

IUT de Montpellier- 2022 / 2023



Développement d'applications mobiles

Android

Enseignant :
Malo Gasquet

Déroulement du module

- Séance 1 – **Introduction à Android**
- Séance 2-3 - Développement et utilisation d'une **API REST** (The Feed, l'application)
- Si le temps le permet : utilisation d'une **base de données en temps réel** (firebase)
- Séance 4 : Début du **projet**

Evaluation

- **Note de participation** : remise de l'état d'avancement du TP en cours à la fin de chaque séance.
- **Note de projet** : projet en **binôme**, 4h de cours prévues pour le lancement.
- **A priori, pas d'examen**

Introduction

- Jusqu'en **2007**, les services proposés par les téléphones étaient très limités
- Il n'y avait **pas de tactile**, ou alors avec un stylet. La plupart des téléphones étaient à touches
- On ne parlait pas vraiment d'**applications mobiles**. L'utilisation d'internet sur les téléphones était très limitée et utilisait principalement les données mobiles (qui étaient très limitées et coûteuses à l'époque)
- La **puissance** des téléphones était aussi **limitée** (pas besoin de beaucoup de puissance pour envoyer des message ou passer des appels...)
- Les **téléphones** étaient donc principalement utilisés comme **moyen de communication basique** (appels / sms / mms, éventuellement quelques petits jeux simples), et étaient loin de l'outil multi-fonction que nous connaissons aujourd'hui
- Cependant, en **2007**, **Apple** provoque une **révolution** qui changea à jamais notre mode d'utilisation de cet outil en présentant l'**iphone**, lors d'une [conférence](#). Toutes les bases du monde du développement mobile ont été posées à ce moment-là. L'idée est simple : proposer un appareil permettant à la fois de communiquer, de se divertir et d'exploiter internet, tout en proposant une utilisation facilitée en utilisant ses doigts.

Introduction

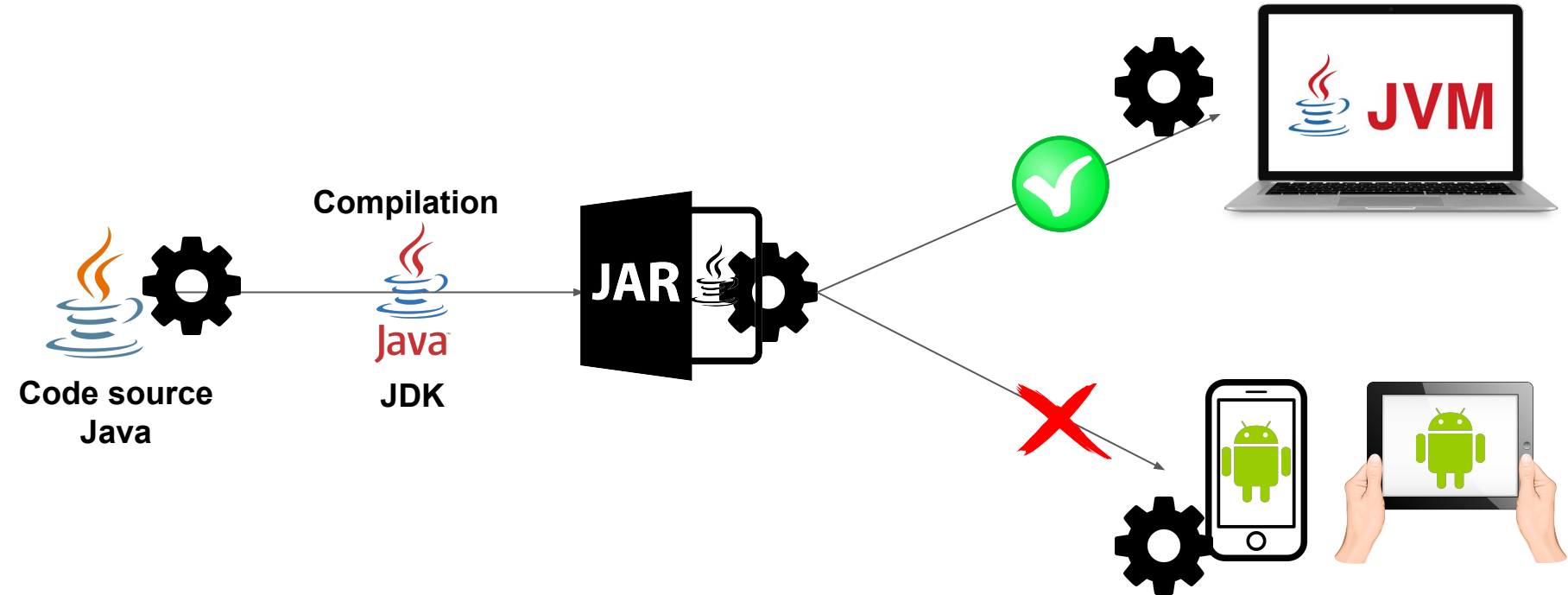
- **Android** est un **système d'exploitation mobile**. La première version date de 2008
- A la base utilisé principalement pour les **smartphones** et les **tablettes**, son utilisation s'est étendue aux **télévisions**, aux **montres connectées**, et même aux **voitures**
- Les applications Android sont principalement développées avec **Java**
- Android "remplace" les bibliothèques graphiques de Java par les siennes
- Android est *plus ou moins libre* et **open-source**
- Android est développé par **Google**. C'est un des principaux concurrents à **iOS** d'**Apple**. **Android** possède presque **80%** des parts de marché mondial dans ce secteur



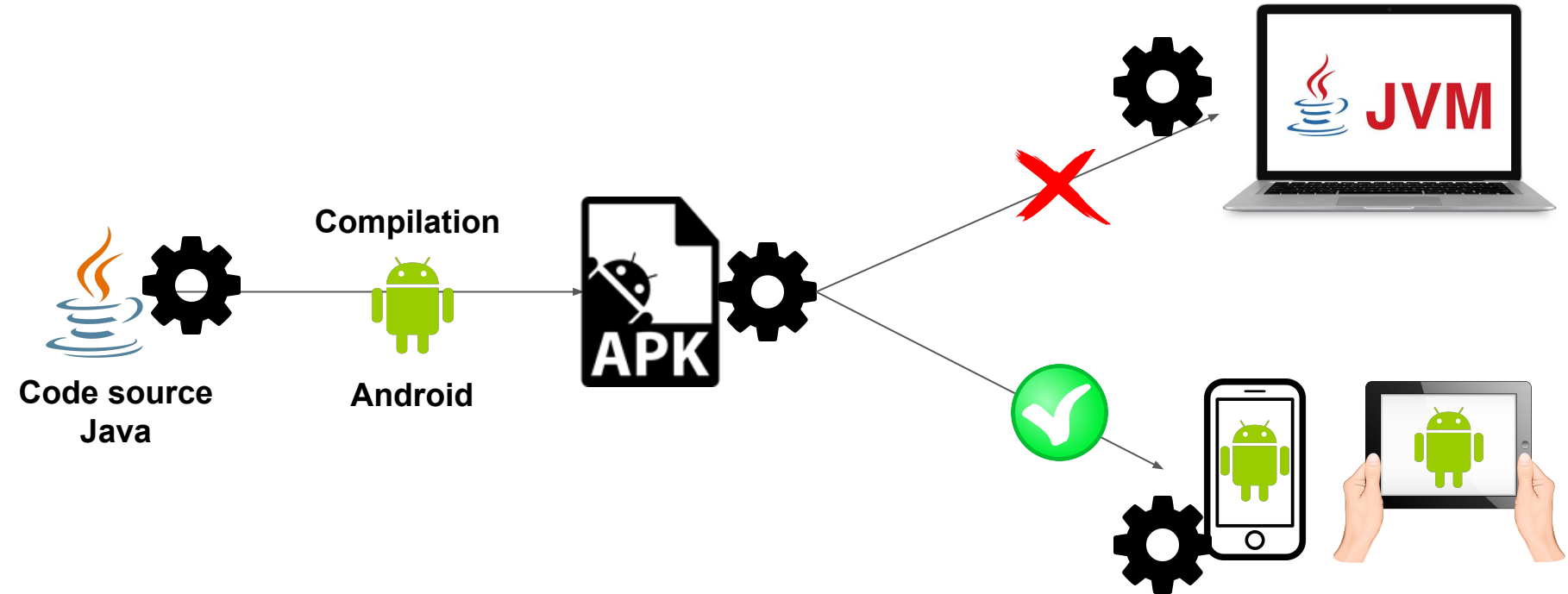
La DVM et l'ART VS La JVM

- Le code **Java** utilisé pour développer les applications **Android** n'est pas exécuté par une **JVM classique**. De même, la **compilation** ne se fait pas de la même manière.
- En effet, l'environnement et l'architecture des appareils mobiles n'est pas la même que pour les ordinateurs classiques. **Android doit donc fournir sa propre machine virtuelle**
- Au départ, il s'agissait d'une machine virtuelle nommée **DVM** (Dalvik Virtual Machine) qui était adaptée à la faible puissance des appareils mobiles de l'époque. Avec l'avancée technologique, à partir de 2014, cette machine virtuelle a été remplacée par **l'ART** (Android Runtime), offrant des performances plus avantageuses
- Ainsi, même si la syntaxe du code **JAVA** utilisée est exactement la même, le **bytecode** généré n'est pas le même que celui utilisable par la **JVM**. Ainsi, une application mobile ne peut pas tourner sur un ordinateur (excepté via un émulateur, bien entendu) et un programme compilé pour la **JVM** sur un ordinateur ne peut pas être exécuté par votre téléphone

La DVM et l'ART VS La JVM



La DVM et l'ART VS La JVM



Kotlin

- **Kotlin** est un **langage de programmation** orienté objet (et fonctionnel) qui permet aussi de développer des applications **Android**.
- Il provient principalement de développeurs issus de **JetBrains** (encore!)
- Il s'inspire de la syntaxe et des concepts de différents (JAVA, C# par exemple)
- Son utilisation permet de faire des applications **multi-plateformes** (Android + iOS) plus facilement.
- Aujourd'hui, Google recommande d'utiliser **Kotlin** plutôt que **JAVA** pour développer des applications Android.
- Dans les TDs, pour ne pas vous surcharger avec un nouveau langage, **nous allons rester sur du JAVA**, mais plus tard, vous pouvez jeter un oeil dans ce langage et essayer de re-coder les applications que nous voyons en TD : <https://kotlinlang.org/>

Choix de la version d'Android

- Quand on commence à développer une **application Android**, on doit choisir quelle **version d'Android** utiliser
- Les nouvelles versions offrent de nouvelles fonctionnalités pour le développement des applications
- Une application développée pour une certaine version sera **compatible** avec un système qui possède **cette version** ou une **ultérieure**
- Une application développée pour une certaine version ne sera **pas compatible** avec un système qui possède une version **antérieure**
- Par exemple, une application développée pour Android 7.0 est compatible avec un système tournant sur Android 7.0, 7.1, 8.0, etc...et toutes les prochaines versions. Elle ne sera pas compatible avec un système tournant sur Android 6.0, 5.1, etc...
- Le choix de la **version d'Android** à utiliser est donc primordial. Il est possible de **mesurer le niveau de compatibilité** de votre application grâce aux statistiques d'utilisations des différentes versions d'Android

