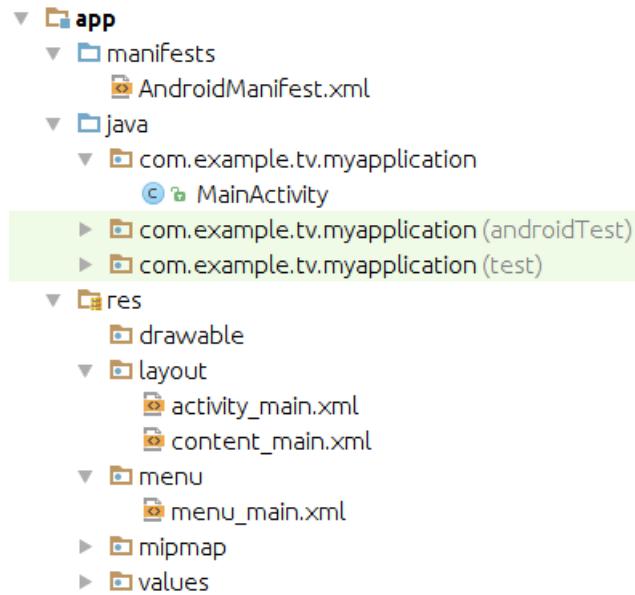


GUI Android

Un projet Android

Les applications Android sont constituées de composants liés par un **manifeste**. Un projet Android contient essentiellement des fichiers sources `.java` et des fichiers de définition XML `.xml`.



AndroidManifest.xml

Chaque projet android doit comporter un fichier XML nommé `AndroidManifest.xml` (localisé dans `app` - > `manifests`) stocké à la racine de la hiérarchie du projet.

Le manifeste décrit les composants, leurs interactions et les métadonnées de l'application, dont notamment ses exigences en matière de plateforme et de matériel.

Activité

Sauf exception, une application Android comporte au moins une classe héritant de `Activity` (ou `AppCompatActivity` pour des raisons de compatibilité) et peut évidemment en avoir plusieurs.

Chaque activité doit impérativement être déclarée dans le fichier `AndroidManifest.xml` dans une balise `<activity>` :

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.tv.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>

```

Remarque : Comme une seule activité peut être lancée au démarrage d'une application, il faut l'indiquer avec la balise `<intent-filter>`. Celle-ci contient deux balises `<action>` (MAIN) et `<category>` (LAUNCHER). On la nomme activité principale de l'application.

GUI

Tous les éléments d'une GUI sont des objets `View` et `ViewGroup` :

- Les objets `View` sont généralement appelés des *widgets* (comme `Button` ou `TextView` par exemple). Ils représentent quelque chose que l'utilisateur peut voir.
- Les objets `ViewGroup` sont généralement appelés *layout* (comme `LinearLayout` ou `ConstraintLayout` par exemple). Ce sont des conteneurs invisibles qui définissent la structure de mise en page pour des objets `View` et d'autres objets `ViewGroup`.

Une GUI sera donc une hiérarchie d'objets `View` et `ViewGroup` imbriqués.

La GUI d'une application Android est décrite dans un **fichier XML**.

Exemple d'un fichier `activity_main.xml` lorsque l'on crée une application Android basée sur le modèle *Empty Activity* :

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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">

    <TextView
        android:id="@+id/texteMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

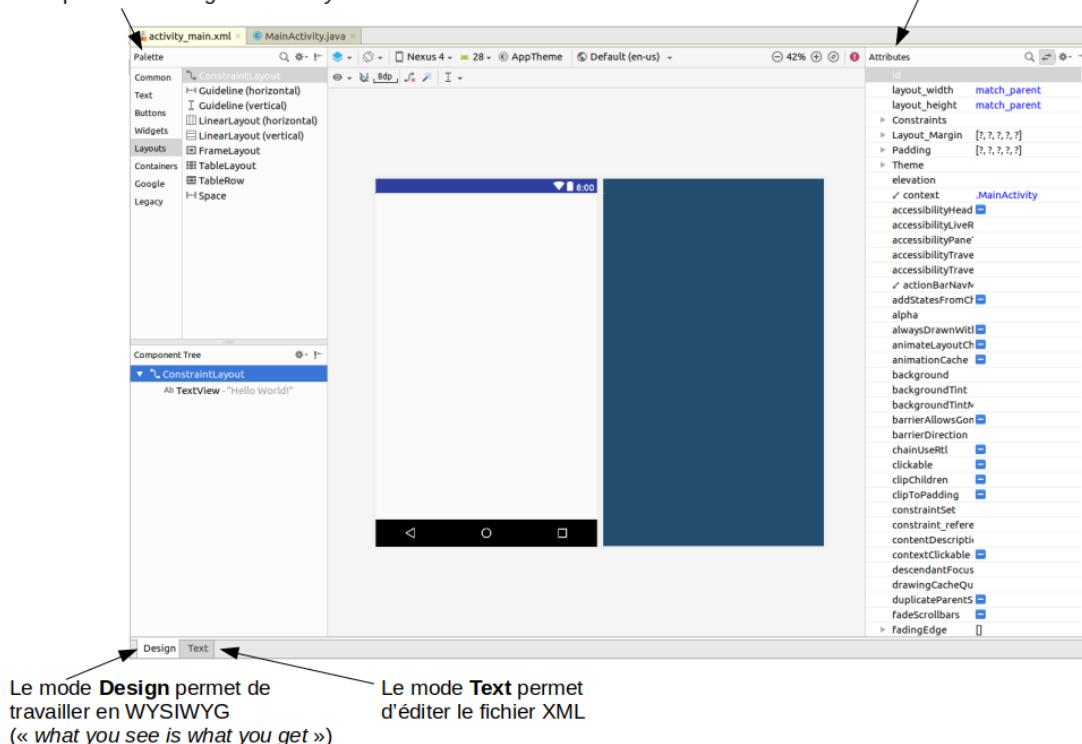
</android.support.constraint.ConstraintLayout>

```

Android Studio fournit un éditeur intégré de GUI :

La palette de composants
comportant les *widgets* et les *layouts*

Chaque composant possède des **attributs**



Remarque : en mode *Design*, Android Studio fournit deux vues, *preview* et *blueprint*, cette dernière étant la plus pratique pour assurer le positionnement des widgets.

Classe

Une application Android met en oeuvre une séparation entre la présentation (la GUI) et son comportement (le code qui la contrôle).

