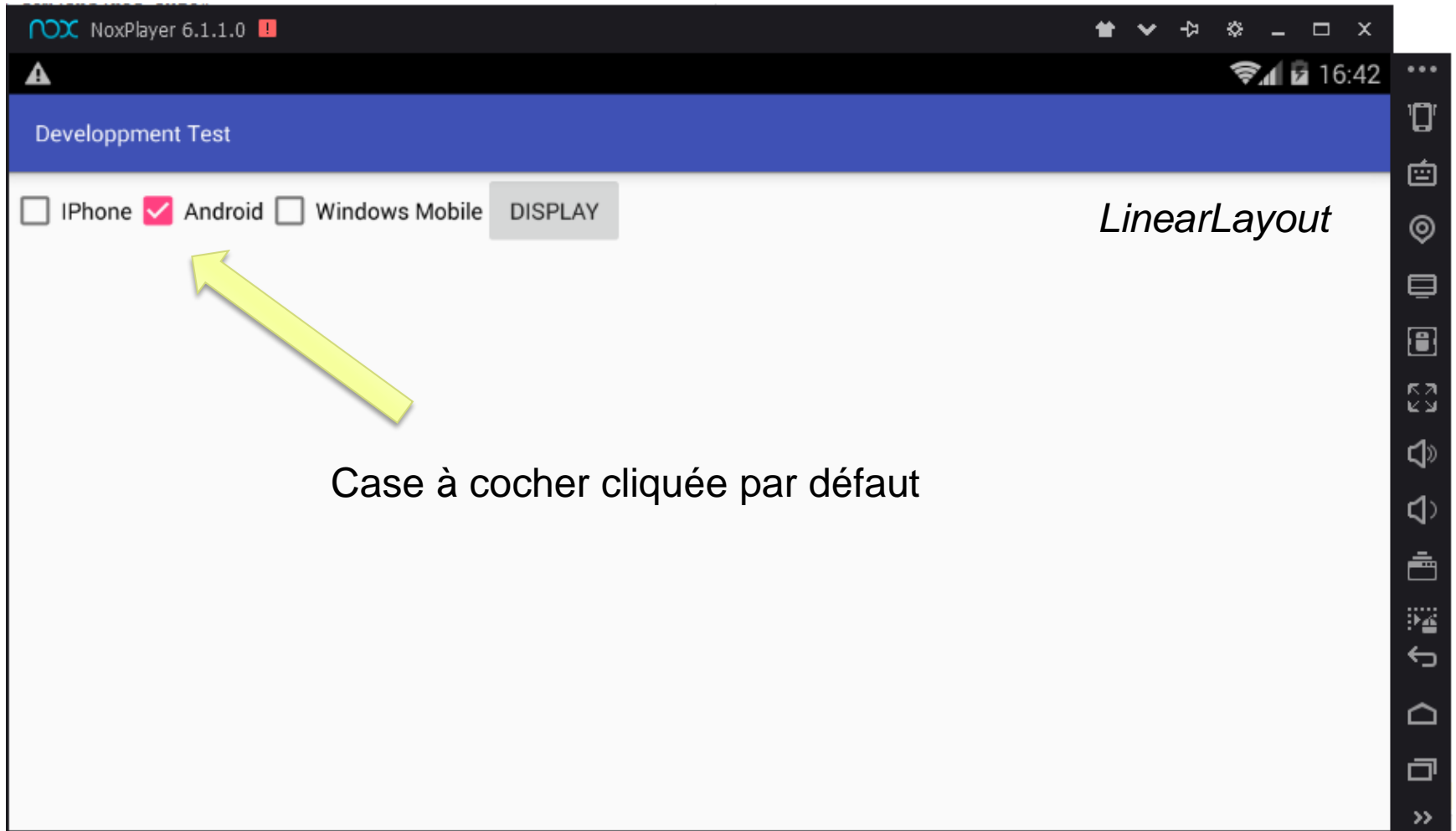


```
<CheckBox  
    android:id="@+id/cbx_abonnement_nl"  
    android:text="@string/abonnement_newsletter"  
... />
```

- Les **ToggleButton** sont une variante :  . On peut définir le texte actif et le texte inactif avec `android:textOn` et `android:textOff`.