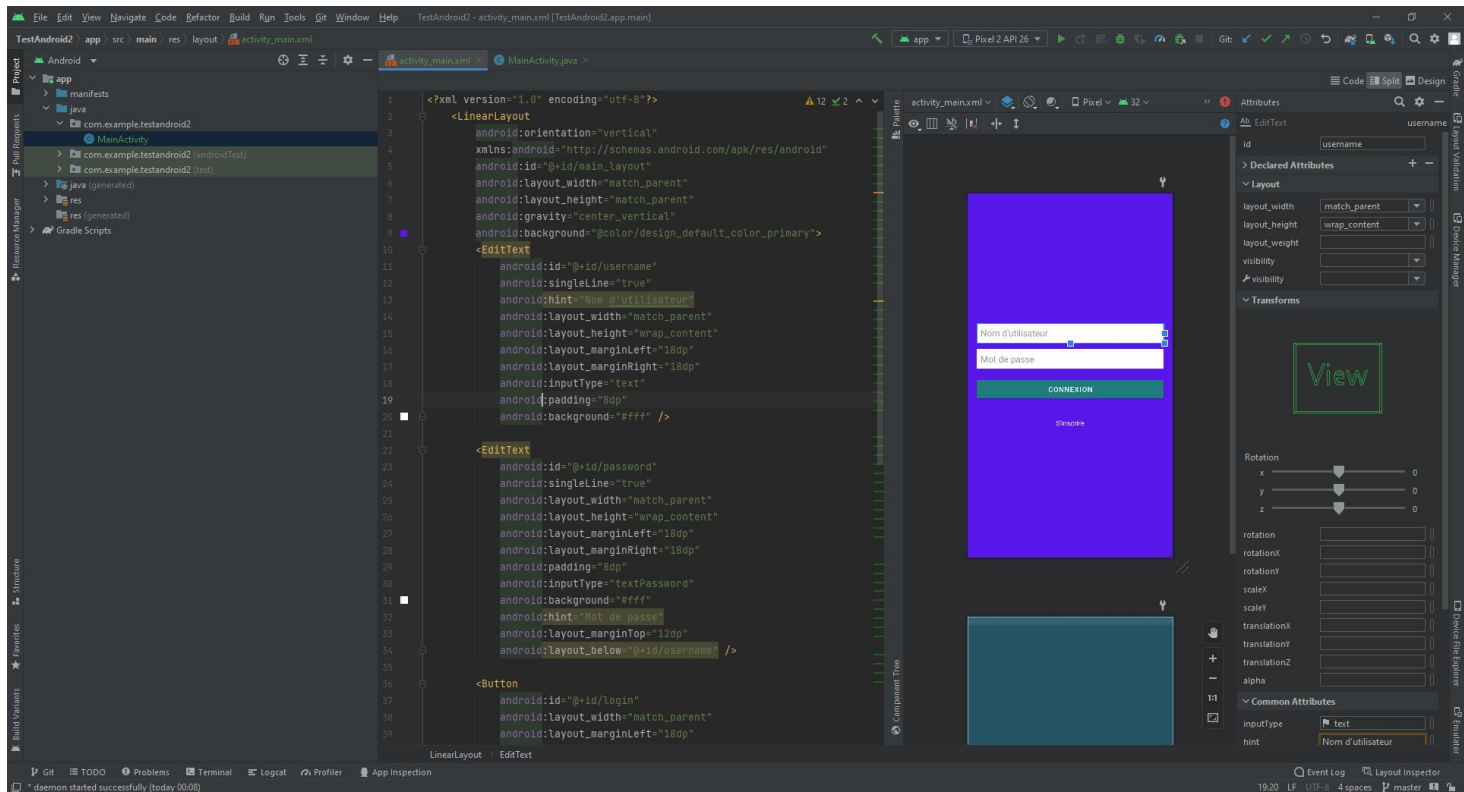
ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.1 Jelly Bean	16	
4.2 Jelly Bean	17	99,9%
4.3 Jelly Bean	18	99,7%
4.4 KitKat	19	99,6%
5.0 Lollipop	21	98,6%
5.1 Lollipop	22	98,1%
6.0 Marshmallow	23	95,6%
7.0 Nougat	24	91,7%
7.1 Nougat	25	89,1%
8.0 Oreo	26	86,7%
8.1 Oreo	27	83,5%
9.0 Pie	28	75,1%
10. Q	29	58,9%
11. R	30	35,0%

Android Studio



- **Android Studio** est un **IDE** conçu pour le développement d'applications **Android**
- Il est développé par **Google** et est un dérivé de **IntelliJ** (IDE Java, par **JetBrains**, comme pour PhpStorm)
- Il fournit différents outils très utiles lors de la conception d'une application :
 - Un “**Scene Builder**” pour construire et visualiser le rendu un écran d'une application à travers une interface, en plus du code **XML** de l'interface
 - Un gestionnaire d'**émulateurs** pour pouvoir tester les applications sur différents types d'appareils (différentes marques, différentes versions d'android, différents périphériques...)
 - Il y a également la possibilité de tester son application directement sur son appareil en le branchant à l'ordinateur

Android Studio



Activity - Les écrans de l'application

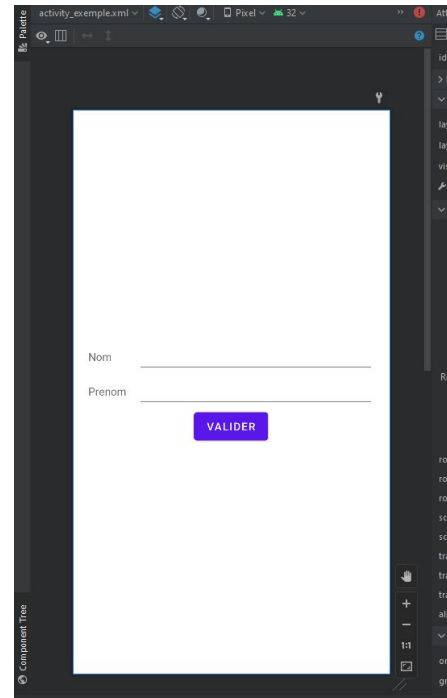
- En **Android**, les différents **écrans** d'une application sont appelés **Activity** (une activité)
- Une application est (généralement) composée de **plusieurs activités**
- Une **Activity** se décompose en deux parties :
 - **L'interface**, représentée par un **document XML** qui définit et organise les différents **composants graphiques**. Par ailleurs, une même interface XML peut être utilisée par des activités différentes
 - Une **classe Java** (héritant de **Activity**) qui s'associe à une interface définie par un document **XML** et définit le **comportement** lié aux **événements qui surviennent sur l'interface (programmation événementielle)**. A noter que la classe Java peut se passer de l'interface définie par le fichier XML et générer / ajouter des composants sur l'écran via du code (comme pour du JS ou comme dans JavaFX, par exemple)

Activity - Le document XML définissant l'interface

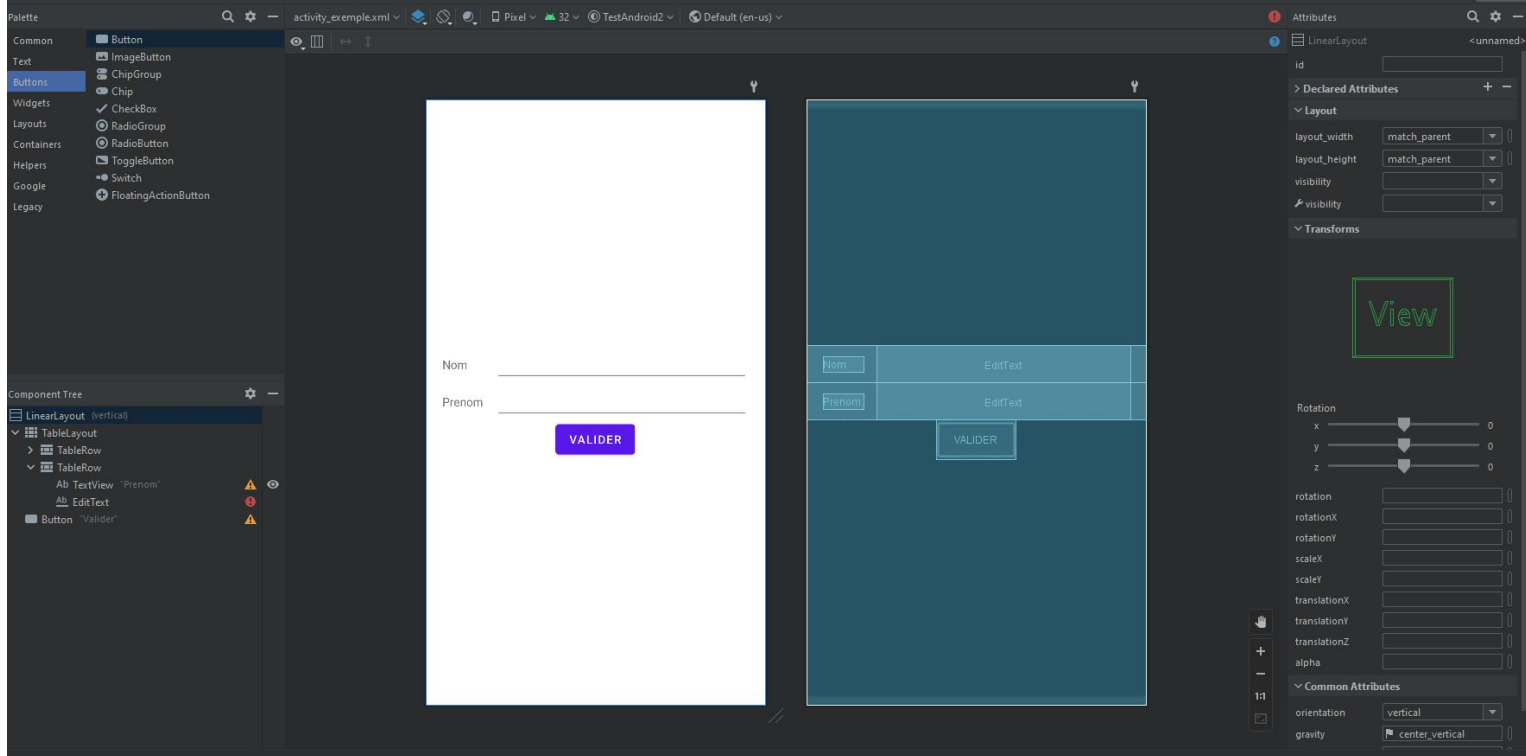
- Le **XML** est un langage utilisant des **balises** pour organiser de l'information
- Ici, il est utilisé pour construire l'**interface** d'une **Activity**
- Dans l'idée, cela est semblable au **HTML** (ou au **FXML!**). **ML** signifie "**Markup Language**" (langage à balises)
- Android propose différents éléments **conteneurs** appelés **ViewGroup** qui possèdent un **layout** spécifique
- Ensuite, il y a bien sûr différents **composants graphiques** appelés **View** (boutons, champs de saisies, menu, etc...) qui peuvent être ajoutés dans des **ViewGroup**
- Comme pour le **HTML**, la **hiérarchie** des éléments se compose en plaçant les éléments fils dans la zone définie entre la balise ouvrante et la balise fermante de l'élément parent
- Les **ViewGroup** peuvent eux-mêmes contenir d'autres **ViewGroup**, ce qui rend possible la création d'interfaces avancées
- Tous ces éléments possèdent différents **attributs** qu'ils est possible d'éditer au niveau de la déclaration de la balise afin de changer le style de rendu