Chaque fichier de disposition XML est compilé dans une ressource `View`.

Il faut charger cette ressource à partir du code de l'activité (dans `onCreate()`). Pour réaliser cela, on appelle `setContentView()` en lui passant la référence de la ressource de mise en page sous la forme :

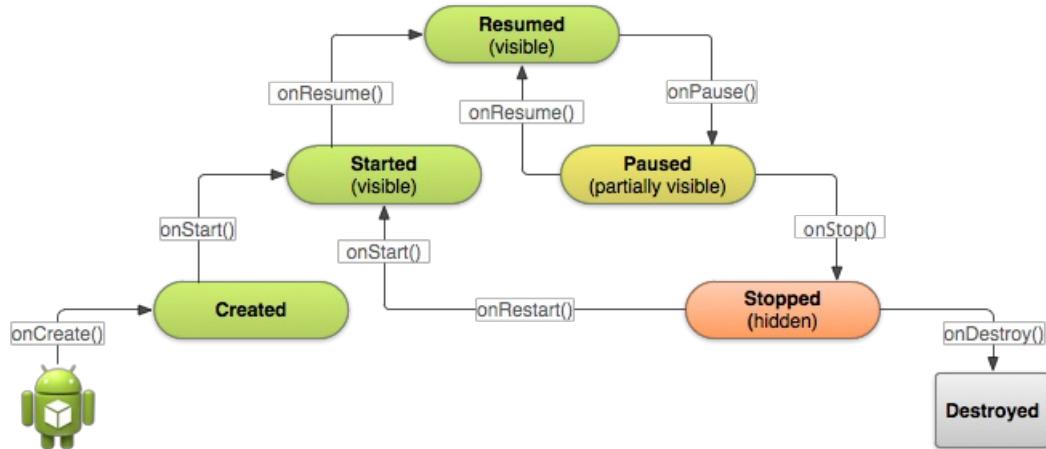
`R.layout.layout_file_name` :

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

Remarque : La méthode de rappel (callback) `onCreate()` d'une activité est appelée par le framework Android lorsque l'activité est lancée.

Le cycle de vie d'une activité

En fonction de l'état de l'activité, il est possible de contrôler celle-ci à l'aide des méthodes héritées de la classe `Activity` :



ID

Chaque composant possède un **ID** afin de l'identifier de manière unique dans l'arborescence.

Remarque : Lorsque l'application est compilée, cet ID est référencé sous la forme d'un entier, mais l'ID est généralement attribué dans le fichier XML sous la forme d'une chaîne dans l'attribut `id` :

```

<TextView
    android:id="@+id/texteMessage"
    ... />
``xml

Il est possible d'interagir avec un composant de la GUI en utilisant son ID :

``java
public class MainActivity extends AppCompatActivity
{
    private TextView texteMessage;

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

        texteMessage = (TextView) findViewById(R.id.texteMessage);

        texteMessage.setText("Bonjour le monde !");
    }
}

```

Action de l'utilisateur

Solution n°1 :

La méthode `setOnClickListener()` installe un gestionnaire d'écoute pour l'objet `Button`. Cette méthode reçoit en argument une interface de *callback* `OnClickListener()` de classe `View`, implémentée par sa méthode `onClick()`.

```

Button boutonShow = (Button) findViewById(R.id.buttonShow);

boutonShow.setOnClickListener(
    new View.OnClickListener()
    {
        public void onClick(View v)
        {
            Toast toast = Toast.makeText(getApplicationContext(), "onClick()", Toast.LENGTH_SHORT);
            toast.show();
        }
    }
);

```

Remarque : Ici, on utilise la classe `Toast` qui fournit un moyen très simple pour signaler à l'utilisateur de manière discrète, rapide et efficace, un événement non bloquant, sous la forme d'un affichage discret et non modal. La méthode `makeText()` crée le composant de classe `Toast` qui est ensuite affiché par `show()`. Elle nécessite trois arguments : le contexte de l'application, le texte à afficher et une durée à choisir entre `LENGTH_SHORT` et `LENGTH_LONG`.

Solution n°2 :

Une autre solution consiste à spécifier que la classe d'activité implémente l'interface

`View.OnClickListener`. L'activité est ensuite enregistrée auprès des éléments concernés afin qu'elle reçoive les événements lorsque l'utilisateur clique sur ceux-ci. Les événements seront alors traités globalement dans la méthode `onClick()` de la classe.

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener
{
    private Button boutonShow;
    private Button boutonHide;

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

        boutonShow = (Button) findViewById(R.id.buttonShow);
        boutonHide = (Button) findViewById(R.id.buttonHide);

        boutonShow.setOnClickListener(this);
        boutonHide.setOnClickListener(this);
    }

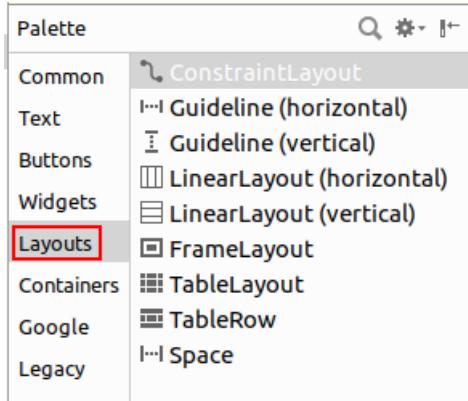
    @Override
    public void onClick(View element)
    {
        Toast toast;