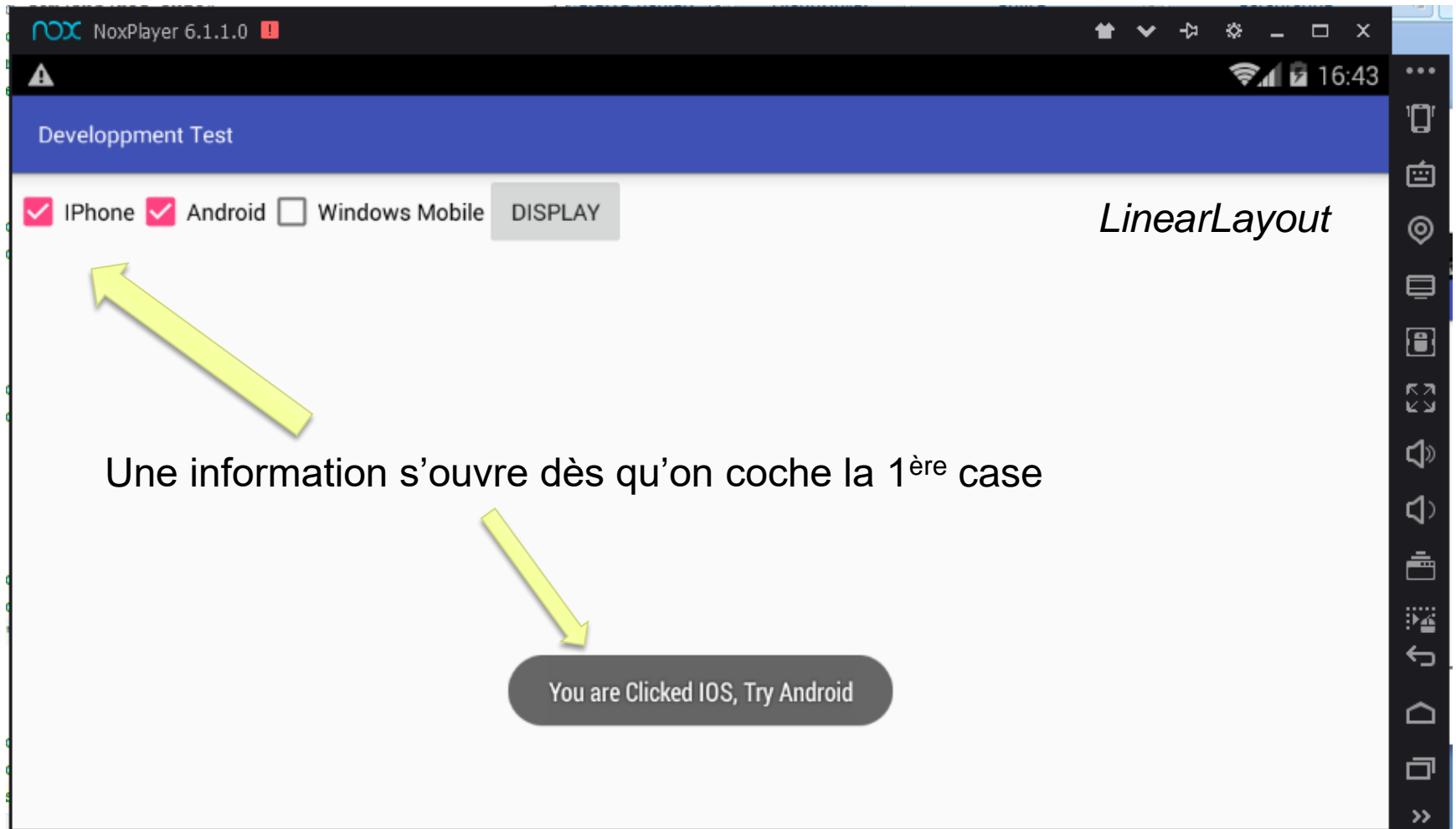
- Méthodes : `isChecked()`, `setChecked(bool)`, `toggle()`

