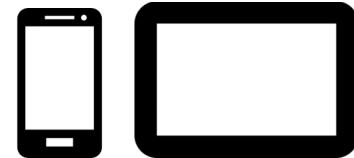
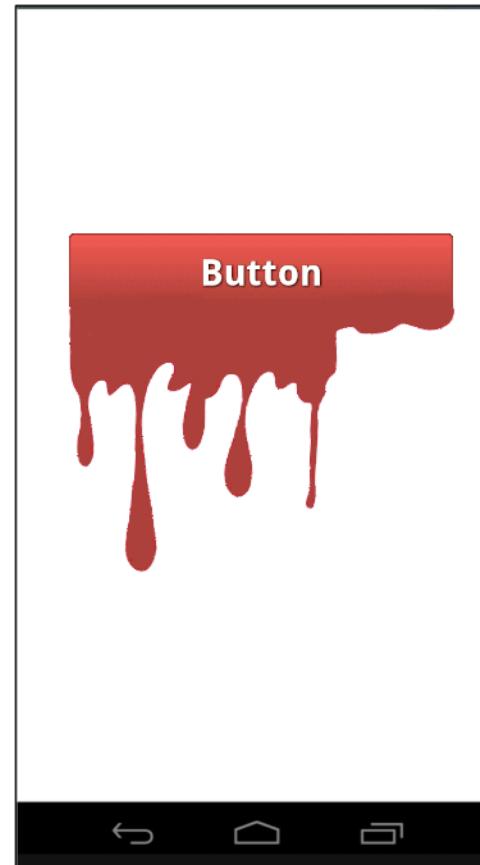


# [TD1] INTRODUCTION AUX INTERFACES





# PRINCIPALES BRIQUES DANS ANDROID

## Activity

Brique principale d'une application, elle représente l'implémentation et les interactions des interfaces.

## Fragment

Un fragment peut être considéré comme une portion d'une interface ; une interface peut donc être composée d'un ou plusieurs fragments.

## Service

Un service ne possède pas d'interface, mais permet d'exécuter de longs traitements. Un service ne s'arrête pas tant qu'il n'est pas interrompu ou terminé.



# PRINCIPALES BRIQUES DANS ANDROID

## Intents

Les composants Android communiquent via des messages systèmes que l'on appelle Intents. Ils sont émis par le terminal pour prévenir du déclenchement d'événements.

## Broadcast receiver

Un broadcast receiver est un composant qui réagit à un événement système. Il permet de savoir si : le smartphone reçoit un SMS, l'écran est verrouillé...

## Content provider

Un content provider permet de partager les données d'une application, le but étant de permettre à d'autres applications de requêter ces données.



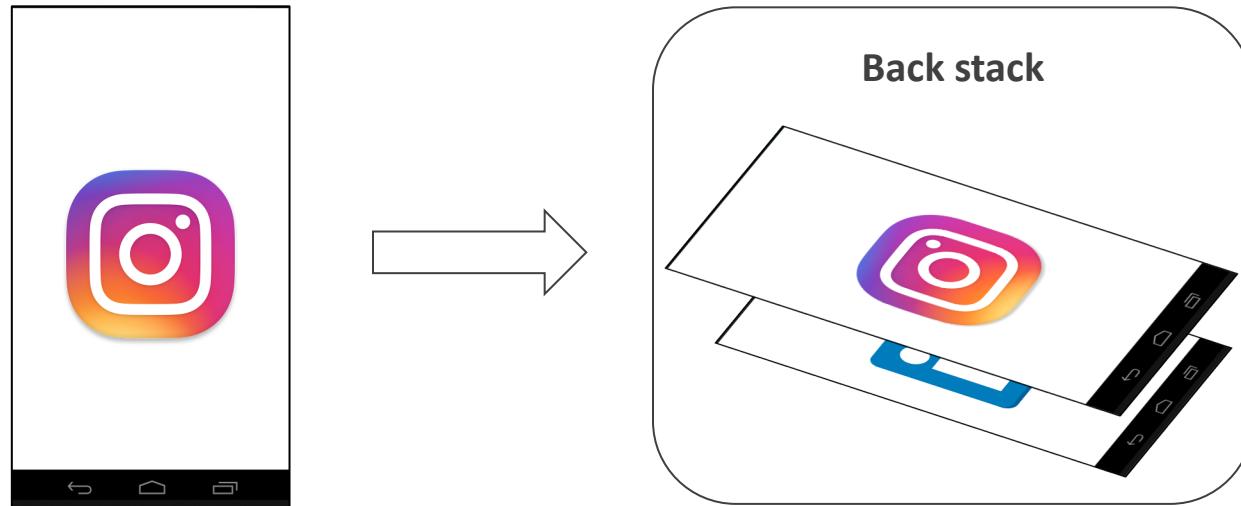
# CYCLE DE VIE D'UNE ACTIVITÉ

## Back stack

Android n'affiche qu'une vue à la fois, la **succession des pages est stockée dans une pile appelée Back stack**.

Cela permet de revenir à la page précédente grâce au bouton « retour ».

Toutes les activités sont stockées dans le Back stack, ajoutées par le haut de la pile.