        if (element == boutonShow)
        {
            toast = Toast.makeText(getApplicationContext(), "boutonShow : onClick()", Toast.LENGTH_SHORT);
            toast.show();
        }
        else if (element.getId() == R.id.buttonHide)
        {
            toast = Toast.makeText(getApplicationContext(), "boutonHide : onClick()", Toast.LENGTH_SHORT);
            toast.show();
        }
    }
}
```

Layout

Un *layout* est un système de mise en page dont la structure est prédéfinie.

Android fournit différents *layouts* :



- Les *layouts* anciens : `LinearLayout`, `RelativeLayout` et `FrameLayout`
- Les *layouts* classiques : `GridLayout` et `TableLayout`
- Les *layouts* récents : `ConstraintLayout` (pour remplacer avantageusement `RelativeLayout`), `CoordinatorLayout`, et `CollapsingToolbarLayout`

Tous les *layouts* incluent une largeur et une hauteur (`layout_width` et `layout_height`) et il est nécessaire de les définir. De nombreux *layouts* possèdent également des marges et des bordures facultatives.

Même si il est possible de définir la largeur et la hauteur avec des valeurs exactes (des valeurs absolues telles que des pixels), il est déconseillé de le faire. Généralement, on utilisera l'une de ces constantes pour définir la largeur ou la hauteur :

- `wrap_content` : les dimensions requises par son contenu ;
- `match_parent` : aussi grande que le parent le permet (remplace `fill_parent`).
- `match_constraint` : utilisé à la place de `match_parent` dans `ConstraintLayout` (= `0dp`)

Sinon, il est possible d'utiliser des mesures relatives telles que les unités de pixels indépendantes de la densité (`dp`).

Cette approche permettra de s'assurer que l'application s'affichera correctement sur une variété de tailles d'écran d'appareils.

LinearLayout

Le *layout* `LinearLayout` organise les composants sur une ligne suivant une `orientation` qui peut être `vertical` (en colonne) ou `horizontal` (en ligne).

L'attribut `android:gravity` permet de définir le placement du contenu dans le composant. Par contre, l'attribut `android:layout_gravity` permettra de définir le placement du composant dans le *layout*.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="match_parent"
    android:layout_margin="50dp"
    android:padding="25dp"
    android:gravity="center"
    android:orientation="vertical">

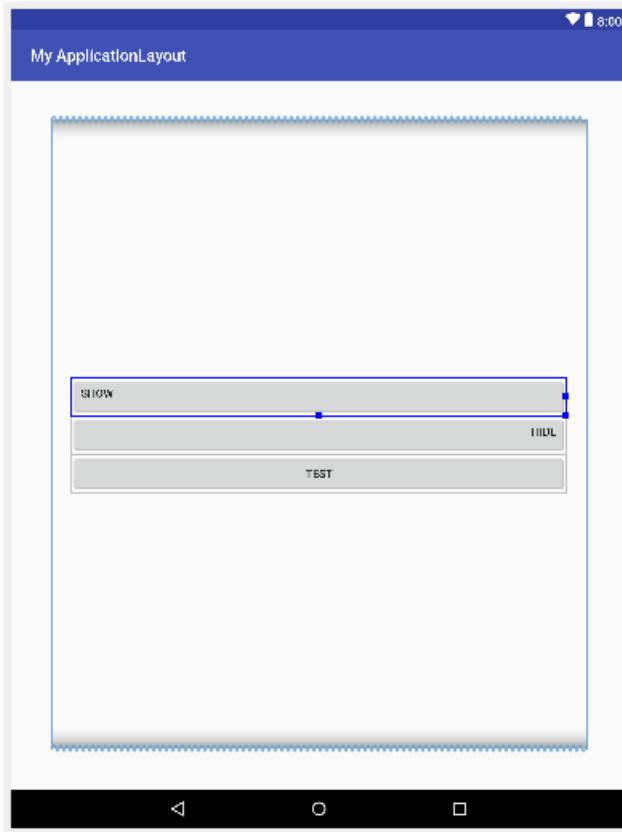
    <Button
        android:id="@+id/buttonShow"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="left"
        android:text="Show" />

    <Button
        android:id="@+id/buttonHide"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="right"
        android:text="Hide" />

    <Button
        android:id="@+id/buttonTest"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="Test" />

</LinearLayout>
```

Ce qui donne :



ConstraintLayout

Le `ConstraintLayout` reprend le fonctionnement du `RelativeLayout` de manière avantageuse :

- gain en performances dans les calculs de dimensions
- placement “à plat” (pour éviter l’imbrication des layouts)
- api externe (permettant de choisir la version et avec une rétro-compatibilité jusqu’à l’API 9 qui couvre 99.9 % des appareils)

`ConstraintLayout` permet de placer les widgets de façon relative aux autres à l’aide de “contraintes” de positionnement, similaires aux règles du `RelativeLayout` (`layout_alignTop`, `layout_toRightOf`, etc.).

Les attributs de contrainte de base ont la forme : `layout_constraintX_toYof` avec `X` le widget où l’on place la contrainte et `Y` le widget de destination.

- Right/Left (ou Start/End) : horizontalement
- Top/Bottom : verticalement

L’ensemble des contraintes disponibles :

layout_constraints		
barrierAllowsGoneWidgets	layout_constraintGuide_percent	layout_constraintVertical_chainStyle
barrierDirection	layout_constraintHeight_default	layout_constraintVertical_weight
chainUseRtl	layout_constraintHeight_max	layout_constraintWidth_default
constraintSet	layout_constraintHeight_min	layout_constraintWidth_max
constraint_referenced_ids	layout_constraintHeight_percent	layout_constraintWidth_min
layout_constrainedHeight	layout_constraintHorizontal_bias	layout_constraintWidth_percent
layout_constrainedWidth	layout_constraintHorizontal_chainStyle	layout_editor_absoluteX
layout_constraintBaseline_creator	layout_constraintHorizontal_weight	layout_editor_absoluteY
layout_constraintBaseline_toBaseline...	layout_constraintLeft_creator	layout_goneMarginBottom
layout_constraintBottom_creator	layout_constraintLeft_toLeftOf	layout_goneMarginEnd
layout_constraintBottom_toBottom...	layout_constraintLeft_toRightOf	layout_goneMarginLeft
layout_constraintBottom_toTopOf	layout_constraintRight_creator	layout_goneMarginRight
layout_constraintCircle	layout_constraintRight_toLeftOf	layout_goneMarginStart
layout_constraintCircleAngle	layout_constraintRight_toRightOf	layout_goneMarginTop
layout_constraintCircleRadius	layout_constraintStart_toEndOf	
layout_constraintDimensionRatio	layout_constraintStart_toStartOf	
layout_constraintEnd_toEndOf	layout_constraintTop_creator	
layout_constraintEnd_toStartOf	layout_constraintTop_toBottomOf	
layout_constraintGuide_begin	layout_constraintTop_toTopOf	
layout_constraintGuide_end	layout_constraintVertical_bias	

