



android 

DEVELOPPEMENT MOBILE ANDROID

Element d' Interaction et Groupes

INSA BADJI Doctorant à l'Université de Thiès / Tuteur à



Séquence 2: Elément D'interaction et Groupes

- Objectif
- Plan
 - Elément Conteneurs et Widgets
 - Les labels
 - Les boutons
 - Les éléments à deux états



Propriété communes aux éléments d'interface (conteneurs et widgets)

- **Identifiant** : Un identifiant peut être associé à chaque élément décrit dans un fichier XML, cet identifiant permet d'accéder à l'objet créé dans le code ou de le référencer dans d'autres fichiers XML. Les éléments ne devant pas être référencés peuvent ne pas avoir d'identifiant.
- **android:id** = "@+id/mon_ident" permettra de retrouver cet élément par `findViewById(R.id.mon_ident)`.
- Méthode correspondante : **setId(int)**

Propriété communes aux éléments d'interface (conteneurs et widgets)

- **Visibilité**
 - `android:visibility` : Rend l'élément `visible`, `invisible` ou `absent` (avec `invisible` la place est conservée, avec `absent` la place n'est pas conservée).
- **Fond**
 - `android:background` couleur ou une image de fond
- **Taille**
 - `android:minHeight` et `android:minWidth` dimensions minimales
- **Placement des éléments contenus (défini pour chaque élément)**
 - `android:layout_height` et `android:layout_width` place prise par l'élément dans le conteneur :
 - `FILL_PARENT` remplit toute la place
 - `WRAP_CONTENT` occupe la place nécessaire
 - `android:layout_gravity` positionnement de l'élément dans le conteneur `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill`

Propriété communes aux éléments d'interface (conteneurs et widgets)

- ***Ascenseurs (s'il y en a)***
 - android:fadeScrollbars Pour choisir de faire disparaître ou pas les ascenseurs lorsqu'ils ne sont pas utilisés
 - android:scrollbarDefaultDelayBeforeFade Définit le délai avant que les ascenseurs non utilisés disparaissent
 - android:scrollbarFadeDuration Définit la durée d'effacement des ascenseurs
- ***Marges internes (défini pour chaque élément)***
 - android:layout_paddingBottom , android:layout_paddingLeft , android:layout_paddingRight , android:layout_paddingTop
- ***Marges externes (défini pour chaque élément)***
 - android:layout_marginBottom , android:layout_marginLeft , android:layout_marginRight , android:layout_marginTop

Propriété communes aux éléments d'interface (conteneurs et widgets)

- ***Prise en compte des événements***

- *Prise en compte des clics sur l'élément*

android:clickable Autorise ou interdit la prise en compte des clics

Méthode correspondante : `setClickable(boolean)`

- ***Prise en compte des clics longs sur l'élément***

android:longClickable **Autorise ou interdit la prise en compte des clics longs**

Méthode correspondante : `setLongClickable(boolean)`

On ajoutera ensuite un écouteur d'événements pour les traiter

Les labels de texte avec XML (dans activity_main.xml)

- TextView est normalement utilisé pour afficher un texte
- propriétés souvent utilisées:
 - **android:text**
 - **android:typeFace**
 - **android:textStyle (bold, italic or bold_italic)**

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
```


Les labels de texte avec Java

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    LinearLayout myLayout = new LinearLayout(this);  
    myLayout.setGravity(Gravity.CENTER); //Centrer les  
elements graphiques  
    myLayout.setOrientation(LinearLayout.VERTICAL); // empiler  
vers le bas  
    TextView texte = new TextView(this);  
    texte.setText("test de programmation des interfaces  
utilisateurs Android UVS");  
    texte.setTextColor(Color.CYAN);  
    texte.setBackgroundColor(Color.RED);  
    myLayout.addView(texte);  
    setContentView(R.layout.activity_main);  
}
```


Les zones de texte : EditText

- EditText est normalement utilisé pour saisir du texte
- Hérite de **TextView**
- Propriétés:
 - android:autoText
 - android:capitalize
 - android:digits
 - android:singleLine
 - android:numeric
 - android:phoneNumber
 - android:password
 - android:inputType («number», «phone»,...)
 - android:hint

