

Programmation et interfaces Android

Interfaces utilisateurs

Sommaire

1. Les Activity (principes)
2. Les Activity (Cycle de vie)
3. Les Activity (Hello Android World)
4. Les Layouts (principes)
5. Les Layouts (types)
6. Les Layouts (attributs)
7. Les Widgets (Basique)
8. Interaction sur nos widgets
9. Un peu d'exercice



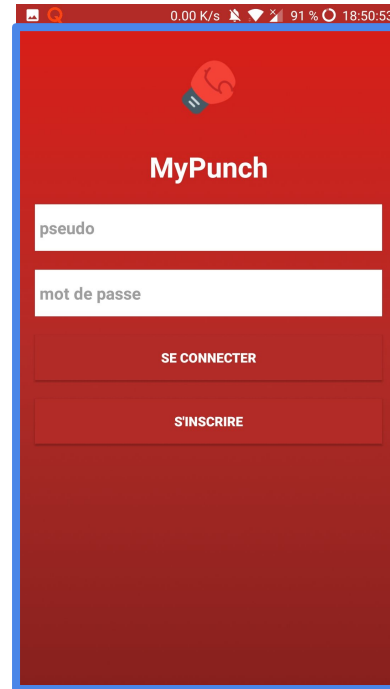
Les Activity (principes)

- Les activités sont le système d'écran principal d'une application Android (elle font office de conteneur)



Les Activity (principes)

- Les activités sont le système d'écran principal d'une application Android (elle font office de conteneur)

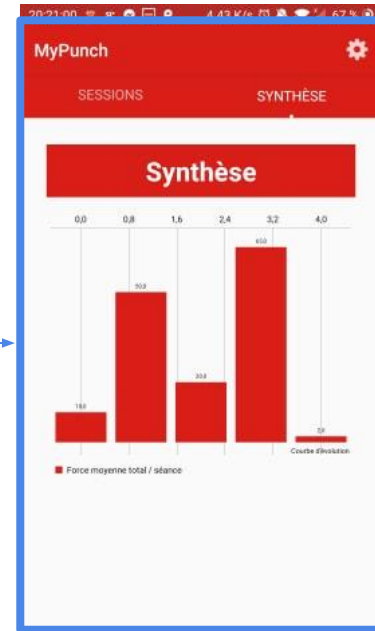
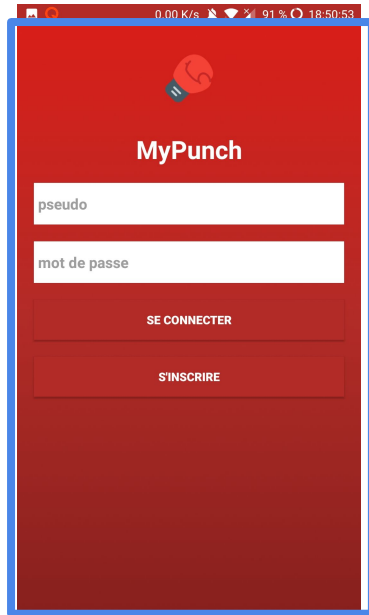


Activity



Les Activity (principes)

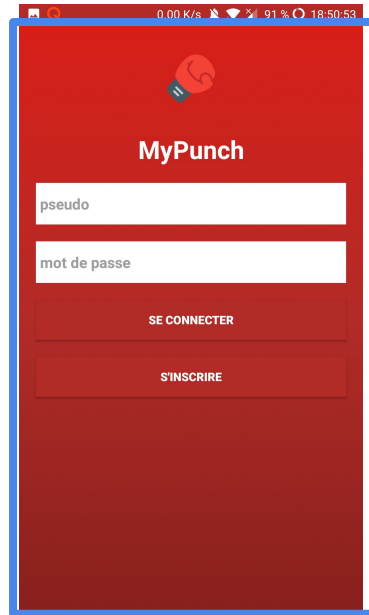
- Par exemple je me connecte sur l'Activity "LoginActivity" celle-ci me renvoi sur une autre activity "SyntheseActivity"



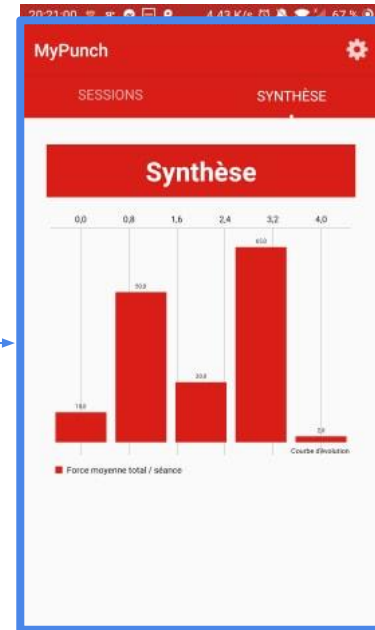


Les Activity (principes)

- Par exemple je me connecte sur l'Activity "LoginActivity" celle-ci me renvoi sur une autre activity "SyntheseActivity"



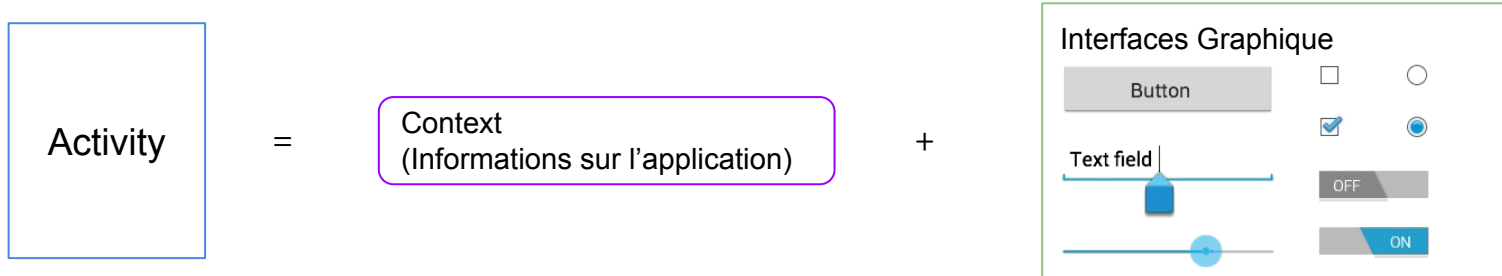
Le système ne vous permet d'afficher qu'une seule activity à la fois !