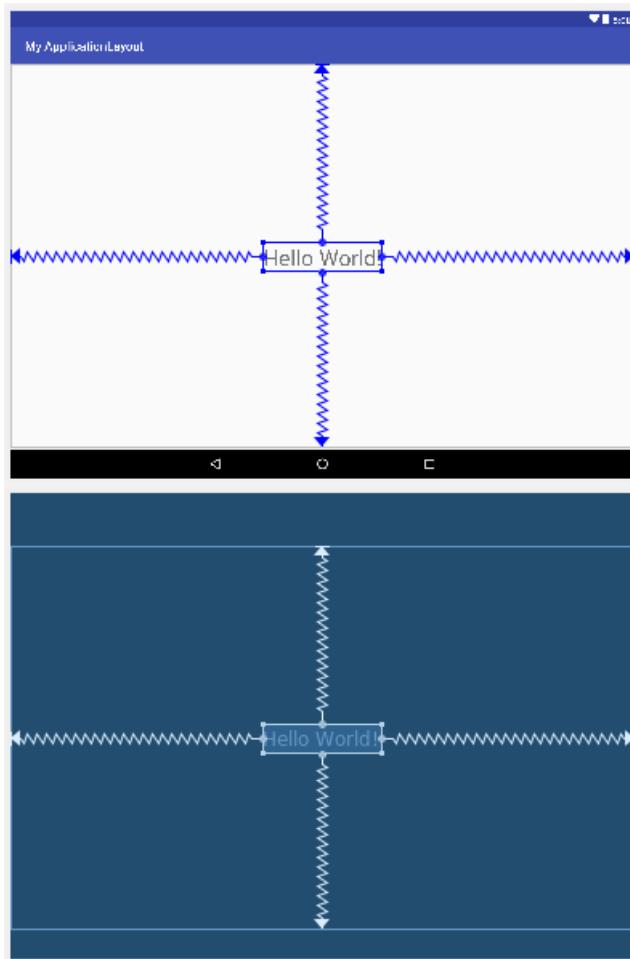
Exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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">

    <TextView
        android:id="@+id/texteMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Ce qui donne (un placement automatiquement centré ici):



Avec deux boutons :

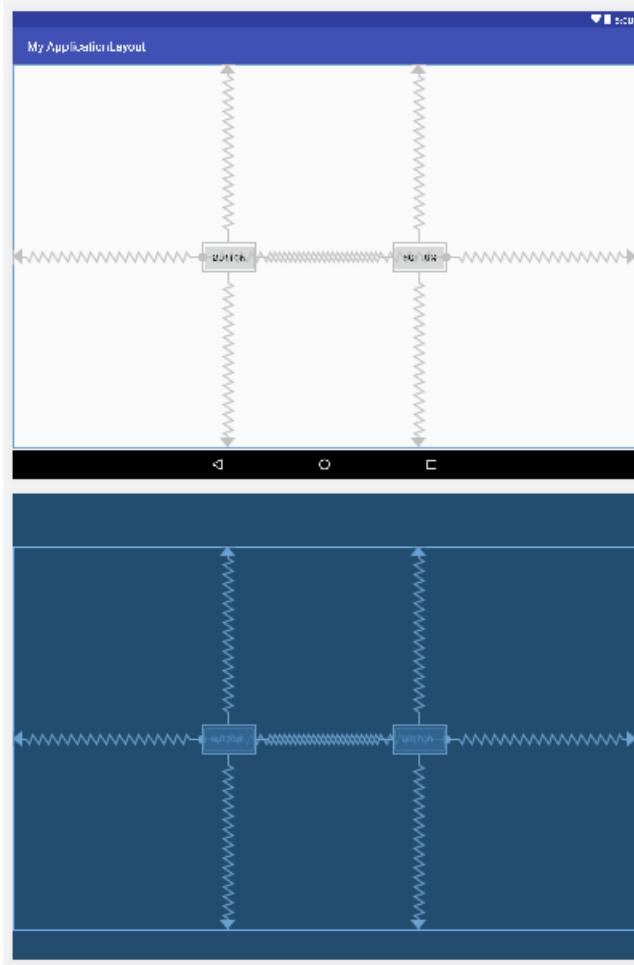
```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="@+id/button3"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="@+id/button2"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        />

</android.support.constraint.ConstraintLayout>
```

Ce qui donne :



`ConstraintLayout` introduit aussi les concepts :

- de biais en % avec `layout_constraintVertical_bias` et/ou `layout_constraintHorizontal_bias` (ici `0.2`)



- de ratio en appliquant `match_constraint` (qui fixe le `layout_width` et/ou `layout_height` à `0dp`) et un rapport dans `layout_constraintDimensionRatio`

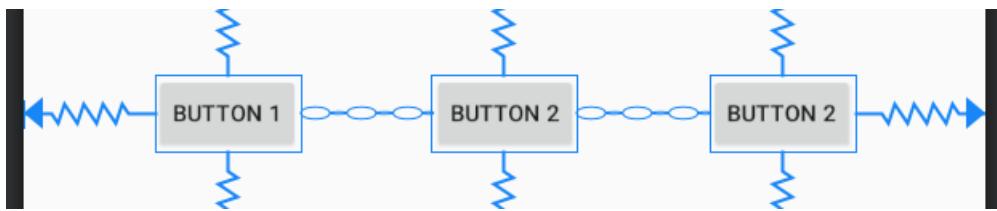


- de **chaînes** (avec `layout_constraintHorizontal_chainStyle` appliqué à l'élément de **tête de chaîne** et qui peut prendre par exemple les valeurs `spread`, `spread_inside` ou `packed`) où les éléments sont liés entre eux avec `layout_constraintLeft_toRightOf` et `layout_constraintRight_toLeftOf`

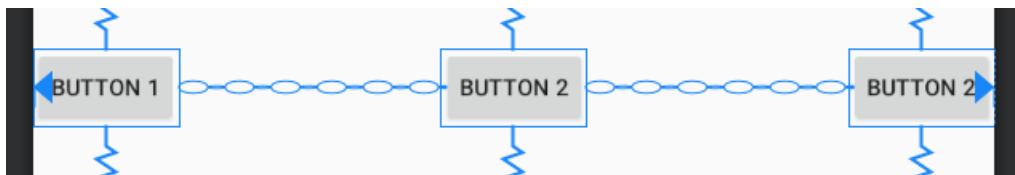


Ce qui donne avec :