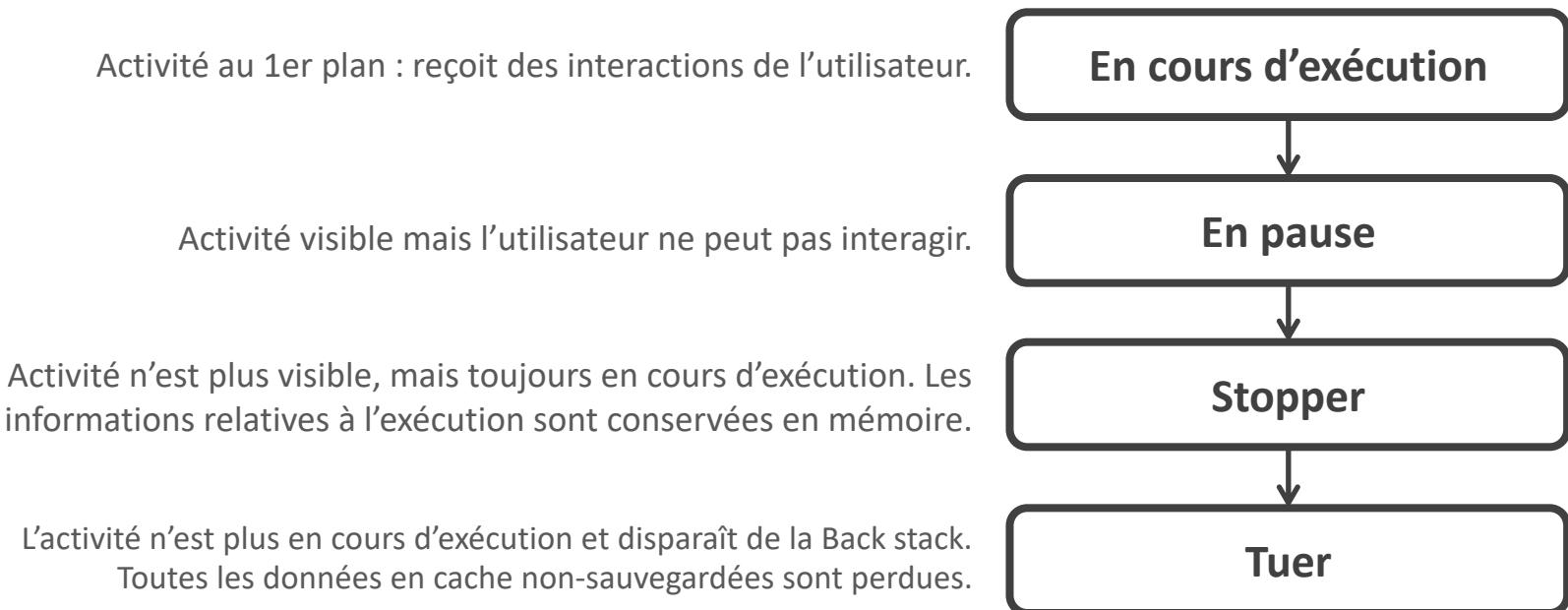


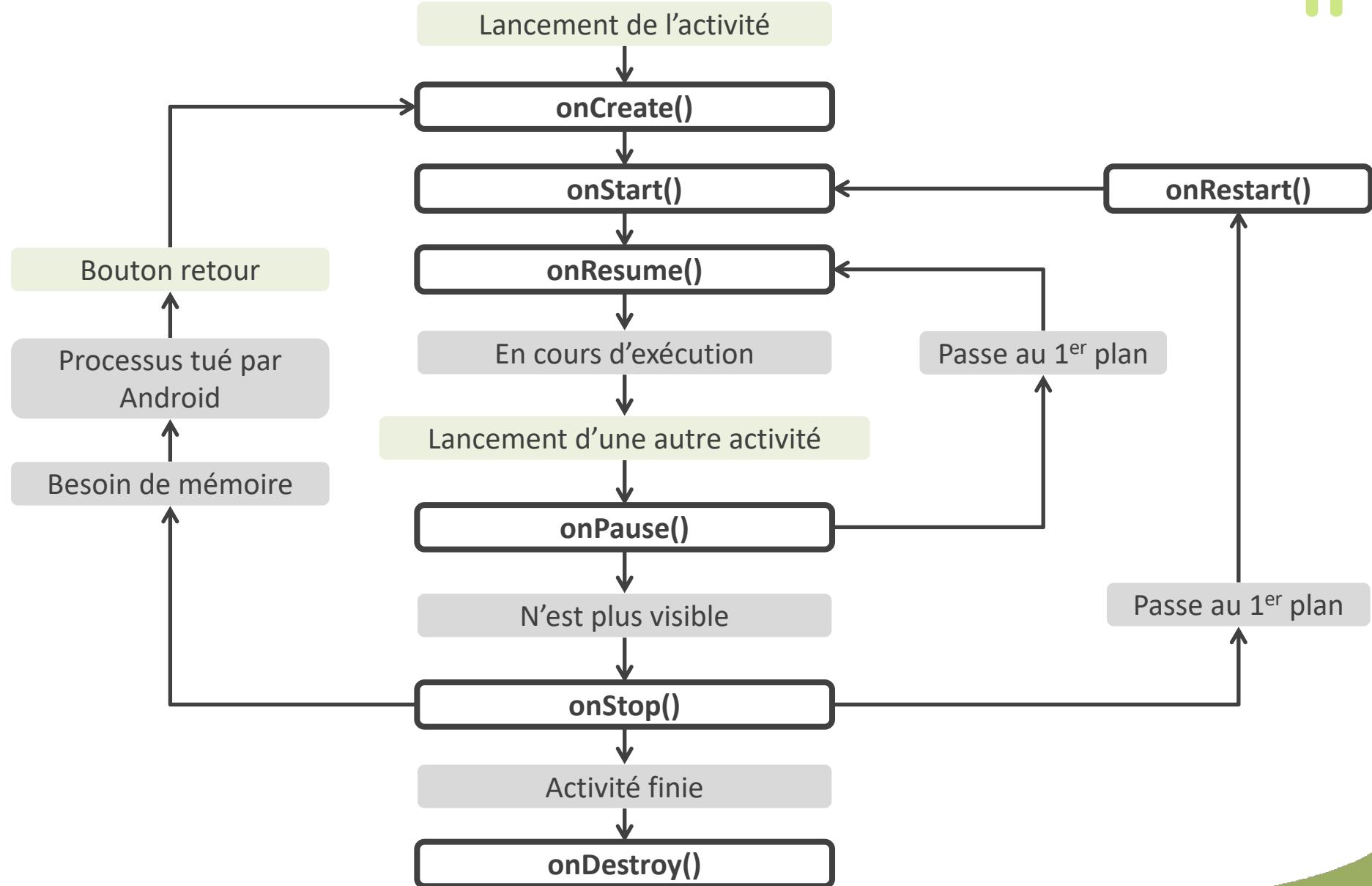


# CYCLE DE VIE D'UNE ACTIVITÉ

Chaque application s'exécute dans un processus séparé.

Afin de libérer des ressources, **Android peut fermer des applications**. Le choix de l'application qui doit être fermée dépend de l'état du processus dans lequel elle se trouve.







Une activité est composée d'une classe Java dans laquelle vous sur-définirez certaines des méthodes.

**onCreate** : permet d'initialiser la vue et lier les données ainsi que de gérer les opérations à faire avant l'affichage de l'activité.

**onStart** : méthode est appelée lorsque l'activité devient visible pour l'utilisateur.

**onResume** : permet d'exécuter tous les traitements nécessaires au fonctionnement de l'activité, d'initialiser des variables et les listeners.

**onPause** : permet d'arrêter tous les traitements effectués par l'activité (traitements non nécessaires si l'activité n'est pas visible).

**onStop** : permet d'arrêter tous les traitements restants qui auraient pas été arrêtés dans onPause.

**onRestart** : cette méthode supplémentaire est appelée quand on relance une activité qui est passée par onStop(). Cela permet de différencier le premier lancement d'un redémarrage.