Les Activity (principes)

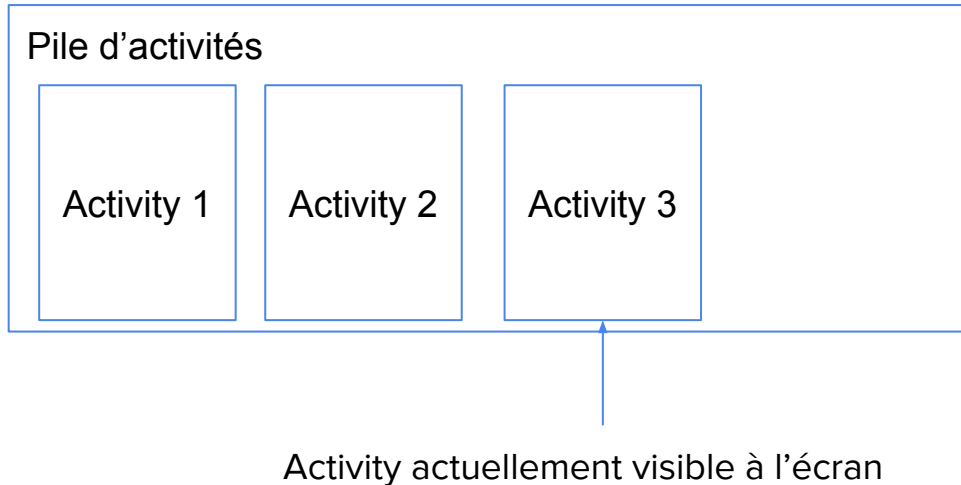
- De plus, une activity, s'exécute dans ce qu'on appelle un **context** qui contient des informations sur l'état de votre application et qui représente un lien avec le système Android





Les Activity (principes)

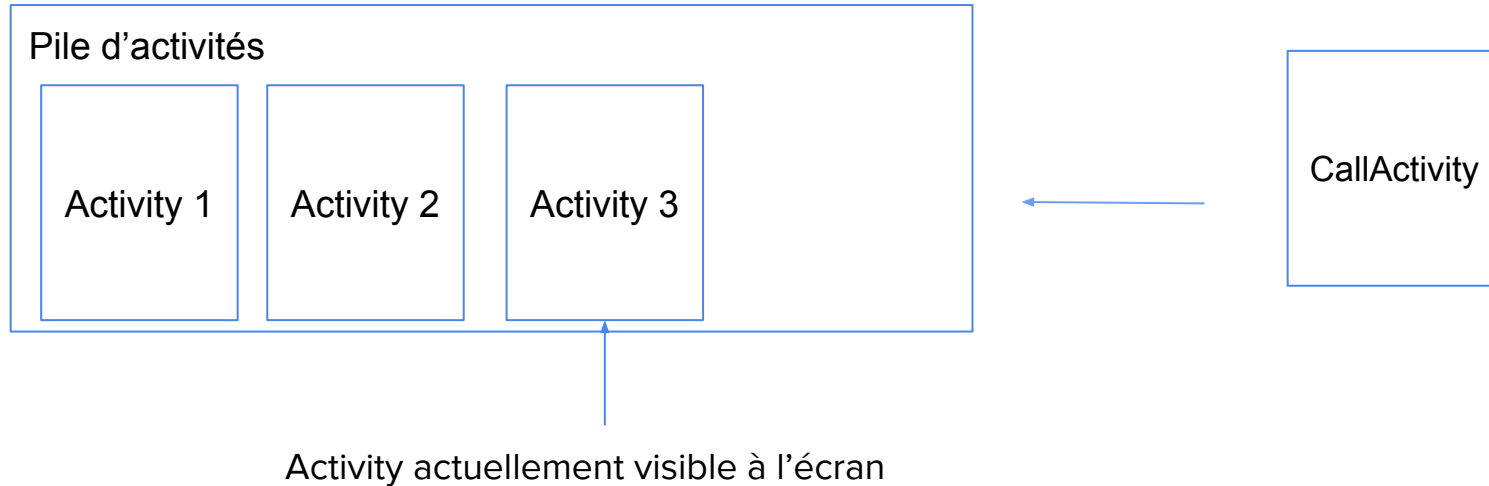
- Cependant, même si une seule activity est visible à la fois, rien n'empêche d'empiler les activity en arrière plan ! Elle se stack dans ce qu'on appelle la ***pile d'activités***





Les Activity (principes)

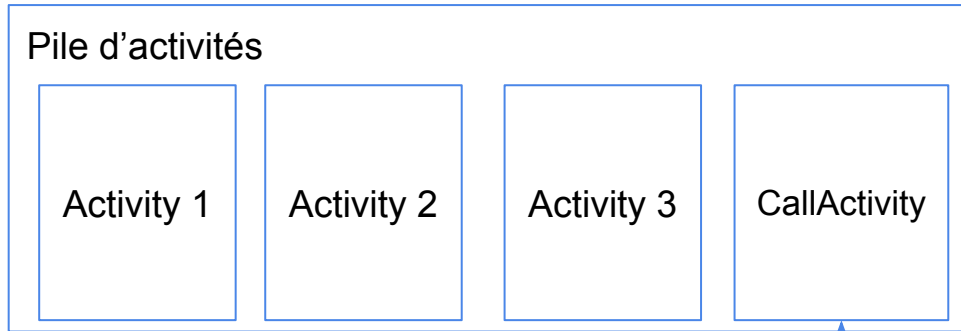
- Exemple si je reçois un appel pendant que j'utilise une application
 - Réception de l'appel dans le téléphone





Les Activity (principes)

- Exemple si je reçois un appel pendant que j'utilise une application
 - Téléphone qui affiche un écran me permettant de décrocher ou de raccrocher