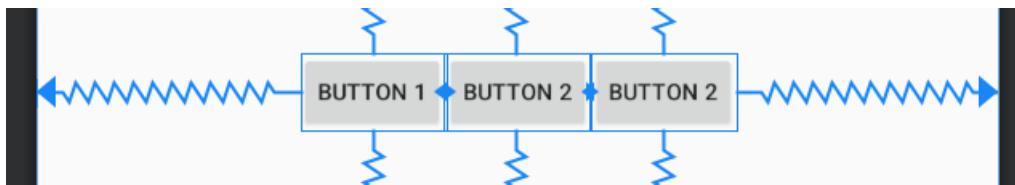
- `spread`



- `spread_inside`

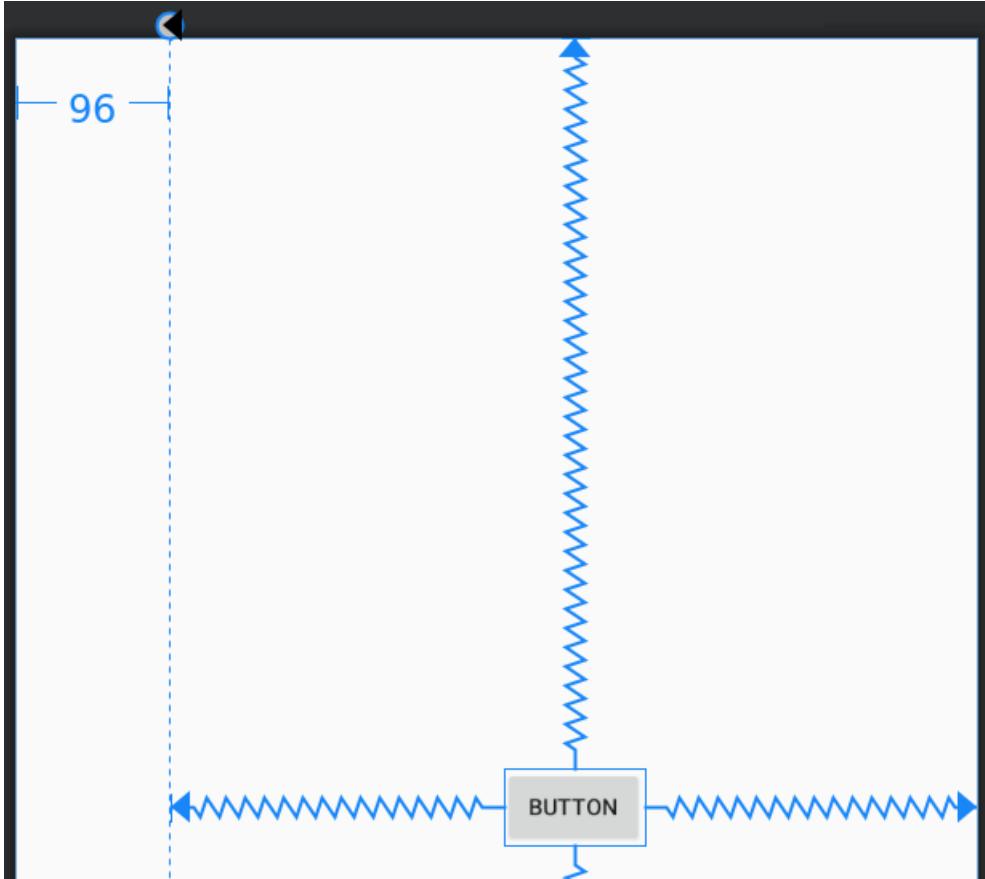


- `packed`



- de **lignes directrices** (*guidelines*) que l'on peut suivre (l'utilisation des % est possible avec `layout_constraintGuide_percent`) et des **barrières** (*barrier*) que l'on ne peut pas dépasser

```
<android.support.constraint.Guideline
    android:id="@+id/guideline"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintGuide_begin="96dp" />
```



- de définir des min/max sur la largeur et hauteur (par exemple `layout_constraintHeight_max` et un % dans `layout_constraintHeight_percent`)
- de réappliquer des contraintes avec la classe `ConstrainSet`

GridLayout

Le `GridLayout` permet de placer les widgets sur une grille. Les attributs `android:rowCount` et `android:columnCount` définissent le nombre de lignes et de colonnes.

Chaque composant se positionne dans la grille avec les attributs `android:layout_row` pour le numéro de ligne et `android:layout_column` pour le numéro de colonne. Il est possible de fusionner des lignes avec `android:layout_rowSpan` et des colonnes avec `android:layout_columnSpan`.

L'attribut `android:layout_weight` permet de définir le « poids » (l'espace) que le composant peut prendre à l'intérieur du *layout*.

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:columnCount="3"
    android:padding="10dp"
    android:rowCount="3">>

    <Button
        android:id="@+id/case1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="0"
        android:layout_margin="5dp"
        android:layout_marginStart="5dp"
        android:layout_row="0"
        android:layout_rowWeight="1"
        android:layout_columnWeight="1"
        android:textColor="#FFFFFF"
        android:text="ZONE 1"/>
    <Button
        android:id="@+id/case2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="1"
        android:layout_margin="5dp"
        android:layout_row="0"
        android:layout_rowWeight="1"
        android:layout_columnWeight="1"
        android:textColor="#FFFFFF"
        android:text="ZONE 2"/>
    <Button
        android:id="@+id/case3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="2"
        android:layout_margin="5dp"
        android:layout_row="0"
        android:layout_rowWeight="1"
        android:layout_columnWeight="1"
        android:textColor="#FFFFFF"
        android:text="ZONE 3"/>
    <Button
        android:id="@+id/case4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="0"
        android:layout_columnSpan="3"
        android:layout_margin="5dp"
        android:layout_row="1"
        android:layout_rowWeight="1"
        android:layout_columnWeight="1"
        android:textColor="#FFFFFF"
        android:text="ZONE 4"/>
    <Button
        android:id="@+id/case5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="0"
        android:layout_margin="5dp"
        android:layout_marginBottom="55dp"
        android:layout_row="2"
        android:layout_rowWeight="1"
```