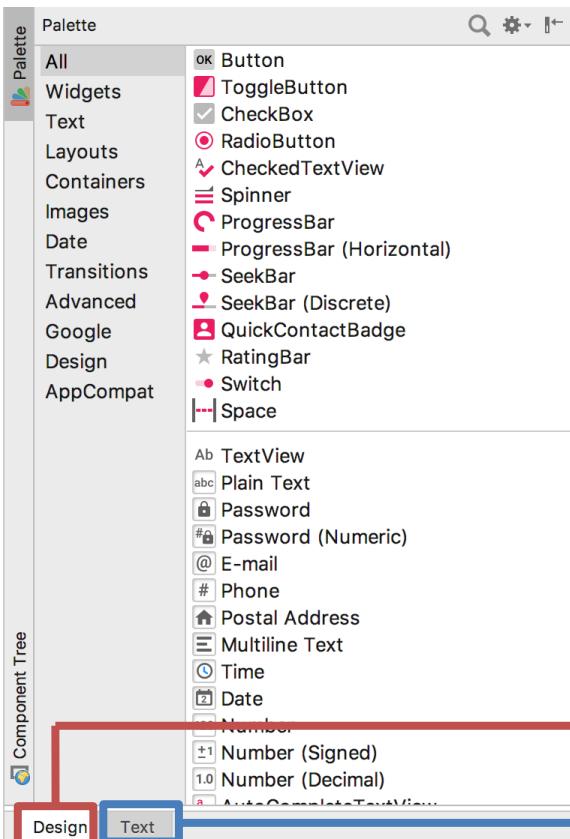
**onDestroy** : permet de gérer la destruction complète de l'activité en mémoire.



# LES VUES

## Interface

Une interface correspondra à une activité qui se compose :



Partie statique (fichier xml), elle est constituée de différents éléments : bouton, texte, ...



Partie dynamique (classe Java), elle permet les interactions avec l'utilisateur et de gérer les traitements à effectuer.

Modifier une vue en mettant à jour :

la vue à l'aide de l'éditeur

le code XML



# LES VUES

## Identifiants

Un identifiant correspond à un nom unique affecté à un élément d'une vue. Il vous permettra de mettre en place les interactions et les traitements sur cet élément.

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

    <Button
        android:id="@+id/button_green"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Green" />

    <Button
        android:id="@+id/button_red"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Red" />
</LinearLayout>
```

android:id nom de l'attribut

@+ indique la déclaration d'un nouvel identifiant

id correspond à la catégorie identifiant

button\_green le nom donné à cet élément



R.id.button\_green



@id/button\_green



# LES VUES

## Taille des éléments

À chaque déclaration d'un élément d'une vue vous devez spécifier sa hauteur et sa largeur.

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

    <Button
        android:id="@+id/button_green"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Green" />

    <Button
        android:id="@+id/button_red"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Red" />
</LinearLayout>
```



# LES VUES

## Taille des éléments

À chaque déclaration d'un élément d'une vue vous devez spécifier sa hauteur et sa largeur.

**android:layout\_height** hauteur

**android:layout\_width** largeur

**match\_parent** la taille de l'élément est égale à celle de l'élément parent

**wrap\_content** la taille de l'élément est égale à celle de son contenu

**spécifier une valeur** la taille peut être définie à l'aide de valeurs fixes

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



# LES VUES

## Combiner avec les activités

Une fois la partie statique d'une interface déclarée, il faut créer une classe Java représentant son activité.

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

Cette classe doit hériter de la classe **AppCompatActivity**.

Redéfinir au minimum la méthode **onCreate**.

Lier l'activité à l'interface à l'aide de la méthode **setContentView**.



# LES VUES

## Combiner avec les activités

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

    <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>
```



N'oubliez pas de déclarer votre activité dans le manifeste de l'application.



# LES RESSOURCES

Afin de maintenir et mettre à jour plus facilement les ressources d'une application, un développeur Android peut les stocker dans des dossiers spécifiques.

Les dossiers contenant des ressources peuvent posséder plusieurs spécificités en fonction de la :

- langue
- largeur minimum de l'écran
- version d'Android
- ...



# LES RESSOURCES

## Drawable

Pour gérer les différentes résolutions des images, vous disposez du groupe de dossier **drawable-xxx** où xxx peut être remplacé par :

- **ldpi** : écrans environ 120 dpi
- **mdpi** : écran environ 160 dpi
- **hdpi** : écran environ 240 dpi
- **xhdpi** : écran environ 320 dpi
- **xxhdpi** : écran environ 480 dpi
- **xxxhdpi** : écran environ 640 dpi
- **nodpi**: ne dépend pas de la résolution de l'écran



# LES RESSOURCES

## Les valeurs

Le dossier **values** permet de stocker les différentes « constantes » utilisées dans l’application.

Les chaînes de caractères sont stockées dans un ou plusieurs fichiers (string.xml par défaut).

```
<resources>
    <string name="mon_string">Le contenu du string</string>
</resources>
```

Pour l’utiliser :



R.string.nom\_string



@string/nom\_string



# LES RESSOURCES

## Les valeurs

Vous pouvez aussi stocker des tableaux de chaînes de caractères

```
<string-array name="days">
    <item>Monday</item>
    <item>Tuesday</item>
    <item>Wednesday</item>
    <item>Thursday</item>
    <item>Friday</item>
    <item>Saturday</item>
    <item>Sunday</item>
</string-array>
```

```
Resources resources = getResources();
```

```
List<String> days = new ArrayList<>(Arrays.asList(resources.getStringArray(R.array.days)));
```



# LES RESSOURCES

## Les valeurs

Il est aussi possible de stocker les couleurs utilisées dans l'application, celles-ci se trouvent dans le fichier colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="red">#F00</color>
</resources>
```

Le dossier values du projet est aussi utilisé pour supporter la langue par défaut d'une application. Si vous souhaitez gérer spécifiquement une langue il faut créer des dossiers tels que :

Français : **values-fr**

Espagnol : **values-es**



# LES PRINCIPAUX ÉLÉMENTS

## Zone d'affichage d'un texte

TextView permet d'afficher un texte à l'aide de son attribut **android:text**.

```
<TextView  
    android:id="@+id/mon_texte"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Écrivez votre texte..."/>
```

Dans le code Java vous pourrez modifier les attributs du TextView, par exemple :

pour changer le fond : **mon\_texte.setBackgroundColor(Color.RED);**

pour changer le texte : **mon\_texte.setText("Nouveau texte");**



# LES PRINCIPAUX ÉLÉMENTS

## Zone d'affichage d'un texte

La gestion de la police s'effectue à l'aide d'attributs classiques :

<TextView

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:text="Hello World!"