Exemple CheckBox : (1 / 7)

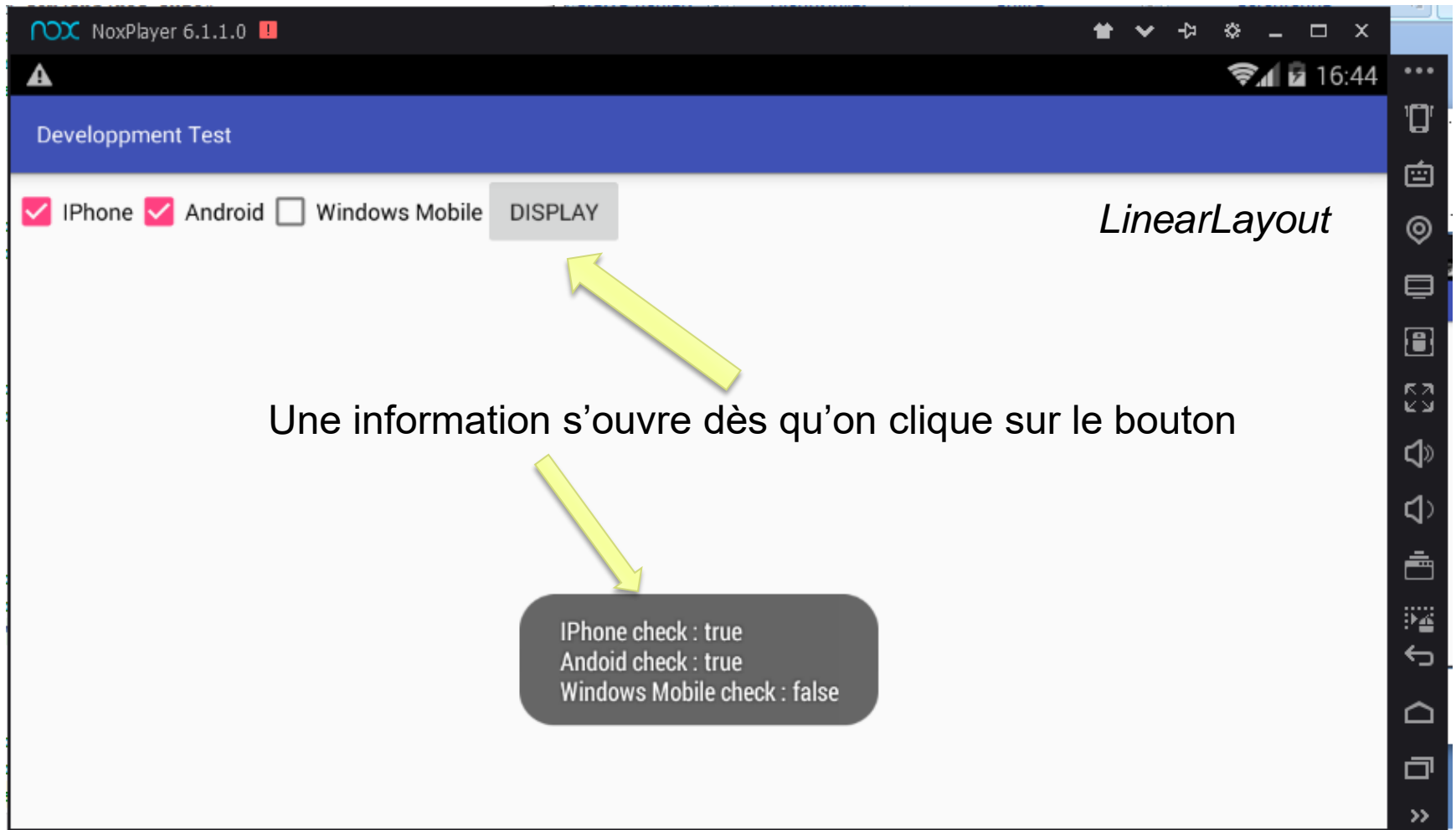


Case à cocher cliquée par défaut

Exemple CheckBox : (2/7)