```

        android:layout_rowWeight="1"
        android:layout_columnWeight="1"
        android:textColor="#FFFFFF"
        android:text="ZONE 5"/>
<Button
        android:id="@+id/case6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="1"
        android:layout_margin="5dp"
        android:layout_marginBottom="55dp"
        android:layout_row="2"
        android:layout_rowWeight="1"
        android:layout_columnWeight="1"
        android:textColor="#FFFFFF"
        android:text="ZONE 6"/>
<Button
        android:id="@+id/case7"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="2"
        android:layout_margin="5dp"
        android:layout_marginBottom="55dp"
        android:layout_row="2"
        android:layout_rowWeight="1"
        android:layout_columnWeight="1"
        android:textColor="#FFFFFF"
        android:text="ZONE 7"/>
</GridLayout>

```

Ce qui donne :



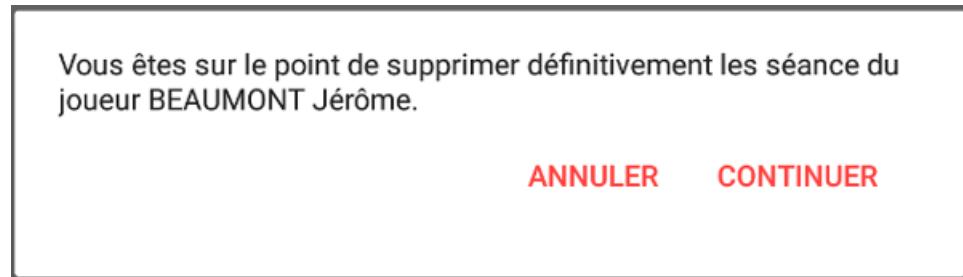
Boîtes de dialogue

Une boîte de dialogue est une fenêtre (qui ne remplit pas l'écran) qui invite l'utilisateur à prendre une décision ou à entrer des informations supplémentaires. La classe [Dialog](#) est la classe de base des dialogues mais on utilisera une [AlertDialog](#) qui est une de ses sous-classes.

Le principe de base est le suivant :

```
AlertDialog.Builder boiteSuppression = new AlertDialog.Builder(this);
boiteSuppression.setMessage("Vous êtes sur le point de supprimer définitivement un élément.");
boiteSuppression.setPositiveButton("Continuer", new DialogInterface.OnClickListener()
{
    public void onClick(DialogInterface dialog, int which)
    {
        // ici l'action de suppression
        ...
    }
});
boiteSuppression.setNegativeButton("Annuler", new DialogInterface.OnClickListener()
{
    public void onClick(DialogInterface dialog, int which)
    {
    }
});
boiteSuppression.show();
```

On obtient quelque chose comme ceci :



Il est possible d'associer une vue à base de *layout* :

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <TextView
        android:id="@+id/texteChoixNumeroTable"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="16dp"
        android:text="Numéro de la table TTPA ?"
        android:textSize="24sp"
        android:textStyle="bold"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <NumberPicker
        android:id="@+id/numeroTable"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/texteChoixNumeroTable" />

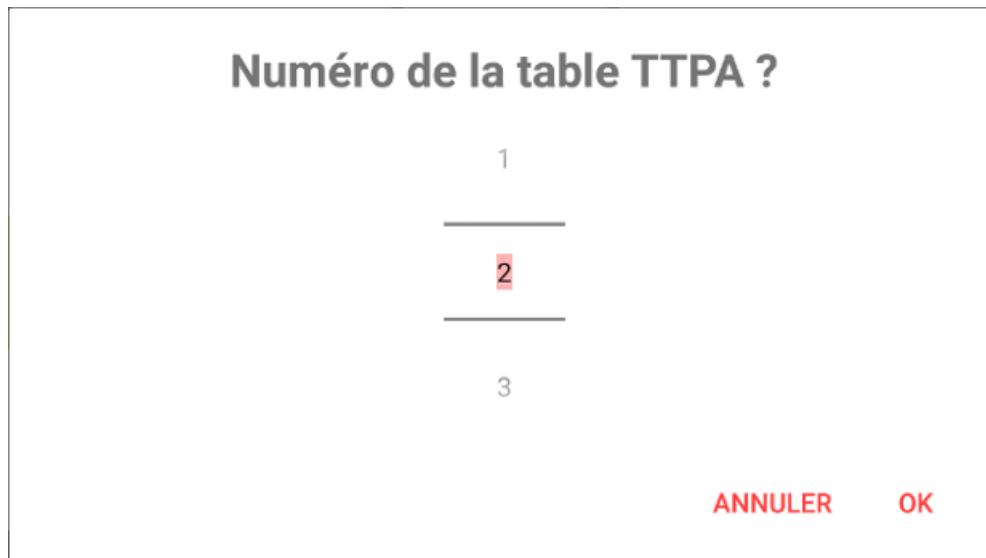
</android.support.constraint.ConstraintLayout>
```

Il suffit maintenant de l'associer avec `setView()` :

```

AlertDialog.Builder choixNumeroTable = new AlertDialog.Builder(this);
LayoutInflater inflater = this.getLayoutInflater();
View vue = inflater.inflate(R.layout.choix_numero_table, null);
choixNumeroTable.setView(vue);
npNumeroTable = (NumberPicker) vue.findViewById(R.id.numeroTable);
npNumeroTable.setMaxValue(9);
npNumeroTable.setMinValue(1);
// numeroTable n'a pas encore été choisi ?
if(numeroTable == 0)
    npNumeroTable.setValue(1); // valeur par défaut
else
    npNumeroTable.setValue(numeroTable); // valeur précédemment choisie
npNumeroTable.setWrapSelectorWheel(false);
choixNumeroTable.setPositiveButton("OK", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {
        numeroTable = npNumeroTable.getValue();
    }
});
choixNumeroTable.setNegativeButton("Annuler", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {
    }
});
AlertDialog choixNumeroTableDialog = choixNumeroTable.create();
choixNumeroTableDialog.show();

```



Il est possible d'ajuster la taille de la boîte de dialogue (ici à 80%) juste après l'appel à `show()` :

```

Rect displayRectangle = new Rect();
Window window = this.getWindow();
window.getDecorView().getWindowVisibleDisplayFrame(displayRectangle);
choixNumeroTableDialog.getWindow().setLayout((int)(displayRectangle.width() * 0.8f), choixNumeroTableDialog.getWindow().getAttributes().height);
//choixNumeroTableDialog.getWindow().setLayout((int)(displayRectangle.width() * 0.8f), (int)(displayRectangle.height() * 0.8f));

```

Lancement d'une nouvelle activité

En pratique, une application est souvent constituée de plusieurs pages parmi lesquelles l'utilisateur peut naviguer selon son gré. Et sous Android, une nouvelle page est ... une nouvelle activité ! On va donc ajouter au projet une nouvelle activité.