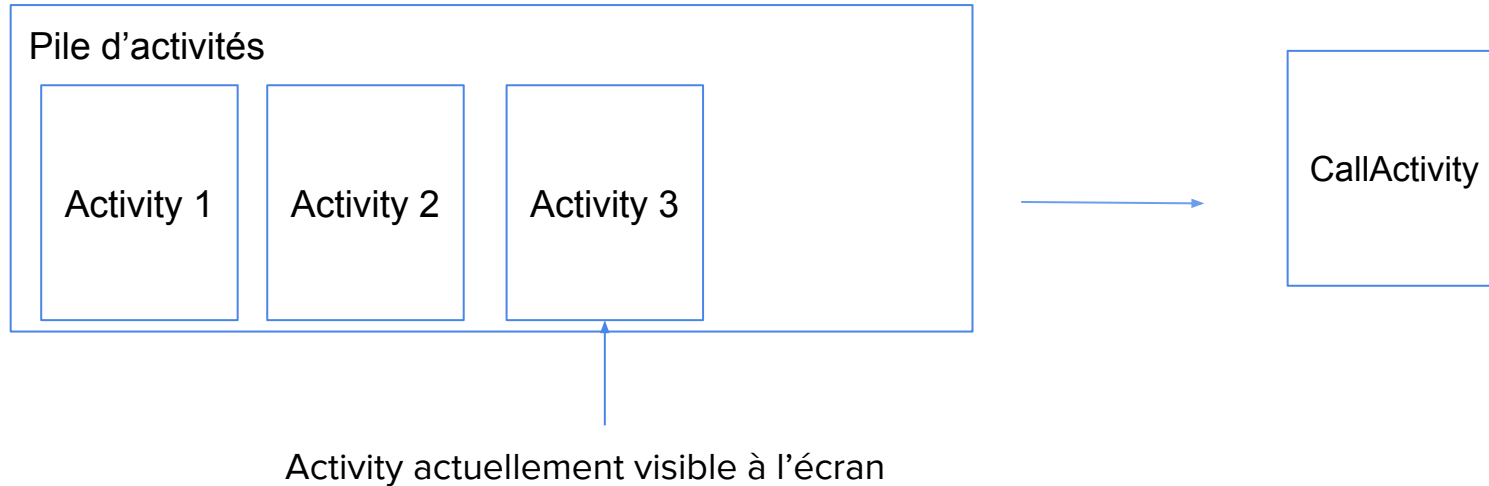


Activity actuellement visible à l'écran



Les Activity (principes)

- Exemple si je reçois un appel pendant que j'utilise une application
 - Fin de mon appel l'écran repart





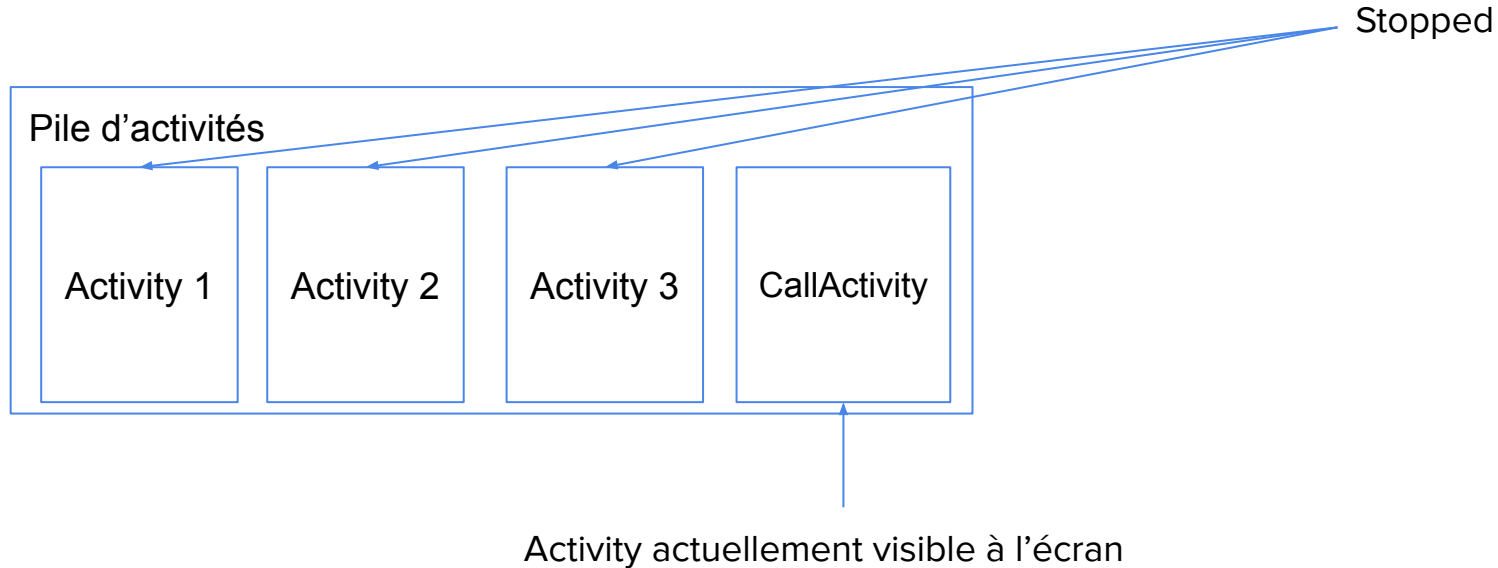
Les Activity (cycle de vie)

- Mais que ce passe t-il lorsque mon Activity n'est plus visible ?



Les Activity (cycle de vie)

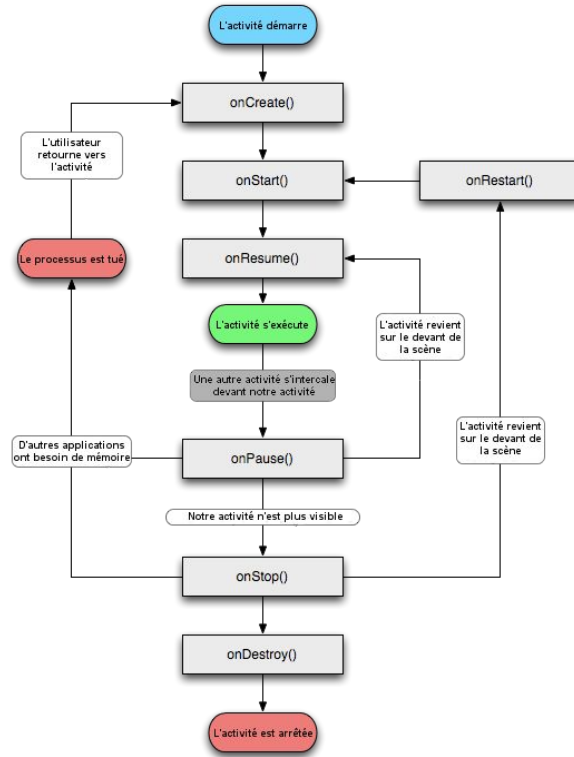
- Elle sont dans un état “Stopped”, donc elle existe mais est “figé”





Les Activity (cycle de vie)