    android:textSize="22sp"

    android:fontFamily="sans-serif-condensed"

    android:textStyle="bold" />
```

Taille des polices : android:textSize

Type de police : android:fontFamily

Style de police : android:textStyle



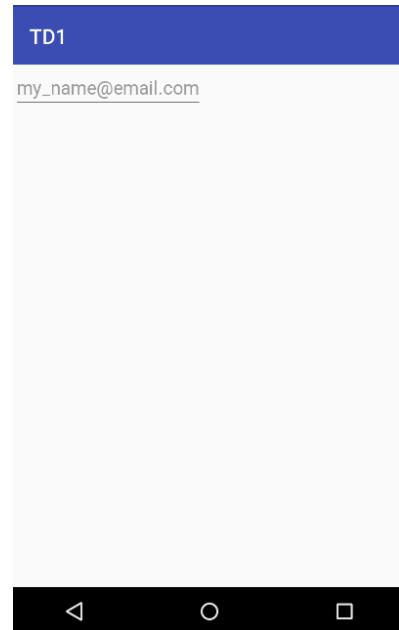
# LES PRINCIPAUX ÉLÉMENTS

## Zone d'édition d'un texte

**EditText** permet à l'utilisateur de saisir un texte.

<EditText

```
    android:id="@+id/zone_saisie"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:inputType="textEmailAddress"  
    android:hint="my_name@email.com" />
```



**android:hint** permet de donner des indications sur le texte attendu.

**android:inputType** permet de préciser un clavier spécifique en fonction du type de texte.



Pour récupérer le texte saisi dans la partie dynamique : `monEditText.getText().toString();`



# LES PRINCIPAUX ÉLÉMENTS

## Image

L'ajout d'une image se fait grâce au code **ImageView**.

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

**android:src** permet de spécifier l'image à afficher



dans le code Java vous pourrez modifier l'image grâce à :

```
image.setImageResource(R.drawable.newImage);
```



# LES PRINCIPAUX ÉLÉMENTS

## Case à cocher

CheckBox permet de créer un case cochable.

### <CheckBox

```
android:id="@+id/isOK"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:checked="false"
android:text="Are you OK?" />
```

Le code à droite vous permettra effectuer une action lorsque le statut de la case changera.

```
public class MainActivity extends AppCompatActivity {

    private CheckBox isOk;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        isOk = findViewById(R.id.isOK);
    }

    @Override
    protected void onStart() {
        super.onStart();
        isOk.setOnCheckedChangeListener(
            new CompoundButton.OnCheckedChangeListener()
            {
                @Override
                public void onCheckedChanged(
                    CompoundButton buttonView,boolean isChecked){

                    // votre code
                }
            });
    }
}
```



# LES PRINCIPAUX ÉLÉMENTS

Les éléments graphiques avec lesquels nous allons interagir dans la partie dynamique seront déclarés en attributs de la classe d'activité (en protected ou private)

Puis instanciés en utilisant leur identifiant unique déclaré dans la partie statique dans la méthode onCreate.

Vous pourrez ainsi interagir avec ces éléments dans l'ensemble des différentes méthodes de la classe.

```
public class MainActivity extends AppCompatActivity {  
    private CheckBox isOk;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        isOk = findViewById(R.id.isOK);  
    }  
  
    @Override  
    protected void onStart() {  
        super.onStart();  
        isOk.setOnCheckedChangeListener(  
            new CompoundButton.OnCheckedChangeListener()  
            {  
                @Override  
                public void onCheckedChanged(  
                    CompoundButton buttonView,boolean isChecked){  
  
                    // votre code  
                }  
            });  
    }  
}
```



# LES PRINCIPAUX ÉLÉMENTS

```
public class MainActivity extends AppCompatActivity {  
  
    private CheckBox isOk;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        isOk = findViewById(R.id.isOK);  
    }  
  
    @Override  
    protected void onStart() {  
        super.onStart();  
        isOk.setOnCheckedChangeListener(  
            new CompoundButton.OnCheckedChangeListener()  
            {  
                @Override  
                public void onCheckedChanged(  
                    CompoundButton buttonView,boolean isChecked){  
  
                    // votre code  
  
                }  
            });  
    }  
}
```

Cette partie de code correspond à l'ajout d'un listener qui permettra de lancer un traitement lorsque l'utilisateur agira sur la CheckBox.



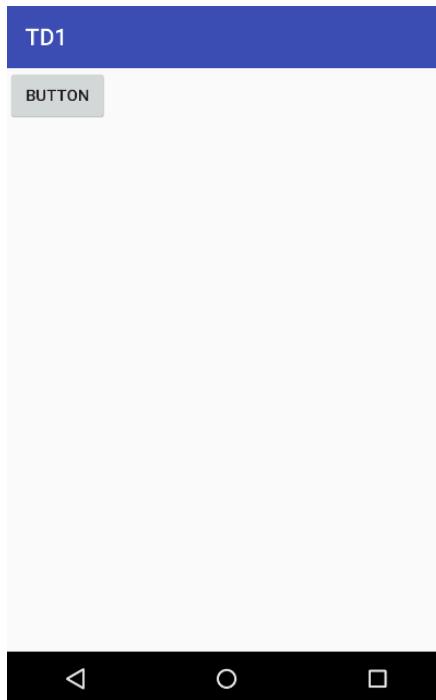
# LES PRINCIPAUX ÉLÉMENTS

## Bouton

**Button** permet de créer un élément de type bouton.

<Button

```
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button" />
```





# LES PRINCIPAUX ÉLÉMENTS

## Gestion du clic

La gestion du clic peut se faire de deux manières différentes :

- gérer le clic sur des boutons séparément
- faire en sorte que l'activité implémente l'interface **onClickListener**



# LES PRINCIPAUX ÉLÉMENTS

## Gestion du clic

### Gérer le clic sur des boutons séparément

```
public class MainActivity extends AppCompatActivity {  
  
    private Button b1;  
    private Button b2;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        b1 = findViewById(R.id.button_1);  
        b2 = findViewById(R.id.button_2);  
    }  
  
    @Override  
    protected void onStart() {  
        super.onStart();  
        b1.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {  
                // Gestion du bouton 1  
            }  
        });  
        b2.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {  
                // Gestion du bouton 2  
            }  
        });  
    }  
}
```

Après avoir été déclarés, les boutons sont initialisés grâce à la méthode **findViewById**.

L'appel à la méthode **setOnClickListener** permet d'ajouter un listener de clic sur les boutons.

Cette méthode prend en argument un nouveau listener qui permet de redéfinir la méthode **onClick**.

Tout le traitement effectué lors du clic sera développé dans la méthode **onClick**.



# LES PRINCIPAUX ÉLÉMENS

## Activité implémentant l'interface OnClickListener

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

    private Button b1;
    private Button b2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        b1 = findViewById(R.id.button_1);
        b2 = findViewById(R.id.button_2);
    }

    @Override
    protected void onStart() {
        super.onStart();
        b1.setOnClickListener(this);
        b2.setOnClickListener(this);
    }

    @Override
    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.button_1:
                // Gestion du bouton 1
                break;
            case R.id.button_2:
                // Gestion du bouton 2
                break;
        }
    }
}
```

Il faut dans un premier temps que l'activité implémente l'interface **OnClickListener**.