Activity - Le document XML - Exemple

```
<?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"
    android:gravity="center_vertical">
    <TableLayout android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical">
        <TableRow android:paddingLeft="20dp" android:paddingRight="20dp">
            <TextView android:text="Nom" android:visibility="visible"/>
            <EditText android:layout_marginLeft="15dp" android:layout_weight="1"/>
        </TableRow>
        <TableRow android:paddingLeft="20dp" android:paddingRight="20dp">
            <TextView android:text="Prenom" android:visibility="visible" />
            <EditText android:layout_marginLeft="15dp" android:layout_weight="1"/>
        </TableRow>
    </TableLayout>
    <Button android:layout_height="wrap_content" android:layout_width="wrap_content"
        android:layout_gravity="center_horizontal" android:text="Valider"/>
</LinearLayout>
```



Activity - L'interface de "design"

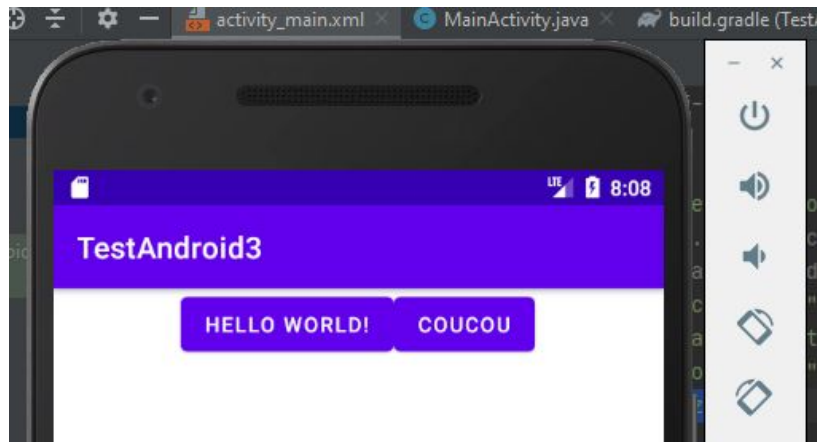


Activity - Les layouts (ViewGroup) - Linear Layout

- **LinearLayout** est un **ViewGroup** qui permet d'organiser les éléments **verticalement** ou **horizontalement** (similaires à une VBox et une HBox en **JavaFX**)
- On règle la disposition du **layout** avec l'attribut **orientation** (horizontal / vertical)

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal"
    android:orientation="horizontal">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Coucou"/>
</LinearLayout>
```

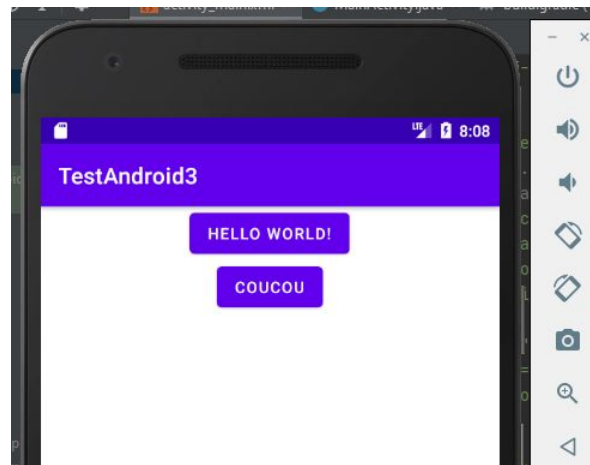


Activity - Les layouts (ViewGroup) - Linear Layout

- **LinearLayout** est un **ViewGroup** qui permet d'organiser les éléments **verticalement** ou **horizontalement** (similaires à une **VBox** et une **HBox** en **JavaFX**)
- On règle la disposition du **layout** avec l'attribut **orientation** (horizontal / vertical)

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal"
    android:orientation="vertical">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Coucou"/>
</LinearLayout>
```



Activity - Les layouts (ViewGroup) - Table Layout

- **TableLayout** est un **ViewGroup** qui permet d'organiser les éléments en grille
- Chaque ligne est représentée par un autre **ViewGroup** appelé **TableRow** contenant les composants graphiques d'une ligne
- On peut préciser le **numéro de colonne** d'un composant avec l'attribut **layout_column**

```
<TableLayout
    android:layout width="match parent"
    android:layout height="wrap content">
    <TableRow android:paddingLeft="20dp" android:paddingRight="20dp">
        <TextView
            android:text="Nom"
            android:visibility="visible" />
        <EditText android:layout_marginLeft="15dp" android:layout_weight="1"/>
    </TableRow>
    <TableRow android:paddingLeft="20dp" android:paddingRight="20dp">
        <TextView
            android:text="Prenom"
            android:visibility="visible" />
        <EditText android:layout_marginLeft="15dp" android:layout_weight="1"/>
    </TableRow>
</TableLayout>
```



Vue sur l'éditeur, les bordures n'apparaissent pas dans l'application

Activity - Les layouts (ViewGroup) - Relative Layout

- **RelativeLayout** permet de disposer les éléments de l'interface par rapport à d'autres éléments
- On indique à l'aide de **layout_below**, **layout_left**, **layout_right**, **layout_top**... par rapport à qui un élément se place (au dessus de..., en dessous de...)
- Pour référencer un élément, on utilise un **identifiant**
- Les attributs **layout_align...** permettent **d'aligner** l'élément par rapport à son parent où par rapport à un autre élément

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:gravity="center_vertical"
    android:layout_height="match_parent" >
    <EditText
        android:id="@+id/champ"
        android:layout_alignParentLeft="true"
        android:layout_width="match_parent"
        android:layout_below="@+id/coucou"
        android:layout_height="wrap_content" >
    </EditText>
    <Button
        android:id="@+id/btn1"
        android:layout_width="wrap_content"
        android:layout_below="@+id/champ"
        android:layout_alignParentLeft="true"
        android:text="Test"
        android:layout_height="wrap_content" >
    </Button>
    <TextView
        android:id="@+id/coucou"
        android:layout_width="match_parent"
        android:layout_alignParentLeft="true"
        android:layout_height="wrap_content" android:text="Demo" />
</RelativeLayout>
```

Activity - Les layouts (ViewGroup) - Relative Layout

- **RelativeLayout** permet de disposer les éléments de l'interface par rapport à d'autres éléments
- On indique à l'aide de **layout_below**, **layout_left**, **layout_right**, **layout_top**... par rapport à qui un élément se place (au dessus de..., en dessous de...)
- Pour référencer un élément, on utilise un **identifiant**
- Les attributs **layout_align...** permettent **d'aligner** l'élément par rapport à son parent où par rapport à un autre élément



Activity - Quelques composants (View) - Button

- Permet de gérer un **bouton**

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Coucou" />
```



Activity - Quelques composants (View) - TextView

- Permet de gérer un **élément textuel** simple
- On peut régler la taille avec **textSize**

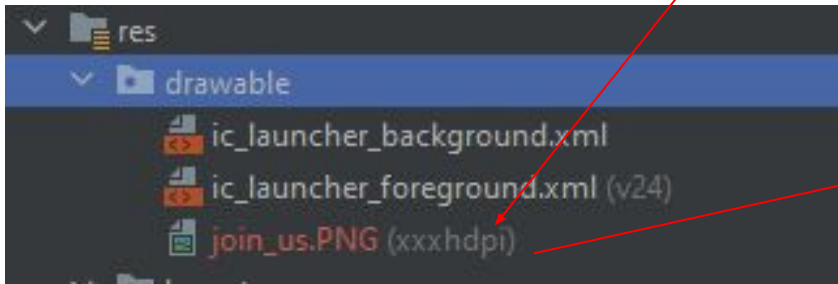
```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textSize="30dp"  
    android:text="Coucou" />
```

Coucou

Activity - Quelques composants (View) - ImageView

- Permet d'afficher une **image**

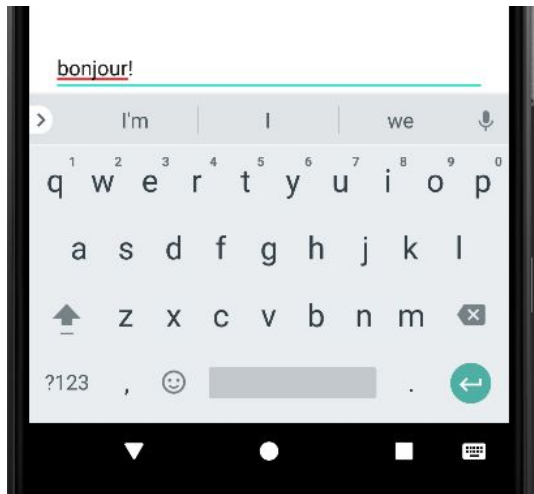
```
<ImageView  
    android:layout_height="match_parent"  
    android:layout_width="match_parent"  
    android:src="@drawable/join_us" />
```



Activity - Quelques composants (View) - EditText

- **EditText** permet de gérer un champ textuel
- Ce composant est **hautement configurable** pour obtenir différents types de champs : mot de passe, numéro, nombres, etc...On fait cela au travers de l'attribut **inputType**. Selon l'input choisis, le clavier de saisie change

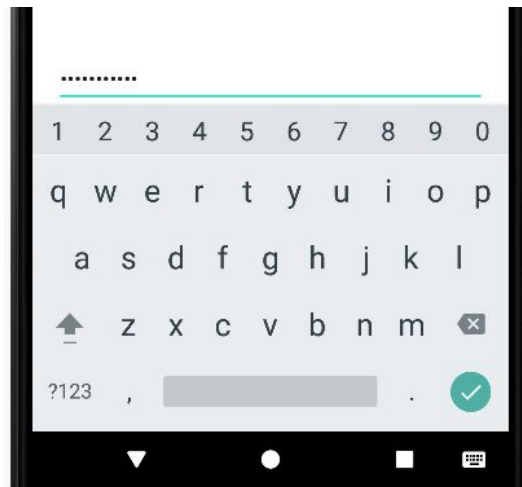
```
<EditText  
    android:layout_height="wrap_content"  
    android:layout_width="match_parent"  
    android:layout_marginHorizontal="20dp"/>
```