Les zones de texte : EditText

- Avec XML

```
<EditText android:id="@+id/EditText01"  
          android:layout_width="fill_parent"  
          android:layout_height="wrap_content"  
          android:text="" >  
</EditText>
```

- Avec Java

```
EditText edit = new EditText(this);  
edit.setText("Edite moi s'il te plait ");  
myLayout.addView(edit);
```

Zones de texte: propriétés

- ***Taille et aspect du texte***

- android:textSize="unité" utiliser de préférence l'unité sp qui est liée aux polices
- android:textScaleX Pour définir l'échelle horizontale du texte
- android:textStyle="g" (où s peut être normal, bold, italic) ces styles peuvent être combinés par |
- android:typeface="s" (où s peut être normal, sans, serif, monospace)
- android:singleLine="b" (où b vaut true ou false) limite le texte à une seule ligne
- android:lines Pour définir le nombre de lignes du texte
- android:maxlines Pour définir le nombre maximal de lignes du texte
- android:minlines Pour définir le nombre minimal de lignes du texte
- android:lineSpacingExtra Pour définir l'espace supplémentaire entre les lignes
- android:scrollHorizontally Pour autoriser ou interdire le défilement horizontal du texte

Zones de texte: propriétés

- ***Comportement du texte***

- android:autoLink="a" (où a peut être : none, web, email, phone, map ou all) indique si les liens de ce type apparaissant dans le texte sont automatiquement rendus cliquables.
- android:autoText Pour valider ou pas le mode correction du texte
- android:capitalize="c" (où c peut être : none, sentences, words, characters) indique le type de saisies que le texte mémorise et peut re-proposer.
- android:digits Pour indiquer si la saisie n'accepte que du numérique ou pas
- android:numeric="x" (où x peut être integer, signed, decimal) définit le mode de saisie numérique
- android:password pour cacher ou pas le texte lors d la saisie
- android:phoneNumber Pour indiquer si la saisie n'accepte que des n0s de téléphone
- android:inputType Pour définir le mode de saisie (none, text, textCapCharacters, textCapWords, textCapSentences, textAutoCorrect, textAutoComplete, textMultiLine, textUri, textEmailAddress, textEmailSubject, textShortMessage, textLongMessage, textPersonName, textPostalAddress, textPassword, textVisiblePassword, textWebEditText, textFilter, textPhonetic, textWebEmailAddress, textWebPassword, number, numberDecimal, numberPassword, phone, datetime, date ou time)

Zones de texte: propriétés

- ***Affichage***

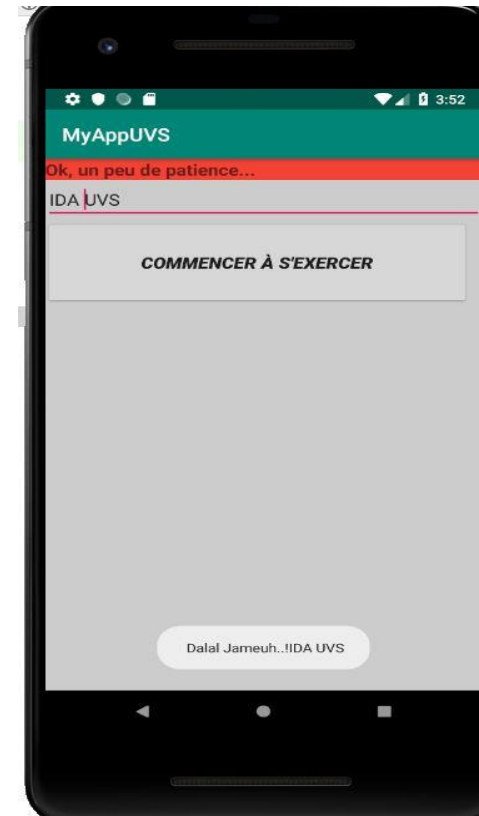
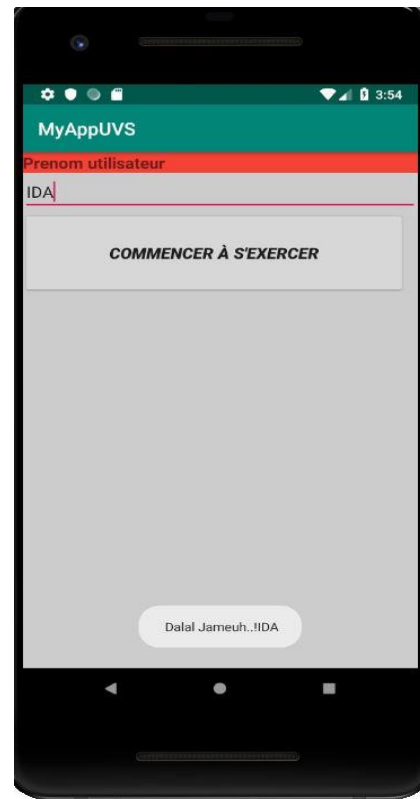
- android:cursorVisible Pour rendre visible ou non le curseur
- android:editable Pour autoriser ou pas la modification du texte
- android:ellipsize="e" (où e peut être : none, start, middle, end, marquee) définit le mode de césure du texte
- android:linksClickable Pour rendre ou pas les liens cliquables
- android:textIsSelectable pour autoriser/interdire la sélection de texte