- Pour mieux comprendre

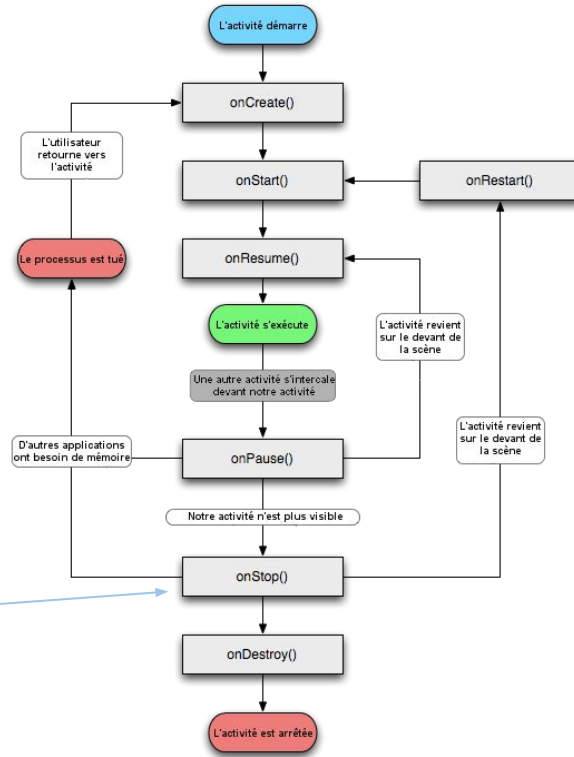




Les Activity (cycle de vie)

- Pour mieux comprendre

l'activity passe par de nombreux états !



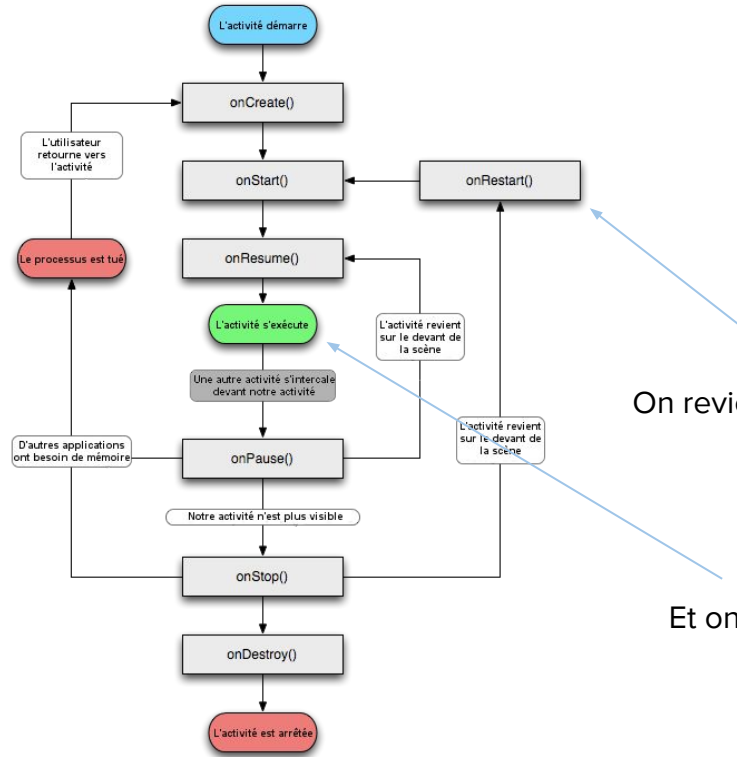
Dans notre cas elle se trouve ici



Les Activity (cycle de vie)

- Pour mieux comprendre

Lorsque l'on raccroche, et qu'on retrouve notre application



On revient par là

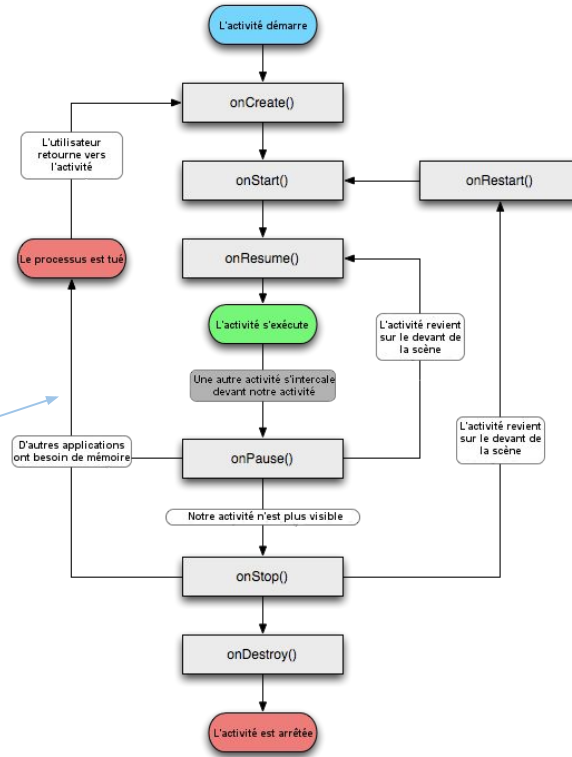
Et on revient ici



Les Activity (cycle de vie)

- Pour mieux comprendre

Ce chemin est emprunté dans le cas où l'on retourne l'écran, ou lorsque le système kill l'application par soucis de mémoire

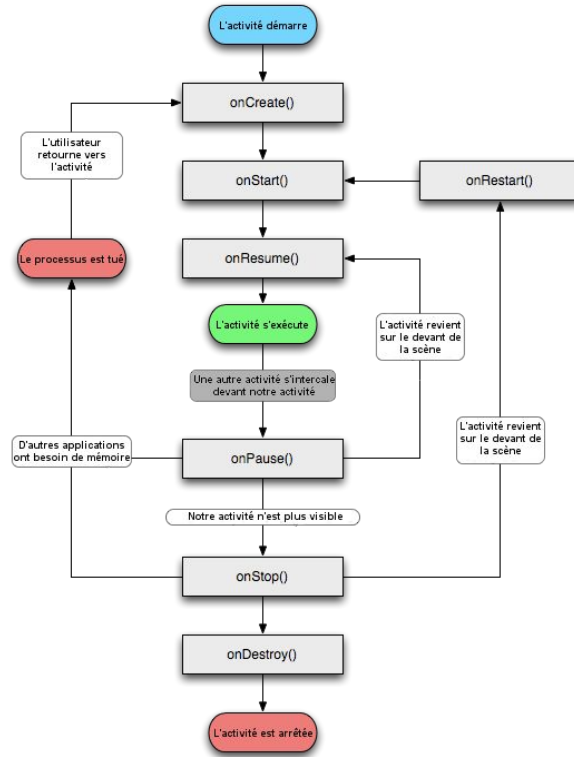




Les Activity (cycle de vie)

- Pour mieux comprendre

En Android ce cycle est important !
Notamment pour éviter pas mal de crash





Les Activity (Hello Android World)

- Et si on s'entraînait avec un petit exo ?