Activity - Quelques composants (View) - EditText

- **EditText** permet de gérer un champ textuel
- Ce composant est **hautement configurable** pour obtenir différents types de champs : mot de passe, numéro, nombres, etc...On fait cela au travers de l'attribut **inputType**. Selon l'input choisis, le clavier de saisie change

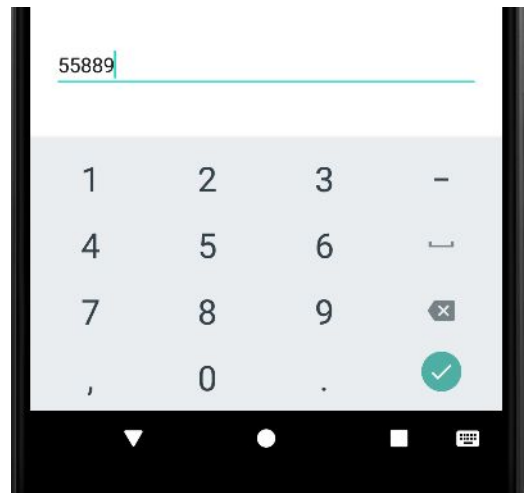
```
<EditText
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:layout_marginHorizontal="20dp"
    android:inputType="textPassword" />
```



Activity - Quelques composants (View) - EditText

- **EditText** permet de gérer un champ textuel
- Ce composant est **hautement configurable** pour obtenir différents types de champs : mot de passe, numéro, nombres, etc...On fait cela au travers de l'attribut **inputType**. Selon l'input choisis, le clavier de saisie change

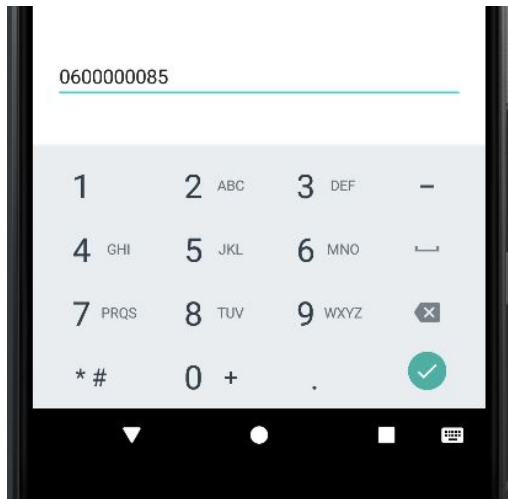
```
<EditText  
    android:layout_height="wrap_content"  
    android:layout_width="match_parent"  
    android:layout_marginHorizontal="20dp"  
    android:inputType="number" />
```



Activity - Quelques composants (View) - EditText

- **EditText** permet de gérer un champ textuel
- Ce composant est **hautement configurable** pour obtenir différents types de champs : mot de passe, numéro, nombres, etc...On fait cela au travers de l'attribut **inputType**. Selon l'input choisis, le clavier de saisie change

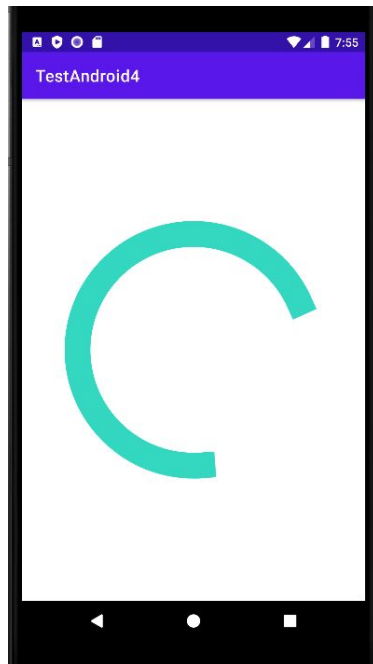
```
<EditText  
    android:layout_height="wrap_content"  
    android:layout_width="match_parent"  
    android:layout_marginHorizontal="20dp"  
    android:inputType="phone" />
```



Activity - Quelques composants (View) - ProgressBar


- Animation de **chargement**
- Il est possible de changer le style de l'animation avec l'attribut **style**

```
<ProgressBar  
    style="@style/Widget.AppCompat.ProgressBar"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" />
```



Activity - Quelques composants (View) - Spinner

- Permet de gérer une **liste de choix**
- On définit les choix pré-existants dans un fichier **"string.xml"** dans **"res/values"** (on peut bien sûr en ajouter avec du code)
- On peut choisir deux modes : **"dropdown"** : liste déroulante, **"dialog"** : ouvre une boîte de dialogue pour faire le choix



```
<resources>
  <string name="app_name">TestAndroid4</string>

  <string-array name="choix">
    <item>Bonjour</item>
    <item>Hello</item>
    <item>There</item>
  </string-array>

</resources>
```

Activity - Quelques composants (View) - Spinner

- Permet de gérer une **liste de choix**
- On définit les choix pré-existants dans un fichier **"string.xml"** dans **"res/values"** (on peut bien sûr en ajouter avec du code)
- On peut choisir deux modes : **"dropdown"** : liste déroulante, **"dialog"** : ouvre une boîte de dialogue pour faire le choix

```
<Spinner  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:spinnerMode="dropdown"  
    android:entries="@array/choix" >
```



Activity - Quelques composants (View) - Spinner

- Permet de gérer une **liste de choix**
- On définit les choix pré-existants dans un fichier **"string.xml"** dans **"res/values"** (on peut bien sûr en ajouter avec du code)
- On peut choisir deux modes : **"dropDown"** : liste déroulante, **"dialog"** : ouvre une boîte de dialogue pour faire le choix

```
<Spinner  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:spinnerMode="dialog"  
    android:entries="@array/choix" >
```



Activity - Tailles des composants

- Il est possible (et souvent, obligatoire) de fixer la **hauteur** et la **largeur** des composants via les **attributs `layout_height` et `layout_width`**
- On peut utiliser une valeur numérique en **dp**, mais celle-ci est fixe et ne s'adapte pas aux différentes tailles d'écrans (**non-responsive**)
- On utilise plus généralement les deux valeurs suivantes, plus **responsives** :
 - **`match_parent`** : même taille que le conteneur parent (pour le conteneur principal, cela prend donc toute la hauteur et / ou la largeur de l'écran)
 - **`wrap_content`** : la taille s'adapte au contenu du composant

Activity - Alignement des composants

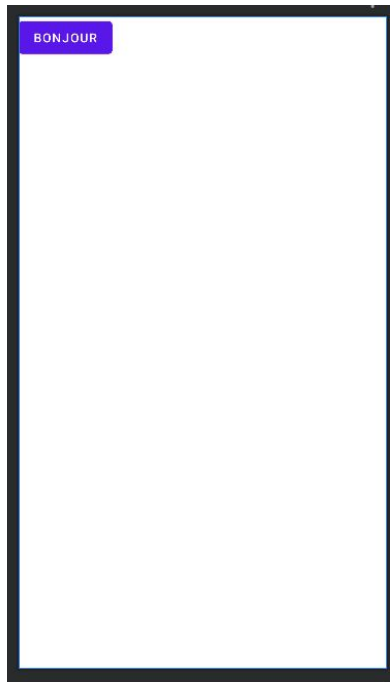
- Il existe deux **attributs** qui contrôle l'**alignement** des composants :
 - **gravity** : contrôle l'**alignement des éléments contenus** (des **enfants**). On l'utilise donc généralement sur un **ViewGroup**. On peut néanmoins également l'utiliser sur d'autres types de composants (changer l'alignement du texte dans un bouton, par exemple...)
 - **layout_gravity** : contrôle l'**alignement d'un élément à l'intérieur de son conteneur** (par rapport à l'alignement donné par son **parent**). Par exemple, si on a un **LinearLayout** avec un **gravity** centrant verticalement les éléments, **layout_gravity** permettra à un élément contenu dans le **LinearLayout** de s'aligner à gauche, à droite, etc...dans la zone du centre. Comme un **ViewGroup** peut contenir un autre **ViewGroup**, il est tout à fait possible d'appliquer gravity sur un **ViewGroup** (conteneur)
- On peut attribuer différentes **valeurs** à ceux des **attributs** : **center**, **center_vertical**, **center_horizontal**, **left**, **right**, **top**, **bottom**, etc....

Activity - Alignement des composants

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

    <Button android:layout width="wrap_content"
    android:layout height="wrap_content"
        android:text="Bonjour" />

</LinearLayout>
```

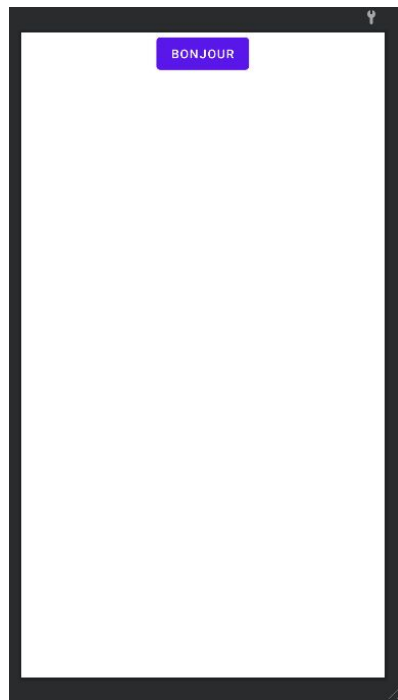


Activity - Alignement des composants

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

    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Bonjour" />

</LinearLayout>
```

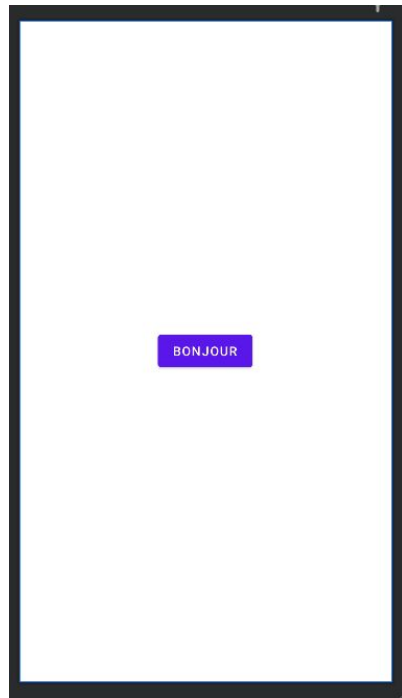


Activity - Alignement des composants

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

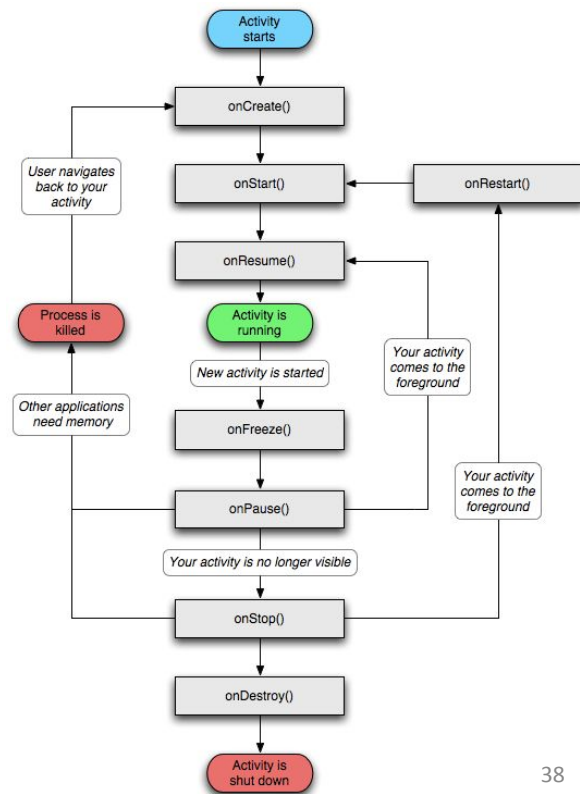
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Bonjour" />