Exemple CheckBox : (3/7)



Exemple CheckBox : (4/7) activity_main.xml

```
<LinearLayout ...  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".MainActivity">  
    <CheckBox  
        android:id="@+id/chkIos"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="@string/chk_ios" />  
    <CheckBox  
        android:id="@+id/chkAndroid"  
        android:text="@string/chk_android"  
        android:checked="true" />  
    <CheckBox  
        android:id="@+id/chkWindows"  
        android:text="@string/chk_windows" />  
    <Button  
        android:id="@+id/btnDisplay"  
        android:text="@string/btn_display" />  
</LinearLayout>
```




Exemple CheckBox : (5/7) strings.xml

```
<resources>
  <string name="hello">Hello World!</string>
  <string name="app_name">Developpment Test</string>
  <string name="chk_ios">IPhone</string>
  <string name="chk_android">Android</string>
  <string name="chk_windows">Windows Mobile</string>
  <string name="btn_display">Display</string>
</resources>
```

Exemple CheckBox : (6/7) MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    private CheckBox chkIos, chkAndroid, chkWindows;
    private Button btnDisplay;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        addListenerOnChkIOS();
        addListenerOnButton();
    }
}
```

```
public void addListenerOnChkIOS() {
    chkIos = (CheckBox) findViewById(R.id.chkIos);
    chkIos.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (((CheckBox) v).isChecked()) {
                Toast.makeText(MainActivity.this, "You are
Clicked IOS, Try Android", Toast.LENGTH_LONG).show();
            }
        }
    });
}
```



Exemple CheckBox : (7/7) MainActivity.java

```
public void addListenerOnButton() {
    chkIos = (CheckBox) findViewById(R.id.chkIos);
    chkAndroid = (CheckBox) findViewById(R.id.chkAndroid);
    chkWindows = (CheckBox) findViewById(R.id.chkWindows);
    btnDisplay = (Button) findViewById(R.id.btnDisplay);
    btnDisplay.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            StringBuffer result = new StringBuffer();
            result.append("IPhone check : ").append(chkIos.isChecked());
            result.append("\nAndoid check : ").append(chkAndroid.isChecked());
            result.append("\nWindows Mobile check :
        ").append(chkWindows.isChecked());