Les Activity (Hello Android World)

- Et si on s'entraînait avec un petit exo ?
 - Sur Android Studio faire une activity qui affiche les différents états du cycle de vie :
 - Pour cela utiliser la fonction **d()** de l'objet **Log** (**Log.d()**)
 - Chaque état du cycle de vie est écouté par l'objet Activity ou AppCompatActivity et peut-être récupéré dans votre Activity via **l'override** tout comme la fonction onCreate ;)



Les Activity (Hello Android World)

- Quelques tips !



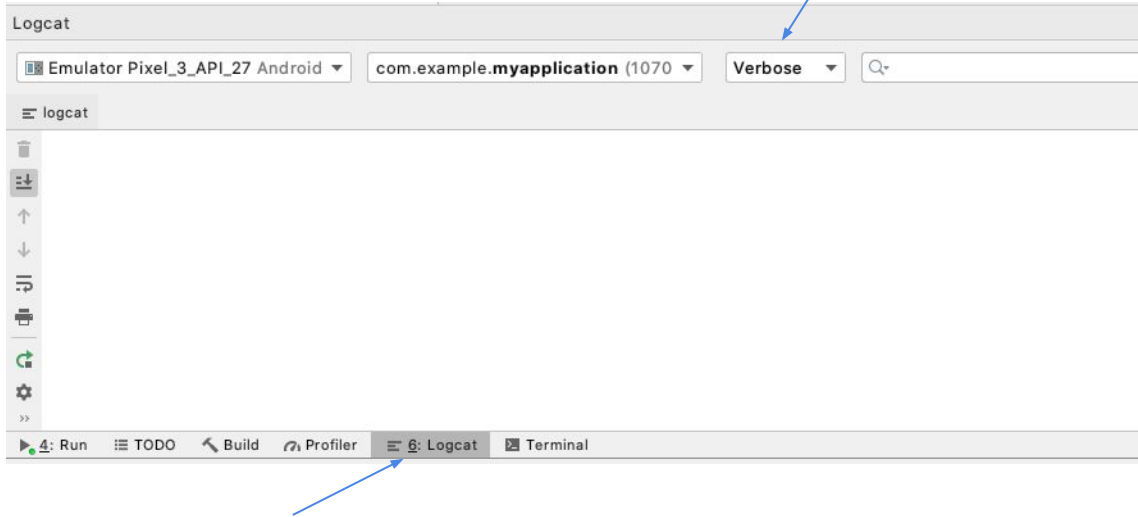
Vous pouvez créer une Activity en faisant un clic droit ici



Les Activity (Hello Android World)

- Quelques tips !

Si vous avez utilisé **Log.d()**
sélectionner **debug** ici



Vous pouvez faciliter la lecture en entrant ici le premier String que vous avez setup à la fonction **Log.d()**

Vous pouvez afficher les logs en cliquant ici



Les Layouts (principes)

- En Android la feuille de style (feuille de dessin qui représente graphiquement votre écran) est écrite en XML et s'appelle un Layout



Les Layouts (principes)

- C'est dans un Layout qu'on dessine notre interface graphique

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

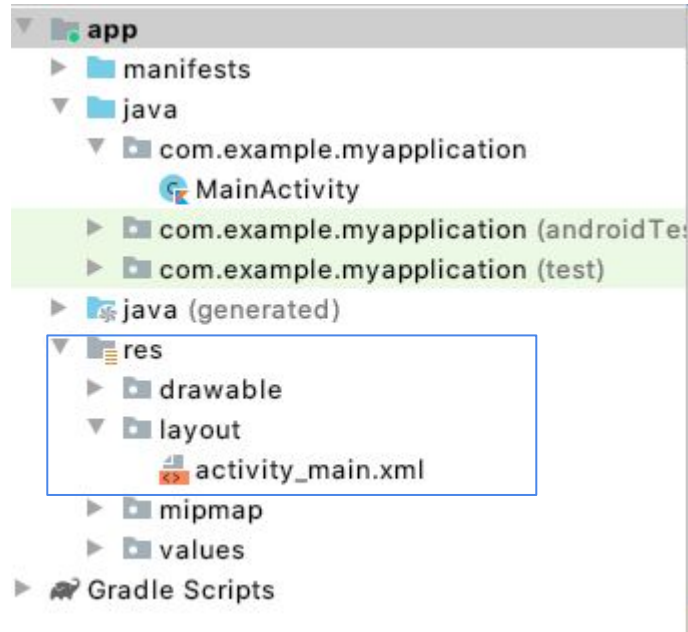
</androidx.constraintlayout.widget.ConstraintLayout>
```

Hello World!



Les Layouts (principes)

- On les trouve et on les crée dans le répertoire res -> layout





Les Layouts (principes)

- On les attribue ensuite à une activity

```
class MainActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
    }  
}
```



Les Layouts (types)

- Ils existe plusieurs type de Layout mais les plus intéressant sont :
 - Le LinearLayout : Tout sur une ligne
 - Le ConstraintLayout : Tout est en lien !
 - TableLayout : Un tableau quoi...
 - FrameLayout : Le différent
 - ScrollView : Du contenue à l'infini



Les Layouts (types)

- Le LinearLayout : Tout sur une ligne

Comme son nom l'indique le LinearLayout permet d'afficher les widgets les uns à la suite des autres en fonction d'une **orientation**



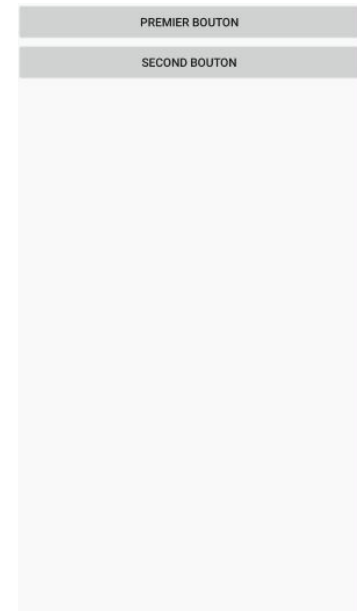
Les Layouts (types)

- Le LinearLayout : Tout sur une ligne

Comme son nom l'indique le LinearLayout permet d'afficher les widgets les uns à la suite des autres en fonction d'une **orientation**

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <Button
        android:id="@+id/premier"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Premier bouton" />

    <Button
        android:id="@+id/second"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Second bouton" />
</LinearLayout>
```





Les Layouts (types)

- Le LinearLayout : Tout sur une ligne

Comme son nom l'indique le LinearLayout permet d'afficher les widgets les uns à la suite des autres en fonction d'une **orientation**

Avantages :

- Il est performant
- Il est simple d'utilisation

Inconvénients :

- Il est très peu flexible



Les Layouts (types)

- Le ConstraintLayout : Tout est en lien !