- ***Couleurs et images***

- android:textColor Pour définir une couleur au texte
- android:textColorHighlight Pour définir une couleur de surlignage du texte
- android:textColorHint Pour définir une couleur au texte par défaut
- android:textColorLink Pour définir une couleur aux liens du texte
- android:drawableBottom Pour définir une couleur ou une image de fond au texte

Zones de texte: exemple

Dans ce petit exemple, nous utiliserons un `LinearLayout` détenant un `label` (`TextView`), un `textBox` (`EditText`) et un bouton. Nous allons utiliser la vue comme une sorte d'écran de connexion simplifiée.



Zones de texte: exemple

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout 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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#ffccccc"
        android:orientation="vertical"
        tools:layout_editor_absoluteX="0dp"
        tools:layout_editor_absoluteY="0dp">

        <TextView
            android:id="@+id/textView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="#F44336"
            android:text="Prenom utilisateur"
            android:textSize="18sp"
            android:textStyle="bold" />

    </LinearLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
```

```
<EditText
    android:id="@+id/txtUserName"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:autoText="true"
    android:capitalize="words"
    android:ems="10"
    android:hint="Prenom et nom"
    android:inputType="textPersonName"
    android:textStyle="normal" />

<Button
    android:id="@+id/buttonCommencer"
    android:layout_width="399dp"
    android:layout_height="97dp"
    android:text="Commencer à S'exercer"
    android:textSize="18sp"
    android:textStyle="bold|italic" />

</LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```


Zones de texte: exemple

```
package com.example.myappuvs;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    Button monBouton;
    EditText editText;
    TextView labelUsername;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        labelUsername = (TextView)findViewById(R.id.textView);
        editText = (EditText)findViewById(R.id.txtUserName);
        monBouton = (Button)findViewById(R.id.buttonCommencer);
        monBouton.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == monBouton.getId()) {
            String userName = editText.getText().toString();
            if (userName.compareTo("IDA UVS") == 0) {
                labelUsername.setText("Ok, un peu de patience...");
                Toast.makeText(getApplicationContext(), text: " Dalal Jameuh..!" + userName, Toast.LENGTH_LONG).show();
            }
            Toast.makeText(getApplicationContext(), text: " Dalal Jameuh..!" + userName, Toast.LENGTH_LONG).show();
        }
    }
}
```

Les boutons : **Button**

- Button : Mêmes paramètres que TextView
- ImageButton : Mêmes paramètres que ImageView càd :
 - android:src="couleur" pour définir une couleur ou une image
 - android:adjustViewBounds Pour indiquer si la taille du bouton doit ou pas être ajustée à celle de l'image
 - android:baselineAlignBottom Pour indiquer que l'image est placée ou pas en bas de la zone
 - android:cropToPadding Pour indiquer si l'image sera coupée ou pas si elle est plus grande que la taille disponible
 - android:scaleType="s" (où s peut prendre les valeurs : matrix, fitXY, fitStart, fitCenter, fitEnd, center, centerCrop, centerInside) permet de redimensionner ou pas l'image à la taille disponible et/ou de la déformer.
 - android:maxHeight Pour définir la hauteur disponible
 - android:maxWidth Pour définir la largeur disponible
 - android:tint Pour définir une couleur qui teinte l'image

Les boutons

<Button

```
    android:id="@+id/Button01"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Clique sur moi " >
```