Pour cela :

- 1 . On crée un nouveau fichier ressource de type *layout*. (Facultatif*)
- 2 . On crée une nouvelle activité Empty (*le fichier ressource/*layout* aurait pu être créé automatiquement) et on la nomme par exemple MyActivity.
- 3 . Dans la classe de la nouvelle activité, on appelle `setContentView()` pour associer le *layout* à l'activité.

La nouvelle activité a été ajoutée au fichier `AndroidManifest.xml` :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.tv.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".MyActivity"></activity>
    </application>
</manifest>
```

Pour afficher la nouvelle activité, on a besoin d'un objet de type `Intent` . Un `Intent` est un objet permettant la transmission entre composants de messages contenant de l'information : demande de démarrage d'une autre activité, action à effectuer, ... Le lancement effectif de l'intention est réalisé par l'appel à `startActivity()` .

```
Intent intent = new Intent(MainActivity.this, MyActivity.class);
startActivity(intent);
```

Remarque : La touche retour de l'appareil permet de revenir sur l'activité principale.

Pour retourner d'une activité, il suffit d'appeler la méthode `finish()` .

Il est aussi possible de récupérer un code de retour de l'activité en appelant `startActivityForResult()` au lancement :

```
startActivityForResult(intent, 0); // un code peut être passé au lancement (ici 0)
```

L'activité doit fixer le code de retour avec `setResult()` puis appeler `finish()` :

```
setResult(1);
finish();
```

Le code retourné peut être récupéré avec la méthode `callback onActivityResult()` :

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    Log.v("onActivityResult()", "resultCode : " + resultCode);
}
```

L'`Intent` utilisé peut servir à “passer” des données à l'activité lancée :

```
Intent intent = new Intent(MainActivity.this, MyActivity.class);
String srtTempMin = "TempMin";
double tempMin = 10.5;
intent.putExtra(srtTempMin, tempMin);
startActivity(intent);
```

Les données sont récupérées :

```
public class MyActivity extends AppCompatActivity
{
    private double tempMin;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.myactivity);
        Intent intent = getIntent();
        tempMin = intent.getDoubleExtra("TempMin", 23);
        ...
    }
}
```

Il est possible de “passer” un objet :

```
Intent intent = new Intent(MainActivity.this, MyActivity.class);
MaClasse unObjet = new MaClasse();
intent.putExtra("unObjet", (Serializable)unObjet);
startActivity(intent);
```

Il faut que la classe utilisée implémente `Serializable` (ou `Parcelable`) :

```
public class Salle implements Serializable
{
    // ...
}
```

L'objet est récupéré comme ceci :

```
public class MyActivity extends AppCompatActivity
{
    MaClasse unObjet;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.myactivity);
        Intent intent = getIntent();
        unObjet = (MaClasse)intent.getSerializableExtra("unObjet");
        ...
    }
}
```

Evidemment, cela fonctionne aussi dans l'autre sens. Dans ce cas, les données "retournées" seront récupérées via la méthode *callback* `onActivityResult()`.

Il est aussi possible de concevoir une activité de façon modulaire en utilisant les[fragments](#). Un fragment (un peu comme une "sous-activité") représente un comportement ou une partie de l'interface utilisateur dans un [FragmentActivity](#). On peut combiner plusieurs fragments dans une seule activité et réutiliser un fragment dans plusieurs activités. Un fragment a son propre cycle de vie, reçoit ses propres événements d'entrée ...

Tutoriel : mathias-seguy.developpez.com/tutoriels/android/comprendre-fragments/

Vue dynamique

Liste déroulante

a . Intégrer un `Spinner` dans le *layout* :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="50dp"
    android:padding="25dp"
    android:gravity="center"
    android:orientation="vertical">

    <Spinner
        android:id="@+id/listeNoms"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:padding="10dp"/>

</LinearLayout>
```

b. Utiliser un adaptateur (comme `ArrayAdapter`) qui se comporte comme un intermédiaire entre les données et la vue :

```

public class MainActivity extends AppCompatActivity implements View.OnClickListener
{
    Spinner listeNoms; // la vue
    List<String> noms; // les données
    ArrayAdapter<String> adapter; // l'adaptateur

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

        listeNoms = (Spinner) findViewById(R.id.listeNoms);

        // Des données à placer dans la liste déroulante
        noms = new ArrayList<String>();
        noms.add("Toto");
        noms.add("Titi");
        noms.add("Tata");

        // On associe les données à l'adaptateur
        adapter = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, noms);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

        // On associe l'adaptateur à la vue
        listeNoms.setAdapter(adapter);

        // On installe le gestionnaire d'écoute lors de la sélection d'une entrée de la liste
        listeNoms.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()
        {
            @Override
            public void onItemSelected(AdapterView<?> arg0, View arg1, int position, long id)
            {
                Log.v("listeNoms", "position : " + position + " - " + " nom : " + noms.get(position));
            }
            @Override
            public void onNothingSelected(AdapterView<?> arg0)
            {
                // TODO Auto-generated method stub
            }
        });
    }
}

```

Remarque : Ici, on utilise un *layout* prédéfini par Android : `R.layout.simple_spinner_item`. Evidemment, il est possible de personnaliser son *layout*.

Liste des *layouts* prédéfinis : [R.layout](#)

Une vue en liste

Un `ListView` est un `ViewGroup` qui affiche des éléments selon une liste. C'est une vue très utilisée dans les applications Android (par exemple pour faire une liste de Contacts ou de News ...). Un `ListView` est composé de `ListItem`.

a . Intégrer un `ListView` dans le *layout* :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="50dp"
    android:padding="25dp"
    android:gravity="center"
    android:orientation="vertical">

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

</LinearLayout>
```

b. Utiliser un adaptateur (comme `ArrayAdapter`) qui se comporte comme un intermédiaire entre les données et la vue :

```

public class MainActivity extends AppCompatActivity implements View.OnClickListener
{
    ListView listeNoms; // la vue
    List<String> noms; // les données
    ArrayAdapter<String> adapter; // l'adaptateur

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

        listeNoms = (ListView) findViewById(R.id.listeNoms);

        // Des données à placer dans la liste déroulante
        noms = new ArrayList<String>();
        noms.add("Toto");
        noms.add("Titi");
        noms.add("Tata");

        // On associe les données à l'adaptateur
        adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, noms);

        // On associe l'adaptateur à la vue
        listeNoms.setAdapter(adapter);

        // On installe le gestionnaire d'écoute pour le clic
        listeNoms.setOnItemClickListener(new AdapterView.OnItemClickListener()
        {
            @Override
            public void onItemClick(AdapterView<?> a, View v, int position, long id)
            {
                Log.v("listeNoms", "position : " + position + " - " + " nom : " + noms.get(position));
            }
        });
    }
}

```

Remarque : Ici, on utilise un *layout* prédefini par Android : `R.layout.simple_list_item_1`. Il existe d'autres *layouts* prédefinis comme : `R.layout.simple_list_item_checked` ou `simple_list_item_multiple_choice`. Evidemment, il est possible de personnaliser son *layout*.