</LinearLayout>
```



Activity - La partie Java

- Côté **Java**, il faut créer une **classe** héritant de **Activity** ou, dans les version plus récentes d'Android, de **AppCompatActivity**
- Il est possible de **surcharger** différentes méthodes qui gèrent le cycle de vie de l'**Activity** et qui sont déclenchées selon certains événements : **onCreate** (initialisation de l'écran), **onStop**, **onResume**...
- Pour relier l'**Activity** à son interface définie par le **document XML**, cela se fait dans la méthode **onCreate**, en appelant la méthode **setContentView**
- Chaque **document XML** définissant une **interface** est rangé dans un dossier "**layout**" dans le dossier **res** (ressources). On peut alors facilement y faire référence ainsi : **R.layout.nom_fichier**



Activity - La partie Java

- Ici, on charge l'interface définie par un fichier **XML** nommé “**activity_main.xml**” dans **layout**
- Dans toute méthode **surchargée** liée au **cycle de vie** de l'**Activity**, il faut impérativement débiter par un appel de la méthode du parent (avec **super**)

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

Activity - La partie Java - Récupérer les composants

- Dans le fichier **XML**, on peut attribuer un **identifiant** à chaque composant en utilisant l'attribut **id**, avec pour valeur **"@+id/nomIdentifiant"** (en remplaçant **nomIdentifiant**, bien entendu)
- Un peu à la manière de **javascript**, dans le code Java de l'**Activity**, au niveau du **onCreate**, on peut récupérer l'élément (après avoir appelé **setContentView**) en utilisant la méthode **findViewById(R.id.nomIdentifiant)**
- On va généralement définir des **attributs** correspondant à chaque composant dans la classe et les affecter dans le **onCreate** pour pouvoir les réutiliser facilement plus tard

```
<Button android:id="@+id/btn1"
        android:layout width="wrap_content"
        android:layout height="wrap_content"
        android:text="Bonjour" />
```

```
public class MainActivity extends Activity {

    private Button btn1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        this.btn1 = findViewById(R.id.btn1);
    }

}
```


Activity - La partie Java - Programmation événementielle

- La **programmation événementielle** se réalise quasiment de la même manière qu'en **JavaFX**, avec deux possibilités :
 - On utilise directement un **attribut** sur le composant dans le **document XML** pour relier la gestion d'un **événement** avec une **méthode** dans la **classe** de l'**Activity**
 - Ou bien, dans le code **Java** (dans le **onCreate**), on attache la gestion d'un **événement** (avec une méthode spécifique) au **composant** en spécifiant la **méthode** à exécuter. Cette méthode est à **privilégier** par rapport à la méthode dans le **XML** qui est **dépréciée**
- Concernant la **programmation d'applications mobile**, on utilise beaucoup l'événement "**onClick**"
- Généralement, le paramètre de base passé à la méthode est un objet de type **View** (le composant sur lequel l'événement a été déclenché). Nous pouvons nous en servir ou non

Activity - La partie Java - Programmation événementielle

- Dans le code **JAVA**, quand on relie un événement à l'exécution d'une fonction, on utilise un type objet particulier appelé **Consumer**.
- Un objet **Consumer** est une interface qui représente une **fonction** qui peut être passée en paramètre à une autre fonction et appelée.
- Le **Consumer**, via la généricité, doit spécifier le type d'attribut que doit prendre la fonction (par conséquence, la fonction représentée par le **Consumer** ne peut prendre qu'un et un seul attribut). Dans le cadre des événements sur l'interface, cet attribut est donc généralement de type **View**.
- Le **Consumer** ne retourne rien.
- Il existe d'autres types d'objets dans ce style :
 - **Runnable** : fonction sans paramètre qui ne renvoie rien (procédure)
 - **Function** : fonction dont on spécifie le type d'entrée et le type de retour (doit donc retourner une valeur).

Activity - La partie Java - Programmation événementielle

- Lors de l'appel de la fonction qui prend en paramètre le **Consumer**, on peut faire référence à la fonction cible de deux manières :

```
public void exemple() {  
    fonctionAvecCallback(10, 3, (result) -> this.callback(result));  
}  
  
public void fonctionAvecCallback(int a, int b, Consumer<Integer> callback) {  
    callback.accept(a+b) ;  
}  
  
public void callback(Integer nombre) {  
    ...  
}
```

Activity - La partie Java - Programmation événementielle

- Lors de l'appel de la fonction qui prend en paramètre le **Consumer**, on peut faire référence à la fonction cible de deux manières :

```
public void exemple() {  
    fonctionAvecCallback( 10, 3, this::callback) ;  
}  
  
public void fonctionAvecCallback (int a, int b, Consumer<Integer> callback) {  
    callback.accept(a+b) ;  
}  
  
public void callback(Integer nombre) {  
    ...  
}
```

Activity - La partie Java - Programmation événementielle

```
<Button android:id="@+id/btn1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

    android:layout_gravity="center_horizontal"
    android:onClick="turnRed"
    android:text="Bonjour" />
```

```
public class MainActivity extends Activity {

    private Button btn1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        this.btn1 = findViewById(R.id.btn1);
    }

    public void turnRed(View composant) {
        this.btn1.setBackgroundColor(Color.RED);
        this.btn1.setText("Je suis rouge!");
    }

}
```



Activity - La partie Java - Programmation événementielle

```
public class MainActivity extends Activity {  
  
    private Button btn1;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        this.btn1 = findViewById(R.id.btn1);  
        this.btn1.setOnClickListener((view) -> this.turnRed(view));  
    }  
  
    public void turnRed(View composant) {  
        this.btn1.setBackgroundColor(Color.RED);  
        this.btn1.setText("Je suis rouge!");  
    }  
}
```

BONJOUR



JE SUIS ROUGE!

Activity - La partie Java - Programmation événementielle

```
public class MainActivity extends Activity {  
  
    private Button btn1;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        this.btn1 = findViewById(R.id.btn1);  
        this.btn1.setOnClickListener((view) -> this.turnRed());  
    }  
  
    public void turnRed() {  
        this.btn1.setBackgroundColor(Color.RED);  
        this.btn1.setText("Je suis rouge!");  
    }  
}
```

BONJOUR



JE SUIS ROUGE!

Activity - La partie Java - Programmation événementielle

```
public class MainActivity extends Activity {  
  
    private Button btn1;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        this.btn1 = findViewById(R.id.btn1);  
        this.btn1.setOnClickListener(this::turnRed);  
    }  
  
    public void turnRed(View composant) {  
        this.btn1.setBackgroundColor(Color.RED);  
        this.btn1.setText("Je suis rouge!");  
    }  
}
```

BONJOUR

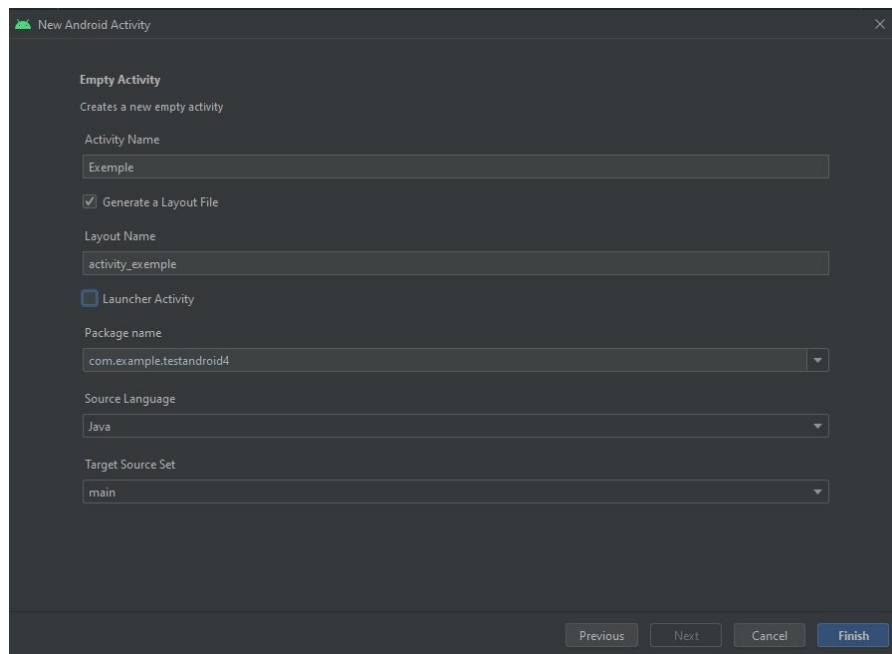
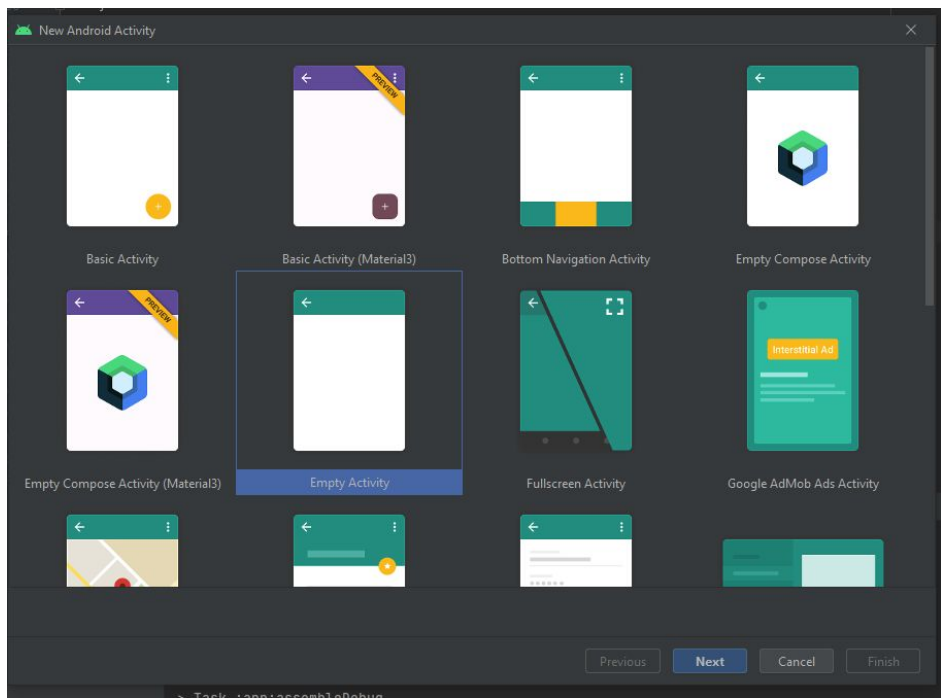


JE SUIS ROUGE!

Activity - Création avec Android Studio

- **Android Studio** permet de simplement créer des activités en générant directement la **classe** et le **document XML** associé (et il place également les fichiers au bon endroit)
- Pour cela, on se rend dans “**File**”, “**New**”, “**Activity**”
- **Android Studio** propose divers modèles d’activités prédéfinies. Si on veut quelque-chose de vierge, il faut sélectionner “**Empty Activity**”
- On précise le **langage de programmation** (Java), le **nom** de l’activité, son **package** de base. Il est possible de cocher “**Launcher Activity**” si on souhaite que cela soit cette activité qui soit lancée au démarrage de l’application

Activity - Création avec Android Studio



Boîtes de dialogues

- Comme nous l'avons vu pour les précédentes technologies permettant de réaliser des IHM, il est possible d'afficher diverses **boîtes de dialogues** :
 - **Alertes** (message, avec confirmation / annulation...)
 - Choix d'une **date**
 - **Message** à taper
 - Il est également possible de définir ses propres boîtes de dialogues
- Les **événements** de validation, d'annulation et les choix effectués peuvent être **captés** et **traités** par l'activité

Boîtes de dialogues - Exemple - Alertes