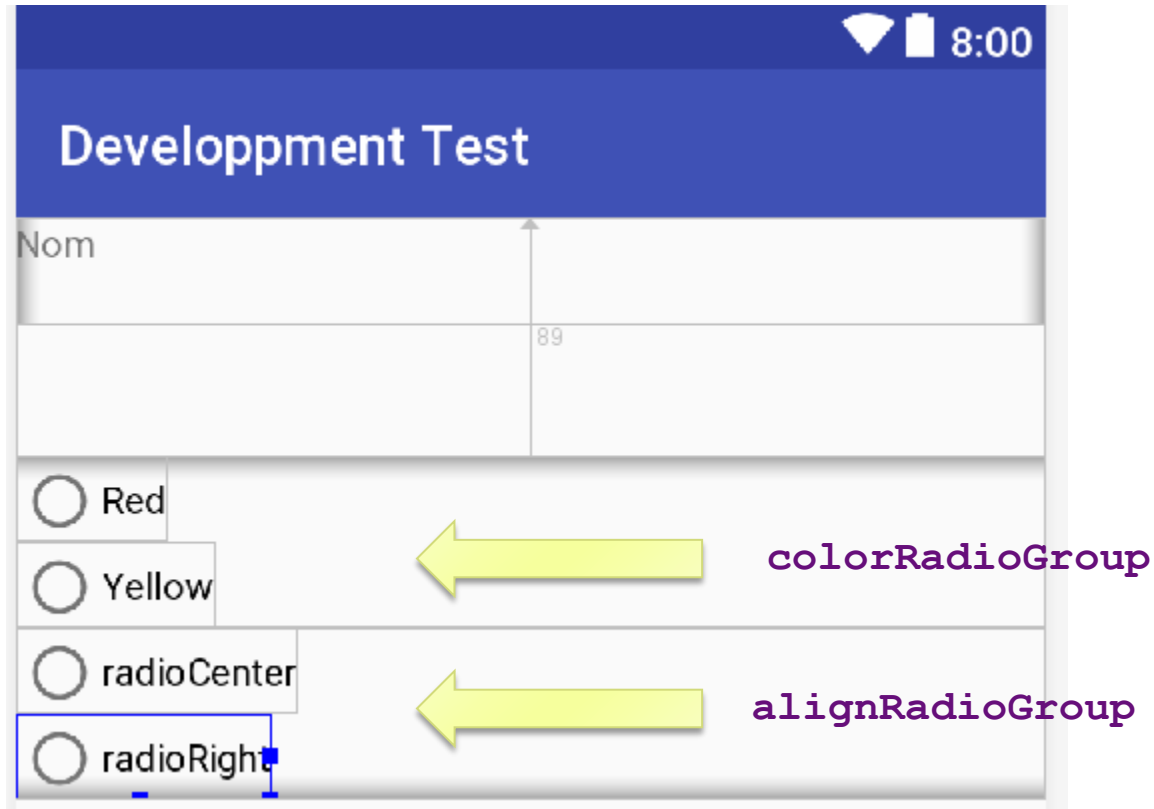
            Toast.makeText(MainActivity.this, result.toString(),
            Toast.LENGTH_LONG).show();
        }
    });
}
```


RadioButton

```
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:checkedButton="@+id/radioButton2">
    <RadioButton
        android:text="Mètre"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/radioButton"
        android:layout_weight="1" />
    <RadioButton
        android:text="Centimètre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/radioButton2"
        android:layout_weight="1" />
</RadioGroup>
```





RadioButton : Exemple (1 / 2)



RadioButton : Exemple (2/2)

```
public class MainActivity extends AppCompatActivity implements RadioGroup.OnCheckedChangeListener {
    RadioGroup alignRadioGroup, colorRadioGroup;
    TextView msg;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

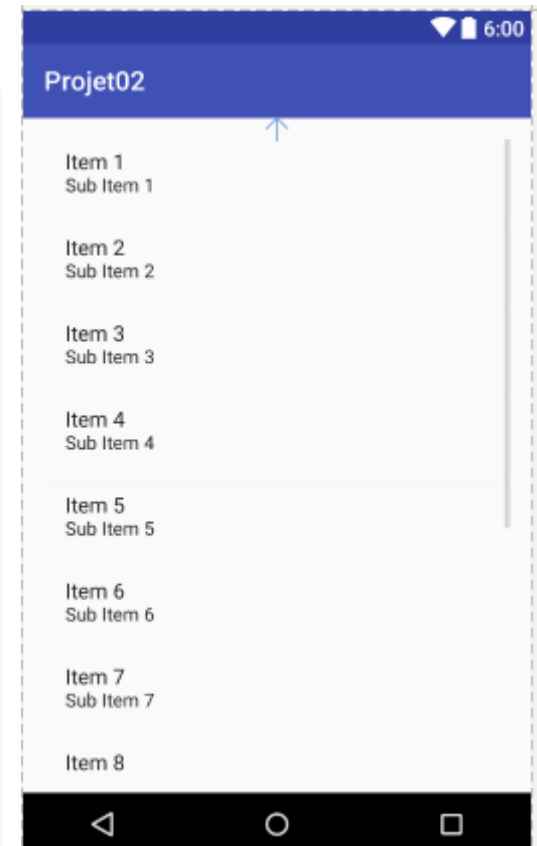
        colorRadioGroup = (RadioGroup) findViewById(R.id.colorRadioGroup);
        colorRadioGroup.setOnCheckedChangeListener(this);
        alignRadioGroup = (RadioGroup) findViewById(R.id.alignRadioGroup);
        alignRadioGroup.setOnCheckedChangeListener(this);
        msg = (TextView) findViewById(R.id.textView);
    }
    @Override
    public void onCheckedChanged(RadioGroup group, int checkedId) {
        int i = group.getCheckedRadioButtonId();
        switch (i)
        {
            case R.id.radioRed:
                msg.setBackgroundColor(Color.RED);
                break;
            case R.id.radioYellow:
                msg.setBackgroundColor(Color.YELLOW);
                break;
            case R.id.radioRight:
                msg.setGravity(Gravity.RIGHT);
                break;
            case R.id.radioCenter:
                msg.setGravity(Gravity.CENTER);
                break;
        }
    }
}
```



ListView

- L'objet ListView permet d'afficher des données sous forme de tableau.
- Etape 1 :
 - Ajouter la ListView dans l'activité en question.