Liste des *layouts* prédefinis : [R.layout](#)

A titre d'exemple, voici le *layout* prédefini `R.layout.simple_list_item_1` :

```

<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/text1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceListItemSmall"
    android:gravity="center_vertical"
    android:paddingStart="?android:attr/listPreferredItemPaddingStart"
    android:paddingEnd="?android:attr/listPreferredItemPaddingEnd"
    android:minHeight="?android:attr/listPreferredItemHeightSmall" />

```

Préférences de l'application

Il est souvent nécessaire de sauvegarder des informations propres à l'application (comme le nom du dernier utilisateur connecté). Pour cela, il existe une solution simple (autre que des fichiers ou d'une base de données SQLite) : [SharedPreferences](#).

Les `SharedPreferences` offre un système de persistance de données (clé/valeur) pour l'application.

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener
{
    ListView listeNoms; // la vue
    List<String> noms; // les données
    ArrayAdapter<String> adapter; // l'adaptateur

    public static final String PREFERENCES = "preferences";
    public static final String PREFERENCES_NOM = "Nom";
    SharedPreferences sharedpreferences;

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

        // On récupère le SharedPreference à partir du contexte
        sharedpreferences = getBaseContext().getSharedPreferences(PREFERENCES, MODE_PRIVATE);

        // On récupère un élément si il existe
        if (sharedpreferences.contains(PREFERENCES_NOM))
        {
            // (voir aussi getBoolean(), getFloat(), getInt() ...)
            String nom = sharedpreferences.getString(PREFERENCES_NOM, null); // null ou une valeur par défaut
            Toast.makeText(getApplicationContext(), "Dernier nom sélectionné : " + nom, Toast.LENGTH_SHORT).show();
        }

        listeNoms = (ListView) findViewById(R.id.listeNoms);

        // Des données à placer dans la liste déroulante
        noms = new ArrayList<String>();
        noms.add("Toto");
        noms.add("Titi");
        noms.add("Tata");

        // On associe les données à l'adaptateur
        adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, noms);

        // On associe l'adaptateur à la vue
        listeNoms.setAdapter(adapter);

        // On installe le gestionnaire d'écoute pour le clic
        listeNoms.setOnItemClickListener(new AdapterView.OnItemClickListener()
        {
            @Override
            public void onItemClick(AdapterView<?> a, View v, int position, long id)
            {
                Log.v("listeNoms", "position : " + position + " - " + " nom : " + noms.get(position));
            }
        });
    }
}
```

```
        ...
        Toast.makeText(getApplicationContext(), "Nom sélectionné : " + noms.get(position)
        , Toast.LENGTH_SHORT).show();

        // On sauvegarde un élément (voir aussi putBoolean(), putFloat(),.putInt() ...)
        sharedpreferences.edit().putString(PREFERENCES_NOM, noms.get(position)).apply();
    // ou .commit()
    }
}
}
}
```

Splash screen (écran de démarrage)

Cet exemple montre l'intégration d'un écran de démarrage réalisée à partir d'une animation.

Créer un nouveau répertoire nommé `anim` dans le répertoire `res` avec un clic droit. Refaire un clic droit sur `anim` et créer un nouveau fichier de ressources d'animation nommé `fade_in.xml` :

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <alpha
        android:duration="1000"
        android:fromAlpha="0.0"
        android:interpolator="@android:anim/accelerate_interpolator"
        android:toAlpha="1.0" />
</set>
```

Créer une activité vide à l'aide d'Android Studio nommée `Splash` :

```
public class Splash extends AppCompatActivity
{
    private Animation animation;
    private ImageView imageView;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);

        imageView = (ImageView) findViewById(R.id.imageView);
        animation = AnimationUtils.loadAnimation(getApplicationContext(), R.anim.fade_in);
        animation.setAnimationListener(new Animation.AnimationListener()
        {
            @Override
            public void onAnimationStart(Animation animation)
            {

            }

            @Override
            public void onAnimationEnd(Animation animation)
            {
                // A la fin de l'animation, on lance l'activité principale
                Intent intent = new Intent(Splash.this, MainActivity.class);
                startActivity(intent);
            }

            @Override
            public void onAnimationRepeat(Animation animation)
            {
            }
        });
        imageView.startAnimation(animation);
    }
}
```

Ajouter une image (un logo par exemple) dans le layout `activity_splash.xml` :

```

<?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=".Splash">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/imageView"
        android:src="@drawable/logo"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

Dans le fichier `AndroidManifest.xml`, on paramètre l'activité `Splash` comme la seule activité à être lancée au démarrage (cf. balise `<intent-filter>` comprenant `<action>` (MAIN) et `<category>` (LAUNCHER)) :

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".Splash">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".MainActivity">
        </activity>
    </application>

</manifest>

```

RecyclerView, CardView et Glide

Le widget `RecyclerView` est une version plus avancée de `ListView`. Le `RecyclerView` est notamment adapté pour des éléments basés sur de grands ensembles de données ou des données qui changent fréquemment.

Dans l'exemple, on affiche une liste de skieurs participant à la coupe du monde de ski alpin 2020 en utilisant : `RecyclerView`, `CardView` et la bibliothèque `Glide`.

100% 13:50

MyApplicationView

- 1 Alexis PINTURAULT FRA
 1148 points
- 2 Aleksander Aamodt KILDE NOR
 1122 points
- 3 Henrik KRISTOFFERSEN NOR
 1041 points
- 4 Matthias MAYER AUT
 816 points
- 5 Vincent KRIECHMAYR AUT
 776 points
- 6 Beat FEUZ SUI
 742 points

◀ ○ □

Lire : [RecyclerView, CardView et Glide \[PDF\]](#)

© Thierry Vaira <tvaira@free.fr>