```
@Override
protected void onCreate(Bundle
savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    this.btn1 = findViewById(R.id.btn1);

    this.btn1.setOnClickListener(this::declencherDi
alog);
}
```

- Il est possible de définir un **titre**, un **message**, si on **peut fermer l'alerte ou non** (en appuyant autre part) et la méthode à exécuter lors de **validation (positive button)**. Il est aussi possible d'ajouter d'autres boutons (**neutral**, **negative**....)

```
public void declencherDialog(View composant) {
    AlertDialog.Builder alertDialogBuilder = new
AlertDialog.Builder(this);
    alertDialogBuilder.setTitle("Harry PotDeBeurre et la
Mega-SAE");
    alertDialogBuilder
        .setMessage("Tu es un développeur, Henry!")
        .setCancelable(true)
        .setPositiveButton("Oui", this::validationDialog);

    AlertDialog alertDialog = alertDialogBuilder.create();
    alertDialog.show();
}

public void validationDialog(DialogInterface dialog, int type) {
    //TO-DO
}
```

BONJOUR



Henry PotDeBeurre et la Mega-SAE
Tu es un développeur, Henry!

OUI

Boîtes de dialogues - Exemple - Choix d'une date

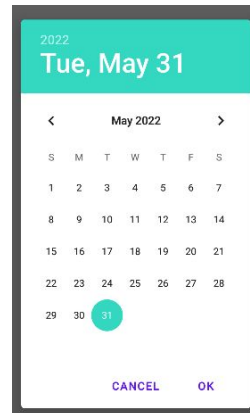
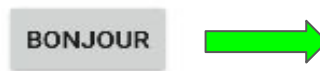
```
public void declencherDialog(View composant) {
    Calendar calendar = new
    GregorianCalendar(Locale.getDefault());

    //On règle le calendrier sur la date d'aujourd'hui
    int annee = calendar.get(Calendar.YEAR);
    int mois = calendar.get(Calendar.MONTH);
    int jour = calendar.get(Calendar.DAY_OF_MONTH);

    DatePickerDialog datePickerDialog =new DatePickerDialog(this,
    this::choixDate, annee, mois, jour);
    datePickerDialog.show();
}

public void choixDate(DatePicker datePicker, int annee, int mois,
int jour) {
    //TO-DO
    //Effectué lors du choix de la date
}
```

- On règle la **date de base** du calendrier (celle qui sera affiché lors de l'ouverture de la boîte de dialogue)
- On définit la **méthode** qui sera appelée lors du choix. Les éléments passés en **paramètres** correspondent aux informations de la date choisie



Parenthèse - Gradle

- Gradle est un **moteur de production** qui permet facilement de gérer un projet incluant diverses dépendances externes
- Un fichier nommé **build.gradle** permet de lister les adresses des **dépendances** qui devront être téléchargées dans l'environnement de développement
- Gradle est largement utilisé dans les projets **JAVA** (“concurrent” à Maven) et donc également pour les projets en **Android**. Il va permettre d’ajouter de nouveaux composants, pour l’interface, pour utiliser des services complexes, etc...

Exemple de l’import de la librairie externe “**async-http**” via un bloc dans **build.gradle**

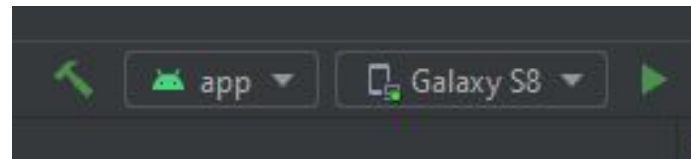
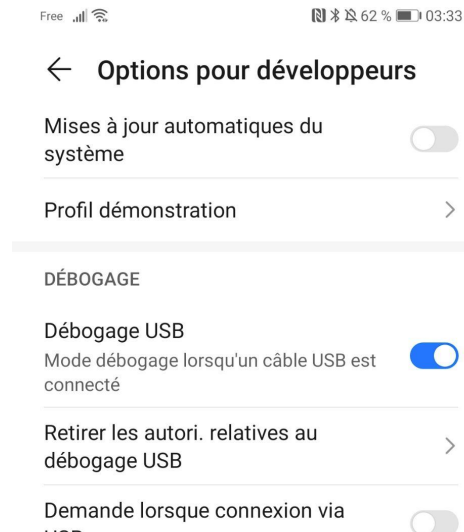
```
dependencies {  
    implementation 'com.loopj.android:android-async-http:1.4.9'  
}
```

Lancer son application

- Comme nous l'avons vu lors de l'introduction de ce cours, il n'est pas possible d'exécuter une **application Android** directement avec la **JVM** de votre ordinateur
- Pour **lancer votre application**, il y a donc trois choix possibles :
 - **Brancher votre appareil** sous Android à l'ordinateur (via un câble **USB**) et sélectionner votre téléphone comme **périphérique de lancement** sur **Android Studio**. Vous pourrez tester votre application directement sur votre téléphone. Il faut néanmoins activer le **mode développeur** sur votre appareil pour cela
 - Lancer votre application sur un **émulateur**. **Android Studio** permet d'émuler une variété de modèles d'appareils mobiles, de différentes marques, sous différentes versions d'Android, avec différentes tailles d'écrans...
 - Lancer votre application sur votre appareil via le **Wi-Fi** (fonctionnalité très récente, depuis la version 11 d'Android...nécessite aussi d'activer le **mode développeur**)

Lancer son application - En physique

- Pour pouvoir lancer votre **application** en cours de développement sur votre **appareil** sous Android, il faut avant tout activer le **mode développeur** sur votre appareil, ainsi que le **débogage USB**
- Pour cela, il faut se rendre dans “**Paramètres**”, “**A propos du téléphone**” et cliquer 7 fois sur “**Numéro de build**”. Un message apparaîtra alors et un nouveau menu sera débloqué dans “**Paramètres**”, “**Système et mises à jour**”, “**Options pour développeurs**”. Il faut alors activer “**Débogage USB**”. Le chemin peut varier selon le type d'appareil / la marque.
- Une fois connecté, le **périphérique** est sélectionnable dans une liste à côté du bouton de lancement. Lors de l'exécution, l'application se lancera directement sur votre périphérique



Lancer son application - Via un émulateur

- **Android Studio** possède un outil nommé “**AVD manager**” (Android Virtual Device Manager) permettant de créer et gérer des **émulateurs** de périphériques Android
- On peut créer des **smartphones**, des **tablettes**, des **montres connectées**, des **télévisions**, des **tableaux de bord** de voitures...
- Pour créer un nouvel émulateur, on se rend dans “**Tools**”, “**Device Manager**”, “**Virtual**”, “**Create Device**”
- On doit préciser le **modèle** d'appareil utilisé, la **version d'Android** installée sur l'appareil, et un **nom**
- Une fois l'appareil créé il sera aussi sélectionnable dans la liste à côté du bouton de lancement de l'application
- Lors de l'exécution, l'**émulateur** se lance (s'il n'est pas déjà allumé) et ouvre l'application. L'émulateur se modélise sous la forme d'une fenêtre représentant l'appareil

Lancer son application - Via un émulateur

Choose a device definition

Category	Name	Play Store	Size	Resolution	Density
TV	Pixel 4a		5,8"	1080x2340	440dpi
Phone	Pixel 4 XL		6,3"	1440x3040	560dpi
Wear OS	Pixel 4	▶	5,7"	1080x2280	440dpi
Tablet	Pixel 3a XL		6,0"	1080x2160	400dpi
Automotive	Pixel 3a	▶	5,6"	1080x2220	440dpi
	Pixel 3 XL		6,3"	1440x2960	560dpi
	Pixel 3	▶	5,46"	1080x2160	440dpi
	Pixel 2 XL		5,99"	1440x2880	560dpi
	Pixel 2	▶	5,0"	1080x1920	420dpi
	Pixel	▶	5,0"	1080x1920	420dpi
	Nexus S		4,0"	480x800	hdpi

New Hardware Profile Import Hardware Profiles Clone Device...

Pixel 2

1080px 5,0" 1920px

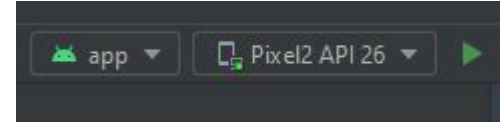
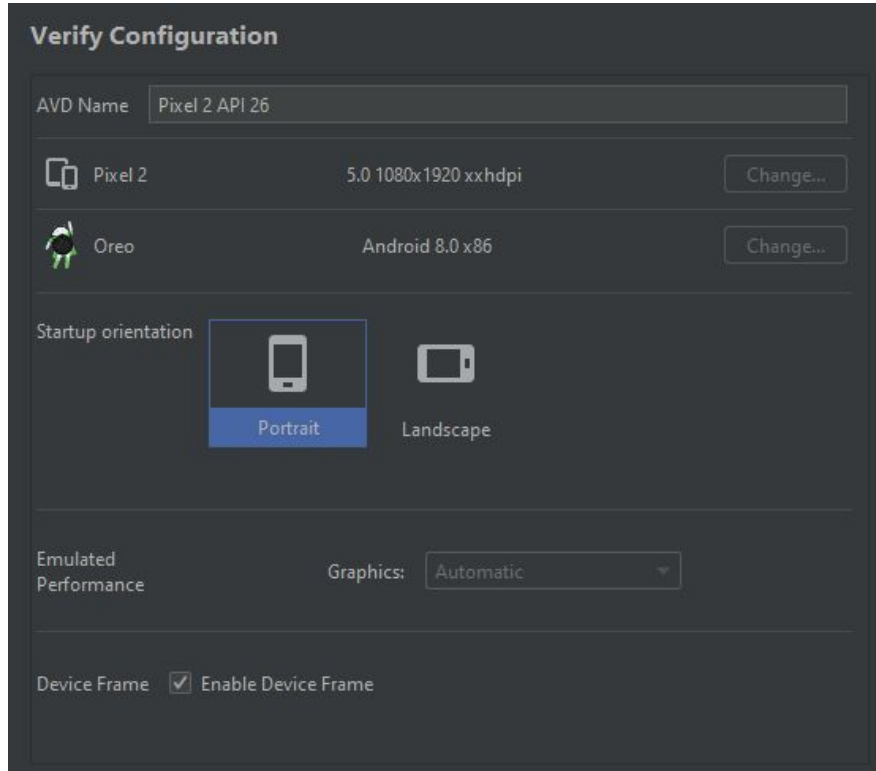
Size: large
Ratio: long
Density: 420dpi

Select a system image

Recommended x86 Images Other Images

Release Name	API Level	ABI	Target
TiramisuPrivacySandbox	TiramisuPrivacySan	x86_64	Android API TiramisuPrivacy
Tiramisu Download	Tiramisu	x86_64	Android API Tiramisu (Google
Sv2 Download	32	x86_64	Android API 32 (Google Play)
S Download	31	x86_64	Android 12.0 (Google Play)
R	30	x86	Android 11.0 (Google Play)
Q Download	29	x86	Android 10.0 (Google Play)
Pie Download	28	x86	Android 9.0 (Google Play)
Oreo Download	27	x86	Android 8.1 (Google Play)
Oreo	26	x86	Android 8.0 (Google Play)
Nougat Download	25	x86	Android 7.1.1 (Google Play)
Nougat	24	x86	Android 7.0 (Google Play)

Lancer son application - Via un émulateur



Les intents

- Un **Intent** est une **demande d'action** qui vise à lancer une **Activity** (écran d'une application) en lui fournissant des **données**
- Il y a **deux formes d'Intent** :
 - Les **Intent explicites** : la demande est réalisée en **spécifiant directement le nom de l'activité à lancer** et en **remplissant les données à fournir** à celle-ci. C'est par exemple quand on veut changer d'écran dans une application. L'activité en cours d'utilisation va réaliser un intent explicite pour passer au nouvel écran. Les données passées peuvent alors faire office de "données de construction"
 - Les **Intent implicites** : la demande est réalisée **sans spécifier le nom de l'activité à lancer**. Il s'agit d'une demande d'action "**standardisée**" (ouverture d'une image, envoi d'un message, etc...). Lorsqu'on réalise une de ces actions, le système affiche alors toutes les applications qui possèdent une activité permettant de réaliser cette action et l'utilisateur en sélectionne une. L'application doit donc spécifier au préalable l'activité qui permet de gérer ce type d'action. Comme les actions sont **standardisées**, les **données fournies à l'activité sont connues**

Les intents explicites - Lancer un nouvel écran