</Button>

```
Button b = (Button) findViewById(R.id.Button01);  
b.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        Toast.makeText(v.getContext(), "Stop ", Toast.LENGTH_LONG).show();  
    }  
});  
myLayout.addView(b);
```




ImageButton

- Bouton normal, mais avec une image. Mêmes paramètres XML que ImageView
- Exemple de fichier XML :

```
<ImageButton android:id="@+id/boutonImage"
  android:src="@drawable/loupe"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"/>
```
- **Méthodes de la classe ImageButton**
 - android.widget.ImageButton
 - ImageButton hérite de ImageView il a donc les mêmes sauf le constructeur :
- ***Construction:*** ImageButton(Context) le paramètre est généralement l'activité elle-même

Les éléments à deux états

Ils ont les mêmes paramètres que TextView auxquels vient s'ajouter la définition de l'état initial : `android:checked="b"` où b vaut true ou false Pour définir l'état initial

- CheckBox  Cocher ici
- RadioButton  Ceci est un bouton radio
- ToggleButton 
 - `android:disabledAlpha` pour définir la transparence appliquée lorsque le bouton est inactif
 - `android:textOff` Pour définir le texte quand le bouton n'est pas allumé
 - `android:textOn` Pour définir le texte quand le bouton est allumé

Les CheckBox

- Hérite de **CompoundButton**
- Possible d'alterer la valeur depuis le code:
 - **isChecked()**
 - **setChecked()**
 - **toggle()**
 - Possible d'enregistrer un event listener **OnCheckedChangeListener**

```
<CheckBox  
    android:id="@+id/check"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="This checkbox is: unchecked" />
```

Exemple CheckBox

```
public class CheckBoxDemo extends Activity implements CompoundButton.OnCheckedChangeListener {
    CheckBox cb;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        cb=(CheckBox)findViewById(R.id.checkbox);
        cb.setOnCheckedChangeListener(this);
    }

    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            cb.setText("Ce checkbox est: coché");
        }
        else {
            cb.setText("Ce checkbox n'est pas coché");
        }
    }
}
```


RadioButton

- **RadioButton:** Mêmes paramètres XML que TextView plus android:checked="b" où b vaut true ou false coche ou décoche le bouton au départ
- Exemple de fichier XML :

```
<RadioButton android:id="@+id/gauche"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Left"
android:checked="false"
android:layout_gravity="left"
/>
```
- **Méthodes de la classe RadioButton:** android.widget.RadioButton
- **Construction:** RadioButton(Context) le paramètre est généralement l'activité elle-même
- **Etat**
 - isChecked() renvoie true si la case est cochée
 - setChecked(boolean) coche (true) ou décoche (false) la case
 - toggle() change l'état de la case

RadioButtons

- Une case d'option est un bouton de deux États qui peut être soit cochée ou décochée.
- Si la case d'option est désactivée, l'utilisateur peut appuyer sur ou cliquez dessus pour le vérifier.
- Cases d'option sont normalement utilisés ensemble dans un RadioGroup.
- Lorsque plusieurs cases d'option sont à l'intérieur d'un groupe de boutons radio, la sélection d'une case d'option désactive toutes les autres.
- De même, vous pouvez appeler `isChecked()` sur un `RadioButtonto` voir si elle est activée, `toggle()` pour le sélectionner et ainsi de suite, comme vous pouvez le faire avec une case à cocher.

RadioGroup

- Pour grouper des boutons radio (ajoute un ascenseur si nécessaire)
- Un seul bouton peut être coché à la fois (attention à l'état initial qui n'est pris en compte par le RadioGroup que s'il est fait par la méthode **check du RadioGroup**)
- Exemple de fichier XML :

```
<RadioGroup
  android:id="@+id/groupe_de_boutons"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:orientation="horizontal">
  <RadioButton
    ...
  />
  <RadioButton
    ...
  />
</RadioGroup>
```



Exemple RadioGroup

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout 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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#ffcccc"
        android:orientation="vertical"
        tools:layout_editor_absoluteX="0dp"
        tools:layout_editor_absoluteY="0dp">

        <RadioGroup
            android:layout_width="match_parent"
            android:layout_height="74dp"
            android:orientation="horizontal">

            <RadioButton
                android:id="@+id/radioButton"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="horizontal" />