Comme son nom l'indique le ConstraintLayout permet d'afficher les widgets les uns par rapport aux autres grâce à des **constraint**



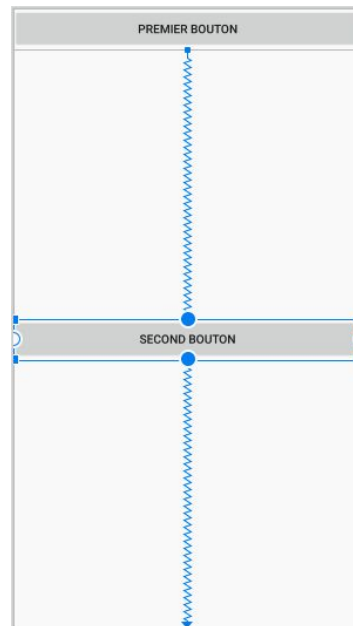
Les Layouts (types)

- Le ConstraintLayout : Tout est en lien !

Comme son nom l'indique le ConstraintLayout permet d'afficher les widgets les uns par rapport aux autres grâce à des **constraint**

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/premier"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Premier bouton"
        app:layout_constraintTop_toTopOf="parent"/>

    <Button
        android:id="@+id/second"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Second bouton"
        app:layout_constraintTop_toBottomOf="@id/premier"
        app:layout_constraintBottom_toBottomOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```





Les Layouts (types)

- Le ConstraintLayout : Tout est en lien !

Comme son nom l'indique le ConstraintLayout permet d'afficher les widgets les uns par rapport aux autres grâce à des **constraint**

Avantages :

- Il permet une grande flexibilité

Inconvénients :

- Il est un peu plus complexe à mettre en place pour de gros layout
- Il est un peu moins performant que le LinearLayout (mais bien plus que le RelativeLayout)



Les Layouts (types)

- TableLayout : Un tableau quoi...

C'est le dernier des trois layouts de bases, il permet d'organiser son layout comme un tableau grâce à des **TableRow** comme en HTML



Les Layouts (types)

- TableLayout : Un tableau quoi...

C'est le dernier des trois layouts de bases, il permet d'organiser son layout comme un tableau grâce à des **TableRow** comme en HTML

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1">
    <TextView
        android:text="Les items précédés d'un V ouvrent un sous-menu"
    />
    <View
        android:layout_height="2dip"
        android:background="#FF909090"
    />
    <TableRow>
        <TextView
            android:text="N'ouvre pas un sous-menu"
            android:layout_column="1"
            android:padding="3dip"
        />
        <TextView
            android:text="Non !"
            android:gravity="right"
            android:padding="3dip"
        />
    </TableRow>
</TableLayout>
```



Les items précédés d'un V ouvrent un sous-menu	
N'ouvre pas un sous-menu	Non !



Les Layouts (types)

- TableLayout : Un tableau quoi...

C'est le dernier des trois layouts de bases, il permet d'organiser son layout comme un tableau grâce à des **TableRow** comme en HTML

Avantages :

- Il permet de produire facilement un tableau

Inconvénients :

- Il peut-être rapidement complexe à lire



Les Layouts (types)

- FrameLayout : Le différent

Le FrameLayout est un peu différent, il est optimisé dans l'affichage d'un seul élément à la fois (pour un album photo par exemple)



Les Layouts (types)

- **FrameLayout** : Le différent

Le **FrameLayout** est un peu différent, il est optimisé dans l'affichage d'un seul élément à la fois (pour un album photo par exemple)

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/colorPrimary"
    android:orientation="vertical">

    <ImageView
        android:id="@+id/backgroundImage"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="centerCrop"
        android:src="@drawable/ic_launcher_foreground" />

    <TextView
        android:id="@+id/descTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        android:background="@android:color/ho_lo_blue_light"
        android:padding=10dp
        android:text="TextView placed at the top of the ImageView"
        android:textColor="@android:color/white"
        android:textSize="22sp" />

</FrameLayout>
```





Les Layouts (types)

- `FrameLayout` : Le différent

Le `FrameLayout` est un peu différent, il est optimisé dans l'affichage d'un seul élément à la fois (pour un album photo par exemple)

Avantages :

- Il est optimiser pour l'affichage d'élément superposé avec un seul élément visible à la fois

Inconvénients :

- Il n'est pas évident de savoir quand l'utiliser ou ne pas l'utiliser



Les Layouts (types)

- ScrollView : Du contenu à l'infini

La ScrollView (est bien un layout), elle apporte une notion de scroll à un autre layout, si son contenu dépasse l'écran



Les Layouts (types)

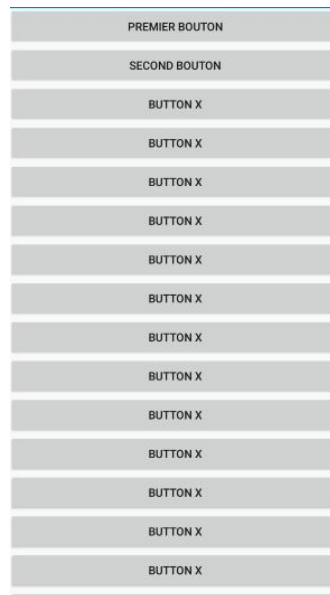
- ScrollView : Du contenu à l'infini

La ScrollView (est bien un layout), elle apporte une notion de scroll à un autre layout, si son contenu dépasse l'écran