```
public class MainActivity extends Activity {

    private Button btn1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        this.btn1 = findViewById(R.id.btn1);
        this.btn1.setOnClickListener(this::changerEcran);
    }

    public void changerEcran(View composant) {
        Intent intentExplicite =new Intent(this, MaSuperActivity.class);

        //On passe des données à l'intent, qui seront utilisées par la nouvelle activité
        intentExplicite.putExtra("monMessage", "Ceci est un message pour le nouvel écran!");

        this.startActivity(intentExplicite);
    }
}
```

Les intents explicites - Lancer un nouvel écran

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

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

xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:orientation="vertical"
    android:gravity="center_vertical"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/messageView"

        android:layout_gravity="center_horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</LinearLayout>
```

```
public class MaSuperActivity extends Activity {

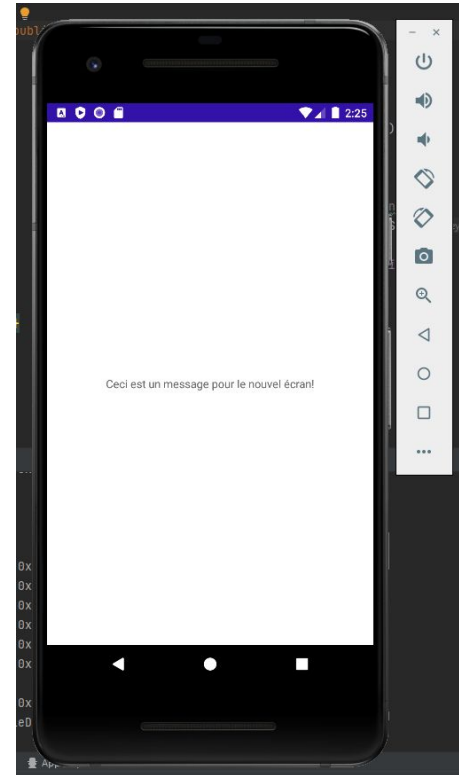
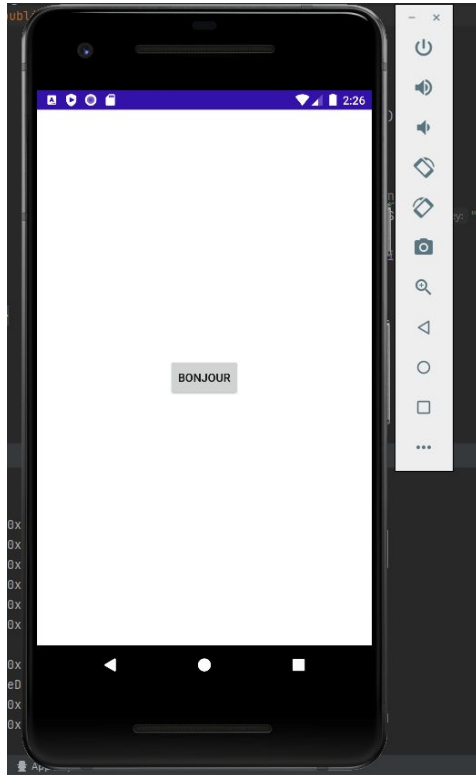
    private TextView messageView;

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

        //On récupère les données passées dans l'intent...
        String msg = this getIntent().getExtras().getString("monMessage");

        this.messageView = findViewById(R.id.messageView);
        this.messageView.setText(msg);
    }
}
```

Les intents explicites - Lancer un nouvel écran



Les intents implicites - Proposer un service

- On va proposer un service basique permettant d'**afficher une image** dans l'application quand on sélectionne l'action "**partager**" sur une image (dans la galerie, ou en ligne...)
- On commence par modifier notre activité pour qu'elle puisse accueillir une **image** :

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

    <ImageView
        android:id="@+id/imageView"
        android:layout_gravity="center_horizontal"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```


Les intents implicites - Proposer un service

- Dans le fichier **AndroidManifest.xml** qui liste toutes les **activités** de l'application, on rajoute un **"intent-filter"** sur notre activité qui permettra au système de détecter qu'elle peut ouvrir une image
- Le type d'intent que l'on vise est **android.intent.action.send** (partage de contenu). On va cibler spécifiquement les **images**

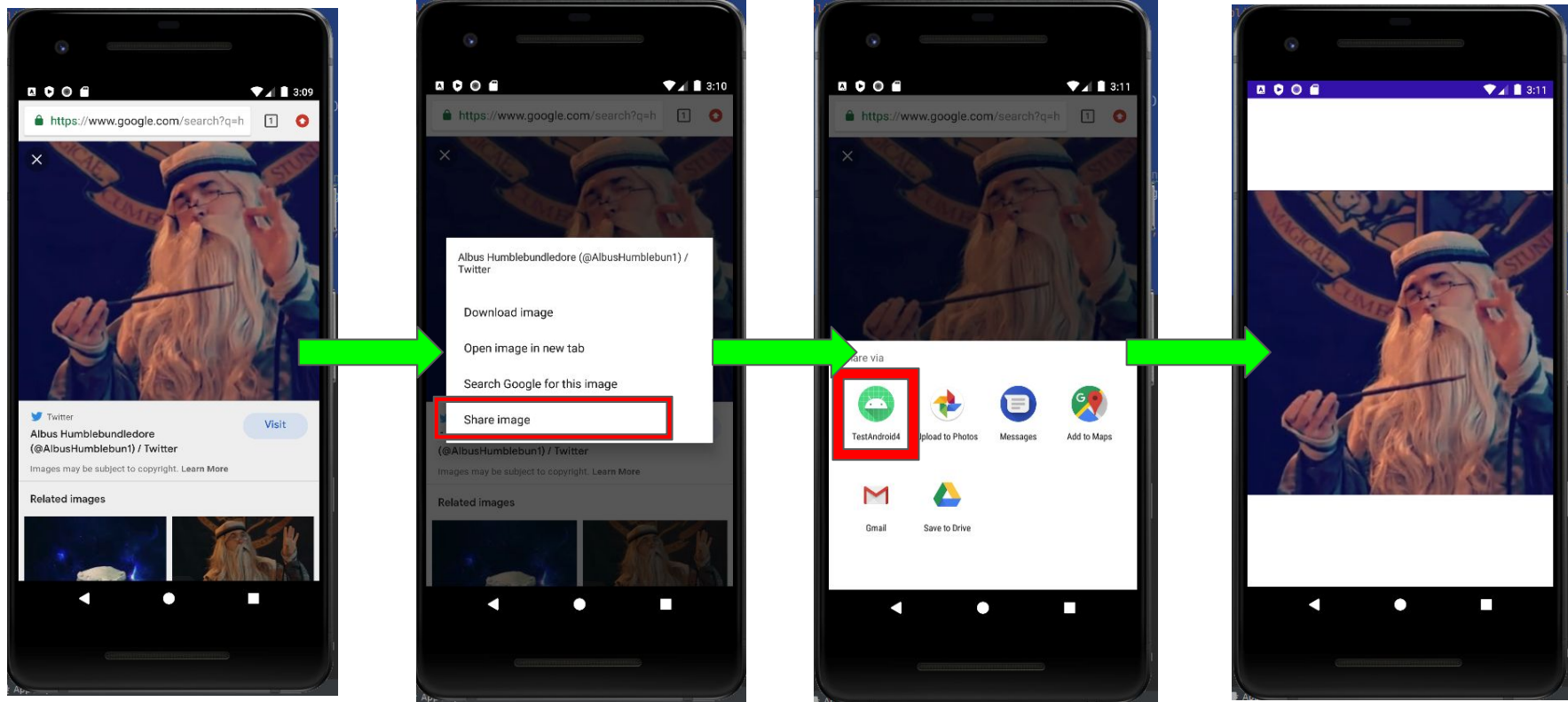
```
<activity
    android:name=".MaSuperActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="image/*"/>
    </intent-filter>
</activity>
```

Les intents implicites - Proposer un service

- Lorsqu'un intent "**SEND**" est émis avec un contenu **image**, une donnée "**Intent.EXTRA_STREAM**" est fournie. Elle donne un objet de type **URI** qui donne le chemin vers l'image
- Une fois l'**URI** récupérée, il ne nous reste plus qu'à l'affetcer à notre **ImageView**

```
public class MaSuperActivity extends Activity {  
  
    private ImageView imageView;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_ma_super);  
  
        //On récupère les données passées dans l'intent...  
        Uri uri = (Uri) this.getIntent().getExtras().get(Intent.EXTRA_STREAM);  
  
        this.imageView = findViewById(R.id.imageView);  
  
        this.imageView.setImageURI(uri);  
    }  
}
```

Les intents implicites - Proposer un service



Ressources de l'application

- Un projet **Android** stocke toutes les **ressources** utiles (**images, icônes, layouts...**) dans un dossier “**res**” organisé selon une **architecture** bien précise
- Grâce à cet **architecture**, il est notamment possible de :
 - Proposer une même image dans différentes **résolutions**. La version de l'image sera automatiquement sélectionnée selon la taille de l'écran
 - Proposer un **layout** (interface XML) différent selon **l'orientation de l'appareil** (une version “paysage” (landscape) alternative par exemple)
 - Et bien d'autres choses...!