```
<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >
    <ListView
        android:id="@+id/list"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:entries="@array/noms"
        android:scrollbars="horizontal|vertical">
    </ListView>
</RelativeLayout>
```



ListView

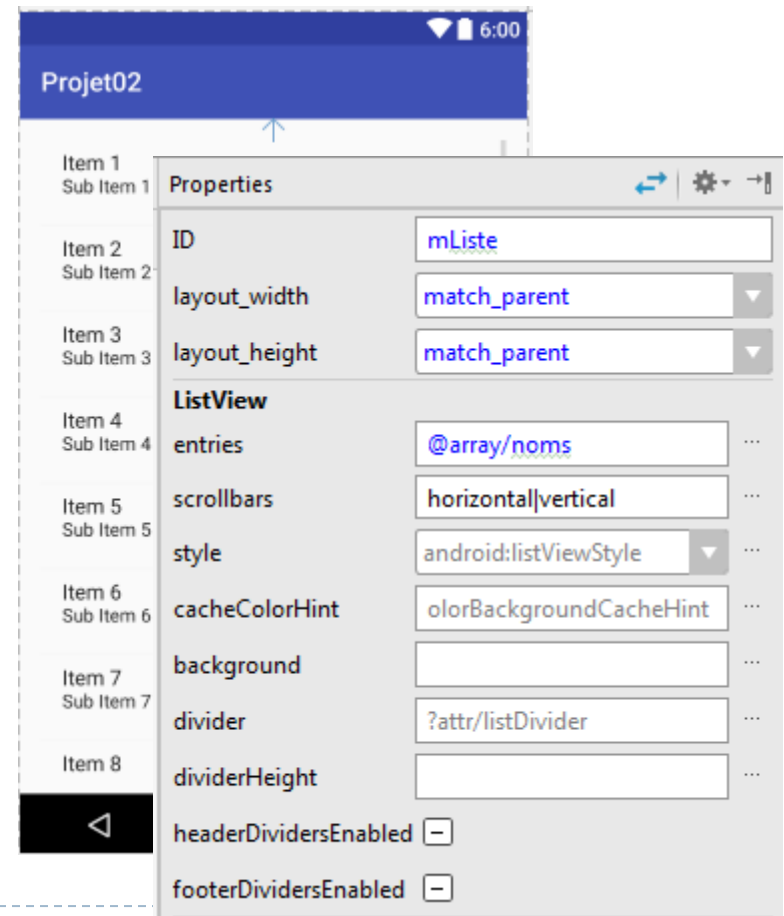
- Etape 2 :
 - Créer un tableau de données soit en fichier XML ou en Java, dans notre cas on utilise la méthode XML, en créant un fichier arrays.xml dans res/values