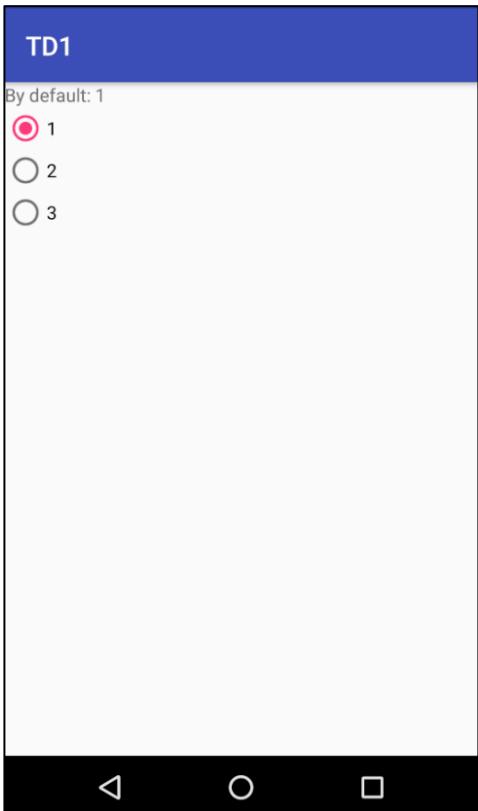
La redéfinition de la méthode **onClick** va permettre de gérer les interactions avec les boutons.

`view.getId()` renverra l'identifiant du bouton sur lequel l'utilisateur aura appuyé et le `switch` permettra de gérer le traitement associé.



# LES PRINCIPAUX ÉLÉMENTS

## RadioButton



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:id="@+id/message"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="By default: 1"/>
    <RadioGroup
        android:id="@+id/group"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
        <RadioButton
            android:id="@+id/rb1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="1"
            android:checked="true"/>
        <RadioButton
            android:id="@+id/rb2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="2" />
        <RadioButton
            android:id="@+id/rb3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="3" />
    </RadioGroup>
</LinearLayout>
```



# LES PRINCIPAUX ÉLÉMENTS

## RadioButton

TD1

3

 1 2 3

```
public class MainActivity extends AppCompatActivity {

    protected RadioGroup group;
    protected TextView message;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        group = findViewById(R.id.group);
        message = findViewById(R.id.message);
    }

    @Override
    protected void onStart() {
        super.onStart();
        group.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(RadioGroup group, int checkedId) {
                RadioButton selected = findViewById(group.getCheckedRadioButtonId());
                message.setText(selected.getText());
            }
        });
    }
}
```





# LES PRINCIPAUX ÉLÉMENTS

## RadioButton

Déclaration des éléments

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

```
    protected RadioGroup group;
    protected TextView message;
```

Instanciation des éléments

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        group = findViewById(R.id.group);
        message = findViewById(R.id.message);
    }
```

Ajout d'un listener sur le RadioGroup. Lorsqu'il est actionné, la méthode `onCheckedChanged` sera invoquée et nous récupérerons une instance du RadioButton sélectionné.

Nous affecterons dans le TextView le texte du RadioButton.

```
    @Override
    protected void onStart() {
        super.onStart();
        group.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(RadioGroup group, int checkedId) {
                RadioButton selected = findViewById(group.getCheckedRadioButtonId());
                message.setText(selected.getText());
            }
        });
    }
```



# LES PRINCIPAUX ÉLÉMENTS

## Toasts

Permet d'afficher un message court ne nécessitant pas d'interaction avec l'utilisateur.

```
Toast.makeText(MainActivity.this, "Message", Toast.LENGTH_LONG).show();
```

Message

La méthode **makeText** prend en paramètres :

le contexte

le message à afficher

la durée d'affichage (**Toast.LENGTH\_SHORT** ou **Toast.LENGTH\_LONG**)

La méthode **show** permet d'afficher le Toast.



# LES PRINCIPAUX LAYOUTS

Afin de faciliter l'organisation des différents éléments qui composent une interface, nous pouvons utiliser des conteneurs : les layouts.



# LES PRINCIPAUX LAYOUTS

## LinearLayout

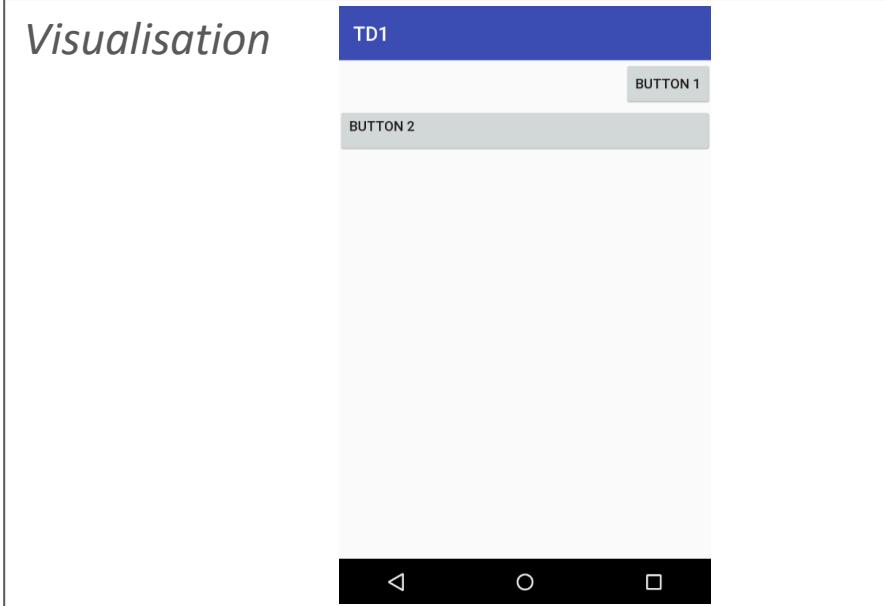
### *Exemple de code*

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

    <Button
        android:id="@+id/button_1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="Button 1" />