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

        <!-- Here your content -->
    </LinearLayout>

</ScrollView>
```





Les Layouts (types)

- ScrollView : Du contenu à l'infini

La ScrollView (est bien un layout), elle apporte une notion de scroll à un autre layout, si son contenu dépasse l'écran

Avantages :

- Il permet de gérer un contenu qui dépasse

Inconvénients :

- Il s'utilise rarement seul
- Il ne faut jamais avoir plusieurs ScrollView imbriqués, ou avoir un élément qui scroll dans le même sens que la scrollView



Les Layouts (attributs)

- Tous ces layouts possèdent des attributs utiles à leurs placement



Les Layouts (attributs)

- Tous ces layouts possèdent des attributs utiles à leurs placement
 - L'attribut : `android:layout_width` et l'attribut `android:layout_height`



Les Layouts (attributs)

- Tous ces layouts possèdent des attributs utiles à leurs placement
 - L'attribut : `android:layout_width` et l'attribut `android:layout_height`
 - valeurs possible : `match_parent` ou `wrap_content`
 - Ou encore `Odp`



Les Layouts (attributs)

- Tous ces layouts possèdent des attributs utiles à leurs placement
 - L'attribut : `android:id`



Les Layouts (attributs)

- Tous ces layouts possèdent des attributs utiles à leurs placement
 - L'attribut : `android:id`
 - La valeur de cet attribut est un string commençant par : `@+id/`



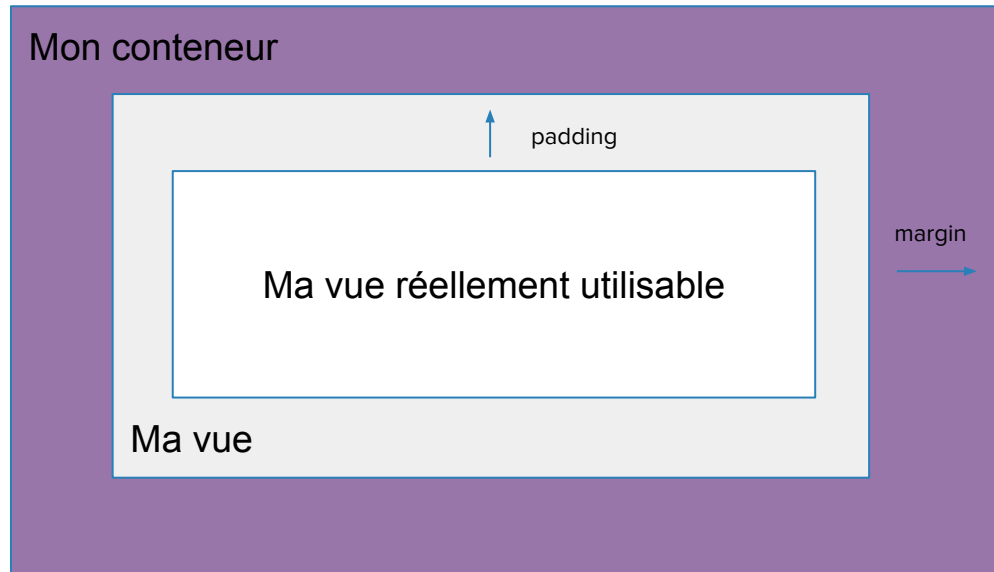
Les Layouts (attributs)

- Tous ces layouts possèdent des attributs utiles à leurs placement
 - L'attribut : `android:paddingXXX` ou `android:marginXXX`



Les Layouts (attributs)

- Tous ces layouts possèdent des attributs utiles à leurs placement
 - L'attribut : `android:paddingXXX` ou `android:marginXXX`
 - Permet de placer des marge externe ou interne autour ou dans sa vue





Les Layouts (attributs)

- Tous ces layouts possèdent des attributs utiles à leurs placement
 - Il y'a plein d'attributs que je vous laisse le soin de découvrir par vous même
 - android:orientation
 - android:elevation
 - android:color
 - android:drawable
 - android:radius
 - android:gravity
 - android:weight
 - etc ...


Aussi beaucoup de ces attributs peuvent être utiles sur certains widgets spécifique et certains sont “globaux”



Les Widgets (Basique)

TextView

Ce widget permet d'afficher une chaîne de caractères que l'utilisateur ne peut modifier. On peut aussi y insérer des chaînes de caractères formatées.

A screenshot of a TextView widget. It is a light gray rectangular box with a thin gray border. Inside the box, the text "Hello World!" is displayed in a black, sans-serif font. The text is centered horizontally and vertically within the box.

Hello World!



Les Widgets (Basique)

TextView

Spécificité XML :

android:text : contient la chaîne de caractère que la TextView va afficher

android:textColor : contient une couleur pour le texte

android:TextSize : contient la taille du texte

android:textStyle : contient un style pour le texte par exemple : gras (bold)

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:textColor="@color/colorPrimary"
    android:textSize="16sp"
    android:textStyle="bold"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```



Les Widgets (Basique)

EditText

Ce widget permet à l'utilisateur d'écrire du texte, Il s'agit en fait d'un TextView mais éditable

Entre ton texte ici



Les Widgets (Basique)

EditText

Spécificité XML :

android:hint : contient une chaîne de caractère à but d'information

android:inputType: contient un type de texte

android:lines : contient le nombre de ligne qui sera occupé par notre EditText

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Entre ton texte ici"
    android:inputType="textMultiLine"
    android:lines="5"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
```



Les Widgets (Basique)

Button

Ce widget est comme son nom l'indique un bouton, il permet de réceptionner un clic utilisateur

A screenshot of an Android button widget. It is a light gray rectangular button with rounded corners and a subtle drop shadow, centered on a white background. The text "MON BOUTON" is written in bold, black, uppercase letters in the center of the button.

MON BOUTON



Les Widgets (Basique)

Button

Spécificité XML :

vue qu'il s'agit grossièrement d'un TextView cliquable, le bouton n'a pas de spécificité différente.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Mon bouton"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
```




Les Widgets (Basique)

CheckBox

Ce widget permet de proposer des choix multiples à l'utilisateur.



Checkbox 1



Checkbox 2



Les Widgets (Basique)

CheckBox

Spécificité XML :

android:checked : contient un boolean (vrai ou faux) et permet de déterminer si par défaut la checkbox doit être cochée ou non