```
res/values/arrays.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="noms">
        <item>Etudiant 1</item>
        <item>Etudiant 2</item>
        <item>Etudiant 3</item>
        <item>Etudiant 4</item>
        <item>Etudiant 5</item>
    </string-array>
    <integer-array name="notes">
        <item>15</item>
        <item>6</item>
        <item>9</item>
        <item>12</item>
        <item>13</item>
    </integer-array>
</resources>
```

Listview

- Etape 3 :
 - Associer le tableau au ListView en ajoutant dans la propriété entries :
`@array/noms`

```
<RelativeLayout    <ListView
    android:id="@+id/list"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:entries="@array/noms"
    android:scrollbars="horizontal|vertical">
</ListView>
</RelativeLayout>
```





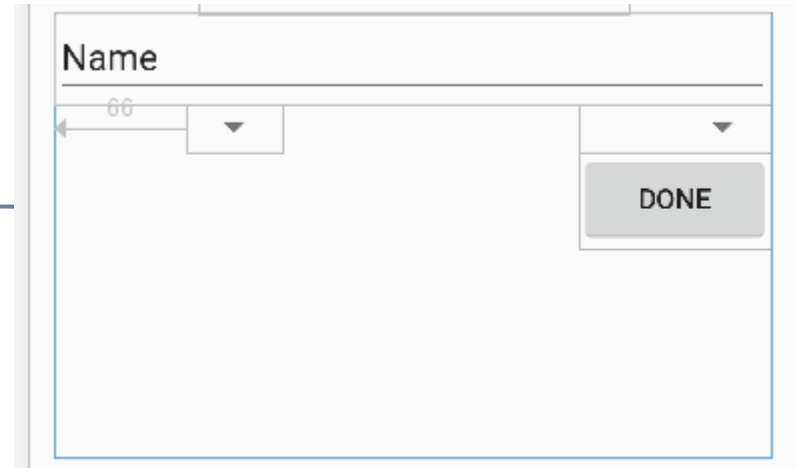
Containers - Spinner

Containers : Spinner

```
<RelativeLayout
    android:layout_width="358dp"
    android:layout_height="222dp">

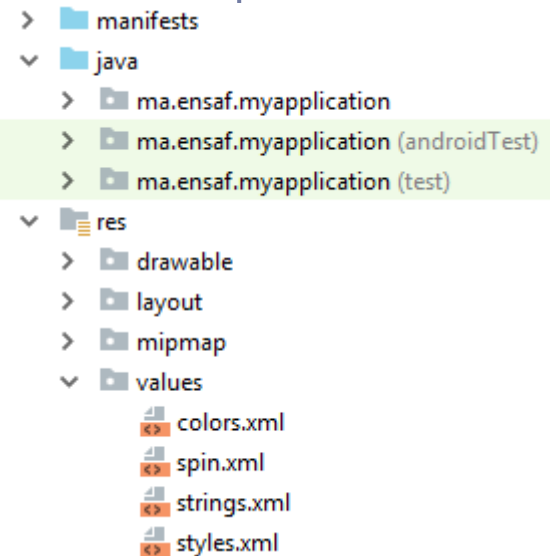
    <Spinner
        android:id="@+id/dates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/name"
        android:layout_marginStart="66dp" />

</RelativeLayout>
```



spin.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="week">
        <item>Lundi</item>
        <item>Mardi</item>
        <item>Mercredi</item>
        <item>Jeudi</item>
        <item>Vendredi</item>
        <item>Samedi</item>
        <item>Dimanche</item>
    </string-array>
</resources>
```



MainActivity.java

```
package ma.ensaf.myapplication;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.Spinner;

public class MainActivity extends AppCompatActivity {
    Spinner sp;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        sp = (Spinner) findViewById(R.id.dates);
        ArrayAdapter<CharSequence> ar =
        ArrayAdapter.createFromResource(this, R.array.week,
        android.R.layout.simple_list_item_1);
        ar.setDropDownViewResource(android.R.layout.simple_dropdown_item_1line);
        sp.setAdapter(ar);
    }
}
```