    <Button
        android:id="@+id/button_2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="left"
        android:text="Button 2" />
</LinearLayout>
```

### *Visualisation*



Permet d'aligner, horizontalement ou verticalement, les éléments dans l'ordre de déclaration.

**android:orientation** pour préciser l'orientation  
**layout\_gravity** position d'un élément dans le conteneur  
**gravity** position du contenu d'un élément



# LES PRINCIPAUX LAYOUTS

## LinearLayout

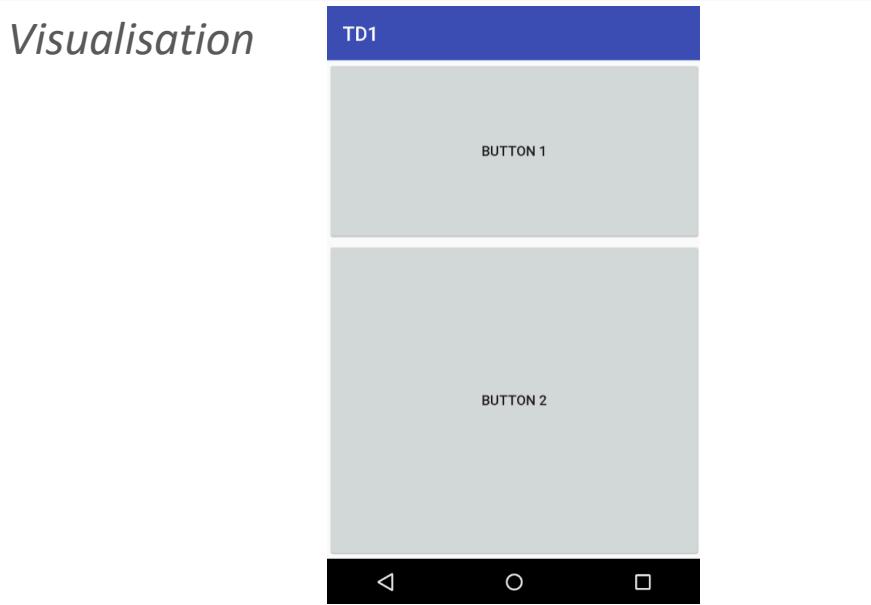
*Exemple de code*

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:weightSum="3">

    <Button
        android:id="@+id/button_1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button 1" />

    <Button
        android:id="@+id/button_2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:text="Button 2" />
</LinearLayout>
```

### *Visualisation*



**android:layout\_weight** permet d'indiquer à un élément l'espace qu'il peut occuper. La taille de la zone à étendre doit être définie à **wrap\_content**

*Dans l'exemple le poids du bouton 2 est deux fois plus grand que celui du bouton 1, il occupera donc les  $\frac{2}{3}$  de l'espace.*



# LES PRINCIPAUX LAYOUTS

## LinearLayout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1">
        <Button
            android:id="@+id/button"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:text="Button 1" />
    </FrameLayout>
    <ImageView
        android:adjustViewBounds="true"
        android:id="@+id/image"
        android:src="@drawable/aquarius"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"/>
    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1">
        <Button
            android:id="@+id/button2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:text="Button 2" />
    </FrameLayout>
</LinearLayout>
```

### Visualisation



La gestion des images peut poser problème, notamment lorsque que l'on veut affecter des poids dessus.

Le code de cette page permet de répartir l'écran en trois parties égales et que l'image s'adapte à l'espace disponible.



# LES PRINCIPAUX LAYOUTS

## ConstraintLayout



Pour l'utiliser, ajouter dans les dépendances du Gradle la ligne suivante :

implementation "androidx.constraintlayout:constraintlayout:2.0.0-beta4"

### Exemple de code

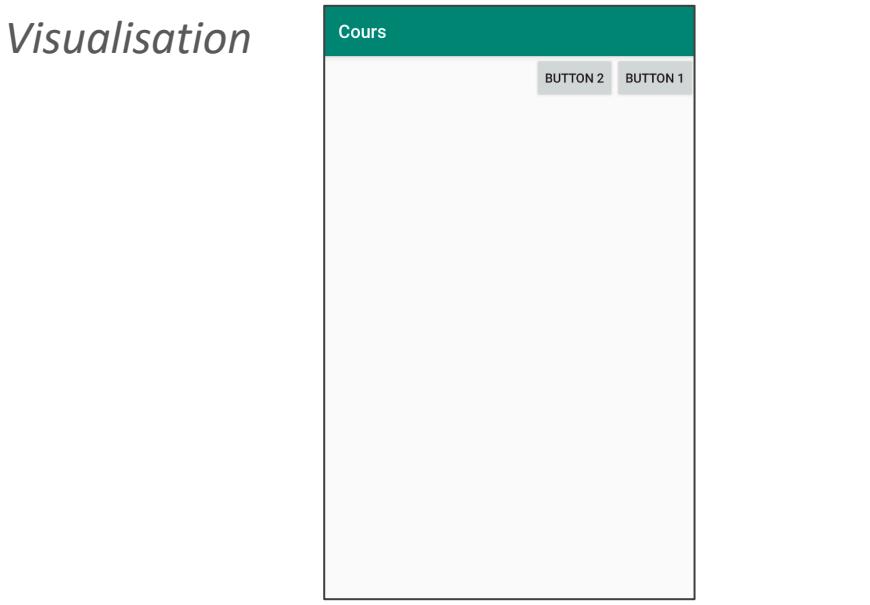
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/button_1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:text="Button 1"/>

    <Button
        android:id="@+id/button_2"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintRight_toLeftOf="@+id/button_1"
        android:text="Button 2"/>

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

### Visualisation



Permet de placer les éléments les uns en fonction des autres.



# LES PRINCIPAUX LAYOUTS

## ConstraintLayout

Permet de placer les éléments les uns en fonction des autres :

- layout\_constraintLeft\_toLeftOf
- layout\_constraintLeft\_toRightOf
- layout\_constraintRight\_toLeftOf
- layout\_constraintRight\_toRightOf
- layout\_constraintTop\_toTopOf
- layout\_constraintTop\_toBottomOf
- layout\_constraintBottom\_toTopOf
- layout\_constraintBottom\_toBottomOf
- layout\_constraintBaseline\_toBaselineOf
- layout\_constraintStart\_toEndOf
- layout\_constraintStart\_toStartOf
- layout\_constraintEnd\_toStartOf
- layout\_constraintEnd\_toEndOf



# LES PRINCIPAUX LAYOUTS

## ConstraintLayout

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

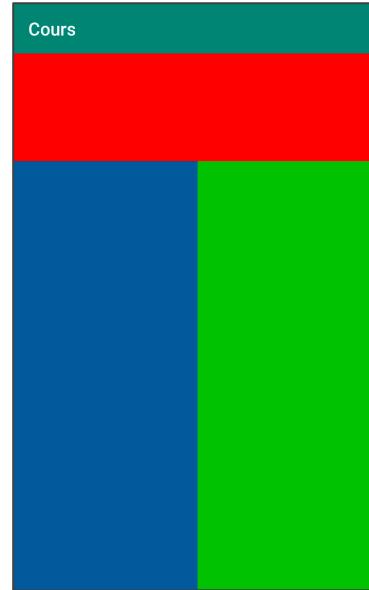
    <TextView
        android:id="@+id/textView_1"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        app:layout_constraintHeight_default="percent"
        app:layout_constraintHeight_percent="0.2"
        android:background="@color/RED"/>

    <TextView
        android:id="@+id/textView_2"
        app:layout_constraintTop_toBottomOf="@+id/textView_1"
        app:layout_constraintLeft_toLeftOf="parent"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintWidth_default="percent"
        app:layout_constraintWidth_percent="0.5"
        app:layout_constraintHeight_default="percent"
        app:layout_constraintHeight_percent="0.8"
        android:background="@color/BLUE"/>

    <TextView
        android:id="@+id/textView_3"
        app:layout_constraintTop_toBottomOf="@+id/textView_1"
        app:layout_constraintRight_toRightOf="parent"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintWidth_default="percent"
        app:layout_constraintWidth_percent="0.5"
        app:layout_constraintHeight_default="percent"
        app:layout_constraintHeight_percent="0.8"
        android:background="@color/GREEN" />

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

### Visualisation



Permet de donner des poids aux différents éléments.



# LES PRINCIPAUX LAYOUTS

## GridLayout

### *Exemple de code*

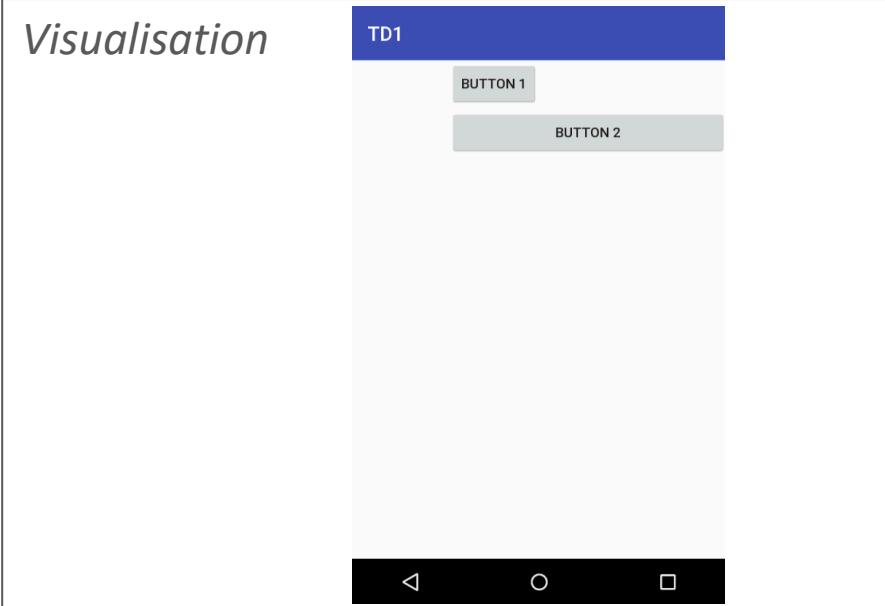
```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="3"
    android:rowCount="3">