            <RadioButton
                android:id="@+id/radioButton2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="vertical" />

        </RadioGroup>

        <RadioGroup
            android:layout_width="match_parent"
            android:layout_height="279dp"
            android:orientation="vertical">

            <RadioButton
                android:id="@+id/radioButton3"
                android:layout_width="164dp"
                android:layout_height="wrap_content"
                android:text="haut" />

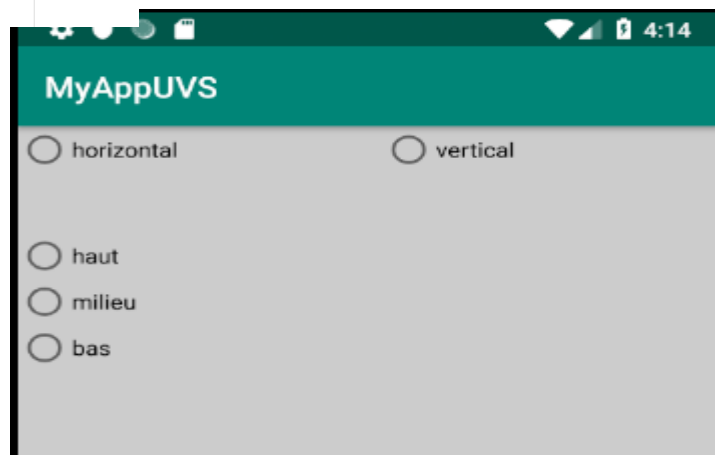
            <RadioButton
                android:id="@+id/radioButton4"
                android:layout_width="167dp"
                android:layout_height="wrap_content"
                android:text="milieu" />

            <RadioButton
                android:id="@+id/radioButton5"
                android:layout_width="166dp"
                android:layout_height="wrap_content"
                android:text="bas" />

        </RadioGroup>

    </LinearLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
```



```
<RadioButton
    android:id="@+id/radioButton2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="vertical" />
</RadioGroup>

<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="279dp"
    android:orientation="vertical">

    <RadioButton
        android:id="@+id/radioButton3"
        android:layout_width="164dp"
        android:layout_height="wrap_content"
        android:text="haut" />

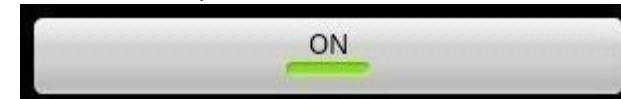
    <RadioButton
        android:id="@+id/radioButton4"
        android:layout_width="167dp"
        android:layout_height="wrap_content"
        android:text="milieu" />

    <RadioButton
        android:id="@+id/radioButton5"
        android:layout_width="166dp"
        android:layout_height="wrap_content"
        android:text="bas" />

    </RadioGroup>
</LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

ToggleButton

- Mêmes paramètres XML que TextView plus :
 - android:disabledAlpha="x" (où x est une valeur réelle entre 0 et 1) définit la transparence appliquée lorsque le bouton est inactif
 - android:textOff="txt" définit le texte quand le bouton n'est pas allumé
 - android:textOn="txt" définit le texte quand le bouton est allumé
 - android:checked="b" (où b vaut true ou false) place le bouton en position allumé ou éteint
- **Méthodes de la classe ToggleButton:** android.widget.ToggleButton
- **Construction:** ToggleButton(Context) le paramètre est généralement l'activité elle-même
- **Etat**
 - isChecked() indique si le bouton est allumé ou éteint (true/false)
 - setChecked(boolean) positionne le bouton (true = allumé, false = éteint)
 - toggle() change l'état du bouton
- **Aspect**
 - getTextOff() renvoie le texte associé au bouton quand il n'est pas allumé
 - getTextOn() renvoie le texte associé au bouton quand il est allumé
 - setTextOff(CharSequence) définit le texte associé au bouton quand il n'est pas allumé
 - setTextOn(CharSequence) définit le texte associé au bouton quand il est allumé



Les groupes

- **Regrouper des éléments participant à un choix:**
 - ListView (plusieurs éléments organisés en liste verticale avec séparateurs). Souvent utilisé pour des listes de mots (type menu).
 - GridView (plusieurs éléments organisés en table). Souvent utilisé pour des tables de mots (type menu).
 - RadioGroup (groupe de boutons radio dont un seul peut être coché à la fois)- Voir partie précédente
 - Gallery (plusieurs éléments organisées horizontalement avec défilement). Souvent utilisé pour des images