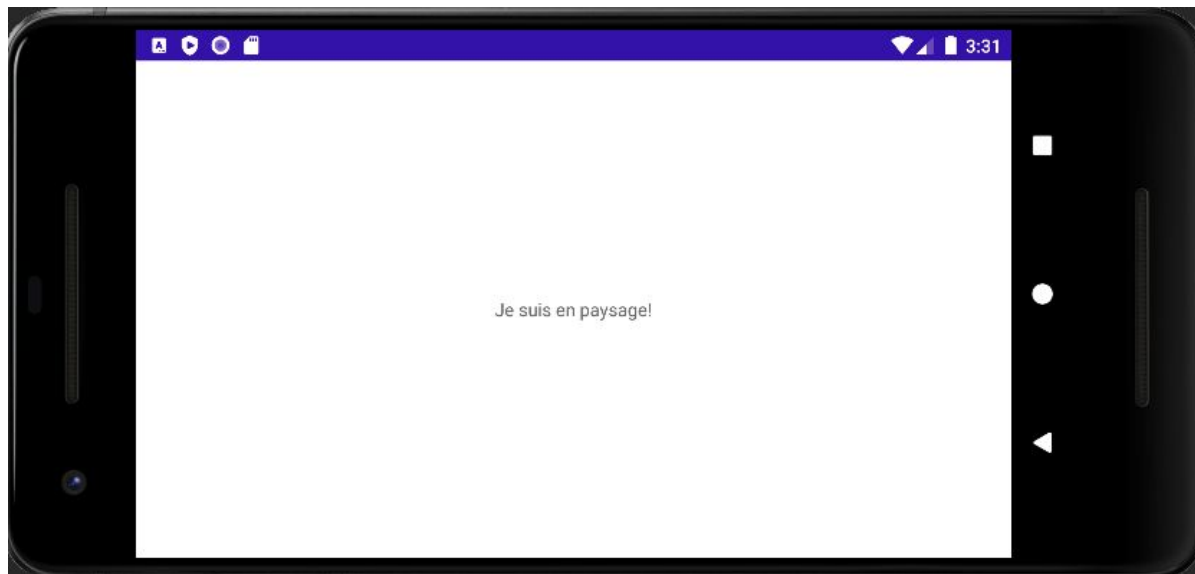
Ressources de l'application



```
<TextView
    android:layout_width="wrap content"
    android:layout_height="wrap content"
    android:layout_gravity="center horizontal"
    android:text="Je suis en portrait" />
```

```
<TextView
    android:layout_width="wrap content"
    android:layout_height="wrap content"
    android:layout_gravity="center horizontal"
    android:text="Je suis en paysage!" />
```

Ressources de l'application



Internationalisation

- **Android** fournit un système **d'internationalisation** très simple d'utilisation
- Le principe est le suivant :
 - Au lieu de coder du texte en dur, on utilise un **identifiant de string** dans le champ textuel, ainsi : **"@string/nomIdentifiant"**
 - On spécifie la **valeur par défaut** de la chaîne dans un fichier **"strings.xml"** disponible dans **"res/values"**. Il s'agira de la langue "par défaut" utilisée
 - Quand on veut proposer une nouvelle langue, on crée un dossier **"values-codelangue"** dans **res** où l'on place un nouveau fichier **"strings.xml"** dans lequel on spécifie la valeur du texte correspondant à chaque identifiant dans la langue souhaitée. Par exemple, **"values-fr-rFR"**
 - Lorsque l'utilisateur changera la **langue du système**, l'application se chargera en utilisant le fichier **strings.xml** du dossier de langue correspondant, ou, s'il ne le trouve pas, le fichier par défaut (**values/strings.xml**)

Internationalisation

```
<TextView android:id="@+id/btn1"  
    android:layout width="wrap content"  
    android:layout height="wrap content"  
    android:layout gravity="center_horizontal"  
    android:text="@string/hello"/>
```

values/strings.xml

```
<resources>  
    <string name="app name">TestAndroid4</string>  
    <string name="hello">Hello!</string>  
</resources>
```

values-fr-rFR/strings.xml

```
<resources>  
    <string name="app name">TestAndroid4</string>  
    <string name="hello">Bonjour!</string>  
</resources>
```


Internationalisation

- Si on a besoin d'accéder à la valeur internationalisée d'une chaîne de caractères côté **JAVA** (dans une **Activity**), on peut faire cela utilisant l'**identifiant** du string et la fonction getString :

```
this.getString(R.string.identifiant) ;
```

Exporter son application - Le fichier .apk

- Le fichier **.apk** correspond à "**l'exécutable**" de votre **application** pour votre appareil Android (équivalent Android d'un **.jar**, si vous voulez)
- Pour le générer avec **Android Studio**, vous devez vous rendre dans "**Build**", "**Build Bundle(s) / APK(s)**", "**Build APK(s)**"
- L'APK est générée dans le dossier "**build/outputs/apk/debug**". Par défaut, elle porte le nom de "**app-debug.apk**"
- C'est ce genre de fichier qui est téléchargé et installé sur vos appareils depuis l'**app store**. C'est également ce genre de fichier que vous devez fournir si vous souhaitez publier votre application sur le store



Point sur les API

- Nous avons plusieurs fois, tout au long de l'année, parlé des **APIs**, sans jamais vraiment définir ce terme
- Une **API (Application Programming Interface)** est un service qu'un développeur peut appeler pour **réaliser une action, obtenir une donnée**
- Le développeur envoie une **requête** selon un format spécifique définie par **l'API**, et **l'API** exécute des actions et lui renvoie un résultat. Tout se passe en “**boîte noire**”, le développeur n'a pas conscience de ce qui se déroule en interne pour produire le résultat
- Il existe diverses **APIs**, par exemple, des **bibliothèques logicielles**. Nous avons vu la bibliothèque logicielle “graphviz” permettant de dessiner des graphes. On fournit des données du graphe à l'API et elle nous renvoie l'image du graphe
- Il existe d'autres types d'API modélisées par des **services web**. Il s'agit d'applications web dont le but est de fournir des données en résultat de demandes émises par le développeur, sous la forme de **requêtes HTTP**. Ainsi, tout le monde peut se connecter à ces services et les exploiter dans sa propre application. C'est souvent le cas avec des applications mobiles (par exemple, pour la météo, avec la SNCF pour collecter les données sur les trains...)
- Il existe des API gratuites (voir [ici](#)) et des API payantes, ou nécessitant un forfait / abonnement (par exemple, l'API Google Translate...)

Point sur les API - Exemple - API de citations d'animés

- [Animechan](#) : **API** (web) permettant de récupérer des citations d'animés / de personnages d'animés
- Retourne des données au format **JSON**

Par exemple, les citations de **Luffy** :

<https://animechan.vercel.app/api/quotes/character?name=luffy>

```
▼ 0:
  anime:    "One Piece"
  character: "Monkey D. Luffy"
  ▼ quote:  "Forgetting is like a wound. The wound may heal but it has already left a scar."
▼ 1:
  anime:    "One Piece"
  character: "Monkey D. Luffy"
  ▼ quote:  "Then just become stronger. I have my ambition, you have your ambition too. Which means you should just keep walking forward towards that goal."
▼ 2:
  anime:    "One Piece"
  character: "Monkey D. Luffy"
  quote:    "Being alone is more painful than getting hurt."
```

Point sur les API - Développer sa propre API

- Développer une **API** revient à développer une **application web** côté **backend**
- **Pas de server-rendering**, on exporte seulement les données (généralement, en **JSON**)
- Cela peut donc être fait avec PHP, ou des frameworks comme **Node** ou bien **Symfony** avec un composant adapté.
- Il existe diverses formes d'API, mais de nos jours l'architecture la plus généralisée est l'**API REST**, qui impose des règles sur la formation des routes et permet d'exploiter toutes les **méthodes HTTP** (**GET, POST, PUT, PATCH, DELETE...**)
- Au niveau de **Symfony**, la tendance est plutôt à l'utilisation de l'outil **API Platform** qui est très puissant et permet de créer une **API REST** assez simplement, en codant directement les règles métiers dans les classes entités, via des **annotations**. Nous aurons l'occasion de découvrir cette technologie lors des prochaines séances.

Bases de données en temps réel

- Le service **Firebase** permet de créer des bases de données en temps réel, souvent utilisées dans le cadre d'applications mobiles.
- Ces bases sont “**en temps réel**” dans le sens où elles permettent automatiquement **notifier** les clients (programmes) abonnés à cette base lorsqu'une mise à jour intervient (ajout d'une donnée, modification, suppression ,etc...)
- Il n'y a pas d'intermédiaire **backend** (géré par le développeur), le programme client se connecte et s'abonne directement à la base et est authentifié via un système de **JWT** (dont nous reparlerons)
- Ce service permet également l'authentification (et l'inscription) automatique sur une application en utilisant son compte Google, par exemple.
- On va généralement utiliser cette technologie pour des fonctionnalités où recevoir l'information “en direct” (sans avoir à faire de requêtes) est important : **chat, notifications, progression d'une commande**, etc...
- Il est possible d'utiliser à la fois une (ou plusieurs) **API** et avoir une autre partie des fonctionnalités qui dépendent d'une base de données **firebase**, au besoin.
- Les données sont hébergées avec du **NoSQL** (ce n'est donc pas une base relationnelle) et est hébergée en **cloud** sur les serveurs dédiés de **Google** (la base nous vous appartient donc pas vraiment)