    <Space
        android:layout_width="100dp"
        android:layout_height="50dp"
        android:layout_column="0"
        android:layout_row="0"/>

    <Button
        android:id="@+id/button_1"
        android:layout_column="1"
        android:layout_row="0"
        android:text="Button 1" />

    <Button
        android:id="@+id/button_2"
        android:layout_column="1"
        android:layout_row="1"
        android:layout_columnSpan="2"
        android:layout_gravity="fill_horizontal"
        android:text="Button 2" />
</GridLayout>
```

### *Visualisation*



Permet de diviser l'écran en lignes et en colonnes, permettant de laisser des emplacements vides dans une interface.

**columnCount** et **rowCount** nb colonnes et lignes

**layout\_column** et **layout\_row** positionnement

**columnSpan** et **rowSpan** étendre sur plusieurs

**Space** emplacement vide



# LES LAYOUTS AVEC ADAPTER

## Liste : création

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

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

</LinearLayout>
```

```
public class MainActivity extends AppCompatActivity {

    private List<String> name_degree = new ArrayList<>(Arrays.asList(
        "Bachelor",
        "Master",
        "Doctorate"));
    protected ListView list_degree;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        list_degree = findViewById(R.id.my_list);
    }

    @Override
    protected void onStart() {
        super.onStart();
        ArrayAdapter<String> adapter = new ArrayAdapter<(this, android.R.layout.simple_list_item_1, name_degree);
        list_degree.setAdapter(adapter);
    }
}
```



# LES LAYOUTS AVEC ADAPTER

## Liste : gestion du clic

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

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

    <TextView
        android:id="@+id/selected_degree"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

```
public class MainActivity extends AppCompatActivity {

    private List<String> name_degree = new ArrayList<>(Arrays.asList(
        "Bachelor",
        "Master",
        "Doctorate"));
    protected ListView list_degree;
    protected TextView message;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        list_degree = findViewById(R.id.my_list);
        message = findViewById(R.id.selected_degree);
    }

    @Override
    protected void onStart() {
        super.onStart();
        final ArrayAdapter<String> adapter = new ArrayAdapter<(this, android.R.layout.simple_list_item_1,
            name_degree);
        list_degree.setAdapter(adapter);
        list_degree.setOnItemClickListener(
            new AdapterView.OnItemClickListener() {
                @Override
                public void onItemClick(AdapterView<?> parent,
                    View view, int position, long id) {
                    String selection =
                        (String) parent.getItemAtPosition(position);
                    message.setText(selection);
                }
            });
    }
}
```



# LES LAYOUTS AVEC ADAPTER

## Liste

Pour aller plus loin dans la maîtrise des listes :

**ViewHolder** : permet de mettre la vue représentant chaque ligne en cache et ainsi éviter de recharger inutilement.

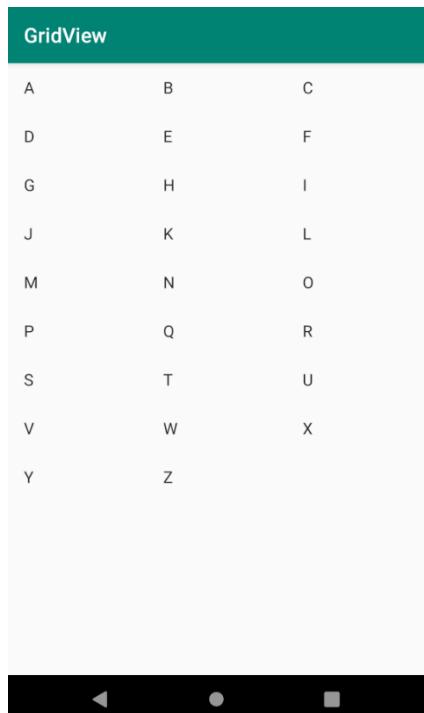
**RecyclerView** : depuis la version 5 d'Android ce composant permet un affichage plus performant lorsque les données sont nombreuses.