ListView

- Place les éléments en liste verticale et ajoute un ascenseur si nécessaire
 - **Séparateurs**
 - `android:divider` Pour définir la couleur des séparateurs ou pour utiliser une image comme séparateur.
 - `android:dividerHeight="unité"` Pour définir la hauteur des séparateurs (même s'ils contiennent une image)
 - **Type de choix**
 - `android:choiceMode="c"` (où c peut prendre les valeurs : `none`, `singlechoice`, `multipleChoice`) pour indiquer le mode de choix dans la liste (aucun, un seul, plusieurs).

ListView (contenu)

- ***En XML (texte seulement)***

- android:entries="@array/maliste" définit le contenu de la liste à partir du contenu d'un fichier xml placé dans res/values/ et qui a la forme :

- ***Dans le code (éléments quelconques)***

On utilise un gestionnaire de contenu (Adapter)

- setAdapter(Adapter) pour associer à la ListView
- ***Soit de classe prédéfinie (ArrayAdapter , SimpleAdapter, CursorAdapter)***
 - ArrayAdapter(Context, type) le second paramètre est une type prédéfini :
 - android.R.layout.simple_list_item_1 pour une liste à choix unique ou
 - android.R.layout.simple_list_item_multiple_choice pour une liste à choix multiple (une case à cocher apparaît à côté de chaque élément de la liste)
 - ArrayAdapter.add(Object) pour remplir la liste
- ***Soit de classe personnalisée (héritage de BaseAdapter)***

Une précision à propos des adaptateurs

- Les adaptateurs sont des classes qui lient des données aux vues de l'UI
 - Les vues concernées étendent `android.widget.AdapterView` : **Spinner**, **Gallery**, **GridView**, **ListView** sont des sous classes concrètes d'`AdapterView`
- Classes d'adaptateurs
 - Héritent de `android.widget.BaseAdapter`
 - `SimpleAdapter`, `ArrayAdapter<?>` : sert à récupérer des données stockées dans une collection
- Exploite par défaut la valeur de la méthode `toString()` des objets de la liste
 - `CursorAdapter` : sert à récupérer des données stockées dans une base de données relationnelle (SQLite)
 - Vous pouvez étendre ces classes de base pour gérer finement vos items (conseillé)

Exemple de ListView

```
<resources>
  <string
name="app_name">MyAppUVS</string>
  <string-array
name="array_technology">
    <item>Android</item>
    <item>Java</item>
    <item>Php</item>
    <item>Hadoop</item>
    <item>Sap</item>
    <item>Python</item>
    <item>Ajax</item>
    <item>C++</item>
    <item>Ruby</item>
    <item>Rails</item>
    <item>.Net</item>
    <item>Perl</item>
  </string-array>
</resources>
```

Values/stringd.xml

```
<LinearLayout
  android:id="@+id/linearLayout1"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:background="#ffccccc"
  android:orientation="vertical"
  tools:layout_editor_absoluteX="0dp"
  tools:layout_editor_absoluteY="0dp">

  <ListView
    android:id="@+id/listView"

  android:layout_width="match_parent"

  android:layout_height="match_parent" />
</LinearLayout>
```