```
<CheckBox
    android:id="@+id/checkbox_one"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Checkbox 1"
    android:checked="true"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
```



Les Widgets (Basique)

RadioButton

Ce widget est le même que CheckBox sauf que lui ne permet à l'utilisateur de n'effectuer qu'un seul choix sur une variété de possibilité



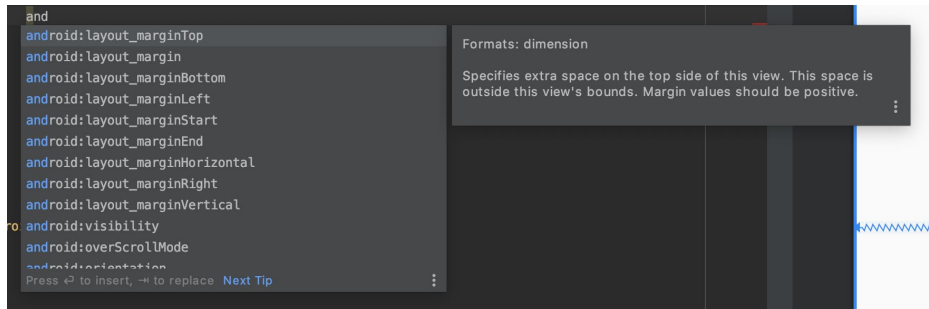
RadioButton 1



Les Widgets (Basique)

Aide sur les attributs

N'hésitez pas si vous cherchez un attribut ou la liste des attributs possible de taper sur votre widget ou layout : **android:** ou **app:** et la documentation du SDK apparaîtra pour vous montrer les possibilités





Interactions sur nos widgets

C'est bien gentil mais j'aimerais par exemple que lorsque l'utilisateur clique sur un bouton des choses ce passe !



Interactions sur nos widgets

Rien de plus simple ! voyons un exemple ensemble !



Interactions sur nos widgets

Tout d'abord dans notre layout (ici activity_main.xml) on va créer ces deux widgets

```
<Button
    android:id="@+id/btn_make_happy"
    android:text="Rendre heureux"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="200dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintBottom_toBottomOf="parent" />

<TextView
    android:id="@+id/tv_mood"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Je ne suis pas heureux :("
    android:layout_marginTop="200dp"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent" />
```



Interactions sur nos widgets

Tout d'abord dans notre layout (ici activity_main.xml) on va créer ces deux widgets

```
<Button
    android:id="@+id/btn_make_happy"
    android:text="Rendre heureux"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="200dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintBottom_toBottomOf="parent" />

<TextView
    android:id="@+id/tv_mood"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Je ne suis pas heureux :("
    android:layout_marginTop="200dp"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent" />
```




Interactions sur nos widgets

Ensuite dans l'activity qui est rattaché à ce layout (ici MainActivity) nous allons utiliser le bouton par son id (ici `btn_make_happy`) et lui ajouter ce qu'on appelle un Listener

```
class MainActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        btn_make_happy.setOnClickListener { it: View!  
            tv_mood.text = "Je suis heureux :)"  
        }  
    }  
}
```



Interactions sur nos widgets

Ensuite dans l'activity qui est rattaché à ce layout (ici MainActivity) nous allons utiliser le bouton par son id (ici `btn_make_happy`) et lui ajouter ce qu'on appelle un Listener

les listeners sont des actions déclenché par des évènement (ici `onClickListener` est déclenché par le clic du bouton)

```
class MainActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        btn_make_happy.setOnClickListener { it: View!  
            tv_mood.text = "Je suis heureux :)"  
        }  
    }  
}
```



Interactions sur nos widgets

Ensuite dans l'activity qui est rattaché à ce layout (ici MainActivity) nous allons utiliser le bouton par son id (ici `btn_make_happy`) et lui ajouter ce qu'on appelle un Listener

les listeners sont des actions déclenché par des évènement (ici `onClickListener` est déclenché par le clic du bouton)

On va donc dire : dès que `btn_make_happy` est cliqué je veut que le `text` de `tv_mood` change par un autre

```
class MainActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        btn_make_happy.setOnClickListener { it: View!  
            tv_mood.text = "Je suis heureux :)"  
        }  
    }  
}
```



Interactions sur nos widgets

Cela marche mais ce n'est pas propre !

```
class MainActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        btn_make_happy.setOnClickListener { it: View!  
            tv_mood.text = "Je suis heureux :)"  
        }  
    }  
}
```



Interactions sur nos widgets

Pour rendre cela propre il faut bien séparer l'initialisation de la gestion des évènements !

car lorsque l'on a plusieurs widget onCreate devient vite un gros pavé de gestion

```
class MainActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        btn_make_happy.setOnClickListener { it: View!  
            tv_mood.text = "Je suis heureux :)"  
        }  
    }  
}
```



Interactions sur nos widgets

Voici le même code mais bien plus propre ! (et encore !)

```
class MainActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        initView()  
    }  
  
    private fun initView() {  
        initMoodTextView()  
        initMakeHappyButton()  
    }  
  
    private fun initMoodTextView() {  
        tv_mood.text = "Je ne suis pas heureux :("   
    }  
  
    private fun initMakeHappyButton() {  
        btn_make_happy.text = "Rendre heureux"  
        manageMakeHappyButton()  
    }  
  
    private fun manageMakeHappyButton() {  
        btn_make_happy.setOnClickListener { it: View!   
            tv_mood.text = "Je suis heureux :)"  
        }  
    }  
}
```

Un peu d'exercice



On a assez d'information pour un petit exercice :) ! Devinez quoi ? :p

Un peu d'exercice



Faire un écran sur une seule Activity qui simule un portique de fac :) !



Un peu d'exercice

En reprenant l'exercice réalisé sur Kotlin développer une interface graphique qui permet de gérer si une personne que vous allez renseigné est autorisé ou non à entrer dans la fac.

AndroidLearning

Prénom

Nom de famille

Age

STUDENT

LANCER LA VÉRIFICATION

Ici s'affichera si la personne peux rentrer ou non



Un peu d'exercice

L'application doit contenir les mêmes choses que dans l'exercice précédent à quelque changement près :

- Les attributs de la classe Person sont désormais tous mutable (de type var) et son initialisé avec des valeurs par défaut
- La classe Person et l'enum STATUS doivent être dans un “package” séparé de la MainActivity
- L'extension Int.isMajot() doit être dans un “package” séparé de la MainActivity
- La gestion du status est optionnel (dans un premier temps)
- L'interface graphique est libre (vous pouvez vous amusez à créer votre propre interface de gestion de cet exercice ou vous appuyé sur la mienne dans celle à gauche dans un premier temps)

Le seul objectif de cet exercice : Lorsque je clique sur “Lancer la vérification” quelque chose doit apparaître/changer pour me dire que l'utilisateur est autorisé ou non à entrer (dans un premier temps modifier le TextView du milieu)

AndroidLearning

Prénom

Nom de famille

Age

STUDENT

LANCER LA VÉRIFICATION

Ici s'affichera si la personne peut rentrer ou non