# LES LAYOUTS AVEC ADAPTER

## GridView

Malgré sa ressemblance avec un GridLayout, un GridView est un layout dynamique comme une liste ListView. Sa principale différence réside dans le fait qu'il est scrollable. Un GridView obtiendra ses données depuis un ListArray (seules les données chargées en mémoire seront affichées).





# LES LAYOUTS AVEC ADAPTER

## GridView

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

    <GridView
        android:id="@+id/gridView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:numColumns="3"/>

</LinearLayout>
```

```
public class MainActivity extends AppCompatActivity {

    private GridView gridView;
    private ArrayAdapter<String> adapter;

    private final List<String> letters = new ArrayList<>(Arrays.asList(
        "A", "B", "C", "D", "E",
        "F", "G", "H", "I", "J",
        "K", "L", "M", "N", "O",
        "P", "Q", "R", "S", "T",
        "U", "V", "W", "X", "Y", "Z"));

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



# DÉBOGAGE

## Débogage pas à pas

Permet de positionner des points d'arrêt dans le code source, d'analyser pas à pas le déroulement du programme et connaître, entre autres, la pile d'appels des méthodes et les valeurs des variables.

Pour positionner un point d'arrêt dans le code, il suffit de cliquer dans la marge de la fenêtre principale (l'arrêt sera matérialisé par un point rouge).

Cliquer sur ce point permet de supprimer le point d'arrêt.

```
protected void onResume() {
    super.onResume();
    calculate.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Double division = 0.;
            Double n = Double.parseDouble(numerator.getText().toString());
            Double d = Double.parseDouble(denominator.getText().toString());
            division = n / d;
            result.setText(division.toString());
        }
    });
}
```

Debug: Android Debugger (8600)

Name	Hits	Time (ms)
Line 33 ir 1	51	
Line 36 ir 1	16	
Line 35 ir 1	5	
toString re 19	2	



# DÉBOGAGE

## Débogage pas à pas

Pour lancer en mode pas à pas, il faut soit utiliser le bouton représentant un bug

The screenshot shows the Android Studio interface during a debug session. The top bar displays the project name 'MyApplication' and the file 'activity\_main.xml'. The code editor shows Java code for 'MainActivity.java' with several breakpoints set (indicated by red circles). The bottom part of the interface is the 'Debug' tool window, which includes tabs for 'Frames' and 'Threads'. The 'Variables' tab shows the message 'Variables are not available'. The 'Overhead' tab displays a table of hit counts:

Name	Hits	Time (ms)
Line 33 ir 1	51	
Line 36 ir 1	16	
Line 35 ir 1	5	
toString rer 19	2	

The status bar at the bottom indicates 'DISCONNECTED FROM THE TARGET VM, ADDRESS: localhost:8600, TRANSPORT: SOCKET'.



# DÉBOGAGE

## Débogage pas à pas

The screenshot shows the Android Studio interface during a step-by-step debug session. The code editor displays the `MainActivity.java` file with several breakpoints set. The current line of execution is highlighted at line 35. The bottom navigation bar indicates the current state is "Debug". The "Variables" tool window shows local variables and their values, including `this`, `v`, `division`, `n`, `result`, `numerator`, and `denominator`. The "Overhead" table in the bottom right corner provides performance metrics for the current frame.

Name	Hits	Time (ms)
Line 33 ir 1	69	
Line 35 ir 1	7	
toString rei 7	1	



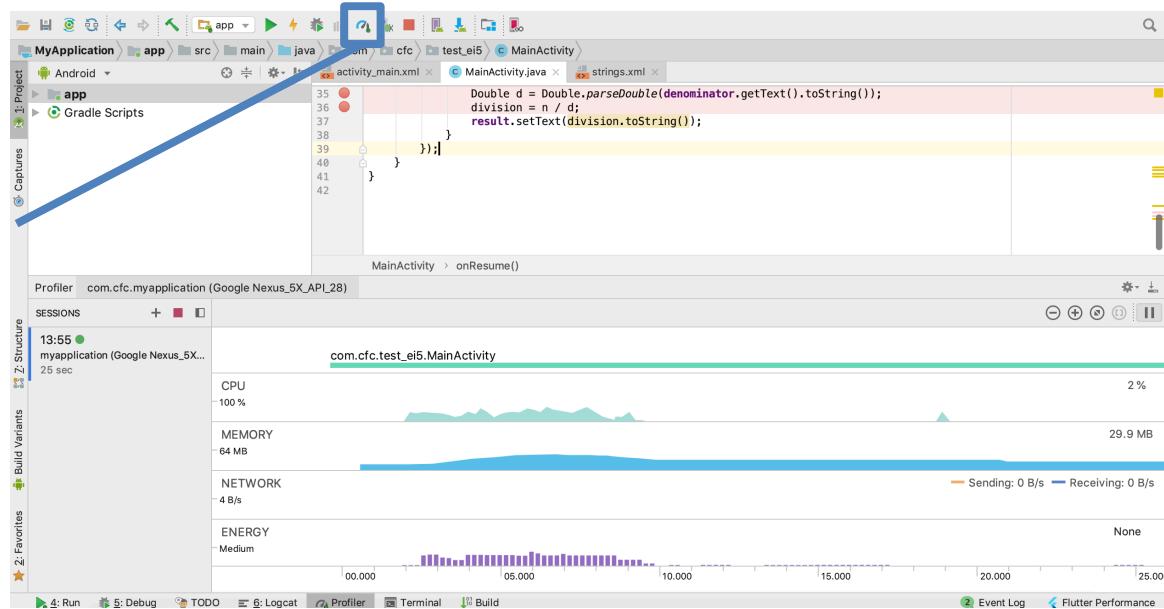
# DÉBOGAGE

## Android Profiler

Permet au développeur d'analyser finement le fonctionnement de son application selon différents points de vue :

- une vue du taux d'occupation du processeur ;
- une vue de la mémoire allouée ;
- une vue du trafic réseau.

Pour lancer Android Profiler il faut cliquer sur l'icône Profile app





**PM\_EI5\_TD1\_EXO1.pdf**