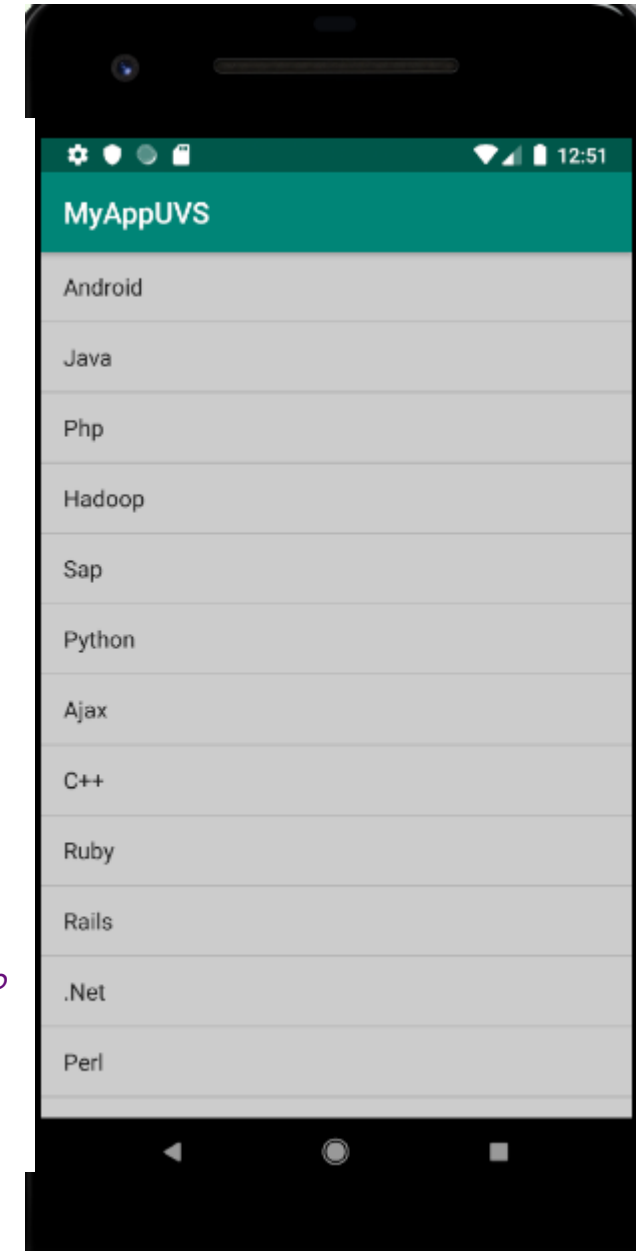
Activity.xml

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

    ListView listView;
    TextView textView;
    String[] listItem;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        listView=(ListView) findViewById(R.id.listView);
        listItem =
        getResources().getStringArray(R.array.array_technology);
        final ArrayAdapter<String> adapter = new
        ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1,
            android.R.id.text1, listItem);
        listView.setAdapter(adapter);
        listView.setOnItemClickListener(new
        AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?>
            adapterView, View view, int position, long l) {
                // TODO Auto-generated method stub
                String value=adapter.getItem(position);

                Toast.makeText(getApplicationContext(),value,Toast.LENGTH_SHO
                RT).show();
            }
        });
    }
}
```

MainActivity.java

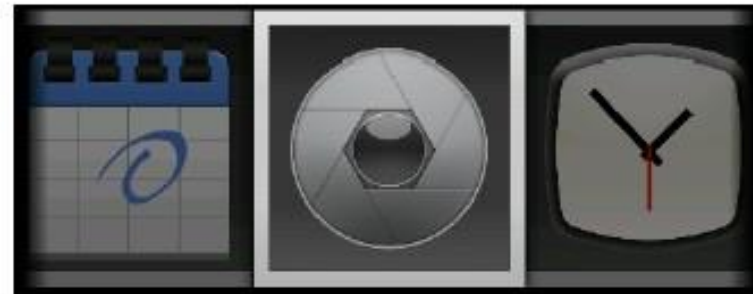


Gallery

- Normalement utilisé pour faire des galeries d'images avec défilement horizontal
- Propriétés supplémentaires
 - `android:animationDuration` Pour définir la durée de la transition (en ms) lorsque l'on passe d'un élément à l'autre
 - `android:unselectedAlpha` Pour définir la transparence des éléments non sélectionnés
- Pour remplir une galerie il faut un **Adapter** (comme pour ListView) mais que l'on doit écrire par héritage de la classe **BaseAdapter** puis l'associer à la galerie par la méthode `setAdapter`

Exemple de fichier XML :

```
<Gallery android:id="@+id/magalerie"  
  android:layout_width="fill_parent"  
  android:layout_height="wrap_content"  
  android:unselectedAlpha="0.5"  
>
```



Liste de choix (Spinner)

- Affiche le choix actuel et affiche un RadioGroup quand on clique dessus pour le changer
- Propriétés :
 - **android:prompt** Pour définir le titre de la fenêtre qui s'ouvre lorsque l'on fait un choix
 - **android:entries="@array/maliste"** définit le contenu de la liste à partir du contenu du fichier string.xml placé dans res/values/ qui a la forme du fichier en face :

```
<resources>
  <string name="hello">Hello World,
    SpinnerExample</string>
  <string name="app_name">SpinnerExample</string>
  <string name="mnth_picker">Select a Month</string>
  <string-array name="mnth_array">
    <item>January</item>
    <item>February</item>
    <item>March</item>
    <item>April</item>
    <item>May</item>
    <item>June</item>
    <item>July</item>
    <item>August</item>
    <item>September</item>
    <item>October</item>
    <item>November</item>
    <item>December</item>
  </string-array>
</resources>
```

Liste de choix (Spinner)

Création du Spinner

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:padding="10dip"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dip"
        android:text="@string/mnth_picker"
    />
    <Spinner android:id="@+id/spinner"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:prompt="@string/mnth_picker"
    />
</LinearLayout>
```

Liste de choix (Spinner)

```
package org.example.SpinnerExample;
import android.app.Activity;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.Spinner;

public class SpinnerExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Spinner spinner = (Spinner) findViewById(R.id.spinner);
        ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
            this, R.array.mnth_array, android.R.layout.simple_spinner_item);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        spinner.setAdapter(adapter);
    }
}
```


Autocomplétion :

AutoCompleteTextView

- C'est une spécialisation de EditText pour apporter l'auto complétion
- Propriétés supplémentaires:
 - android:completionHint="texte" texte affiché en titre du menu déroulant
 - android:completionThreshold Pour définir le nombre de caractères à taper avant que la complétion n'entre en action.
 - android:dropDownHeight="unité" on peut aussi utiliser les constantes fill_parent et wrap_content, définit la hauteur du menu déroulant
 - android:dropDownWidth="unité" on peut aussi utiliser les constantes fill_parent et wrap_content, définit la largeur du menu déroulant
 - android:dropDownHorizontalOffset Pour définir le décalage horizontal du menu déroulant
 - android:dropDownVerticalOffset Pour définir le décalage vertical du menu déroulant

Autocomplétion :

AutoCompleteTextView

- ***Construction***

- AutoCompleteTextView(Context) le paramètre est généralement l'activité elle-même

- ***Aspect et contenu***

- showDropDown() montre la liste déroulante
- dismissDropDown() cache la liste déroulante
- getListSelection() renvoie le rang de la proposition choisie
- setListSelection(int) choisit une proposition par son rang
- setAdapter(ArrayAdapter) Permet de remplir la liste de propositions. La classe ArrayAdapter est décrite dans ListView.
- setCompletionHint(CharSequence) définit le texte affiché en titre du menu déroulant

Autocomplétion: exemple

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <AutoCompleteTextView
        android:id="@+id/complete"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Autocomplétion: exemple

- Déclarer l'activité AutoCompletionActivity suivante :

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.AutoCompleteTextView;
public class AutoCompletionActivity extends Activity {
    private AutoCompleteTextView complete = null;
    // Notre liste de mots que connaîtra l'AutoCompleteTextView
    private static final String[] COULEUR = new String[] {
        "Bleu", "Vert", "Jaune", "Jaune canari", "Rouge", "Orange"
    };
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // On récupère l'AutoCompleteTextView déclaré dans notre layout
        complete = (AutoCompleteTextView) findViewById(R.id.complete);
        complete.setThreshold(2);
        // On associe un adaptateur à notre liste de couleurs...
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_dropdown_item_1line, COULEUR);
        // ... puis on indique que notre AutoCompleteTextView utilise cet adaptateur
        complete.setAdapter(adapter);
    }
}
```



ImageView : Afficher des images

- Permet d'afficher des images
- Propriétés :
 - Contenu
 - `android:src` Pour définir une couleur ou une image.
 - `android:tint` Pour définir une couleur qui teinte l'image
 - Position et dimensions de l'image
 - `android:adjustViewBounds` La taille de l'ImageView sera ou pas modifiée
 - `android:baselineAlignBottom` Cadrage ou pas de l'image en bas de la zone
 - `android:cropToPadding` L'image sera ou pas coupée si elle est plus grande que la taille disponible
 - `android:scaleType` Pour définir le mode de redimensionnement de l'image avec ou sans déformation. (voir exemples transparent suivant)
 - Taille
 - `android:maxHeight` Pour définir la hauteur maximale
 - `android:maxWidth` Pour définir la largeur maximale

**Fin de la
séquence 3**