

<p><b>Installation des outils</b> Pré-requis et installation</p>  <p>Jérôme CLAMENT-SANZ</p>	<p><b>Introduction à l'univers Android</b> Système d'exploitation mobile fondé sur le noyau Linux et développé actuellement par Google</p>  <p>Jérôme CLAMENT-SANZ</p>	<p><b>Environnement de développement</b> Android Studio est un environnement de développement pour développer des applications mobiles Android. Il est basé sur IntelliJ IDEA et utilise le moteur de production Gradle.</p>  <p>Jérôme CLAMENT-SANZ</p>	<p><b>Ressources</b> Les ressources sont des fichiers qui contiennent des informations utiles pour l'application, telles que des images, des chaînes de caractères, des couleurs ou des styles.</p>  <p>Jérôme CLAMENT-SANZ</p>
<p><b>Les layouts</b> Un layout définit la structure d'une interface utilisateur dans votre application.</p>  <p>Jérôme CLAMENT-SANZ</p>	<p><b>Interagir avec les vues</b> Assesseurs, mutateurs et listeners</p>  <p>Jérôme CLAMENT-SANZ</p>	<p><b>Les activités</b> Une activité est une tâche unique et ciblée que l'utilisateur peut faire. Presque toutes les activités interagissent avec l'utilisateur, la classe d'activité se charge de l'affichage dans laquelle vous pouvez placer votre interface utilisateur avec setContentView(View).</p>  <p>Jérôme CLAMENT-SANZ</p>	<p><b>Les fragments</b> Un fragment est un composant indépendant qui doit être utilisé dans une activité. Un fragment encapsule des fonctionnalités qui le rendent facile à réutiliser dans une activité ou dans une mise en page.</p>  <p>Jérôme CLAMENT-SANZ</p>
<p><b>Architecture MVVM</b> Comment avoir une belle architecture dans son code ?</p>  <p>Jérôme CLAMENT-SANZ</p>	<p><b>Les permissions</b> Le but d'une autorisation est de protéger la vie privée d'un utilisateur. L'autorisation permet à l'application de demander l'autorisation d'accéder aux données utilisatrices sensibles (tels que les contacts, la localisation, les photos, etc.). Les fonctionnalités du système telles que l'appareil photo et Internet doivent être autorisées par l'utilisateur pour qu'il puisse approuver la demande.</p>  <p>Jérôme CLAMENT-SANZ</p>	<p><b>Les librairies</b> Le but d'une librairie est de ne pas réinventer la roue.</p>  <p>Jérôme CLAMENT-SANZ</p>	<p><b>SharedPreferences</b> Stockez les préférences de son app facilement.</p>  <p>Jérôme CLAMENT-SANZ</p>
<p><b>Les listes avec RecyclerView</b> Afficher des listes propres et scrolables à l'infini et de manière fluide!</p>  <p>Jérôme CLAMENT-SANZ</p>	<p><b>Communiquer avec une API grâce à Retrofit2</b> Qu'est-ce que la gestion des APIs? Le format standard de communication est JSON, appelé via des REST. Cependant, il existe d'autres formats, comme SOAP, OData et XML. JSON est devenu le standard par défaut. Une API JSON peut être facilement intégrée par des projets d'applications Web ou mobiles.</p>  <p>Jérôme CLAMENT-SANZ</p>	<p><b>Base de données avec Room</b> Gérer la persistance de données dans son app</p>  <p>Jérôme CLAMENT-SANZ</p>	<p><b>Tests unitaires</b> Tester le bon fonctionnement d'une partie précise d'un programme</p>  <p>Jérôme CLAMENT-SANZ</p>



# Installation des outils

---

Pré-requis et installation



# Installation des outils

- Java Development Kit (JDK)
- Environnement de développement Android Studio : téléchargement des API et outils grâce au SDK intégré à Android Studio
- Optionnel : git pour le versioning

Google search results for "android studio". The search bar shows "android studio". Below it, there are tabs for "Tous", "Vidéos", "Images", "Actualités", "Livres", "Plus", "Paramètres", and "Outils". The main content area shows "Environ 1050 000 000 résultats (0,38 secondes)". A top result is "Download Android Studio and SDK tools | Android Developers" with a link to <https://developer.android.com/studio/>. Below it, there are sections for "Install Android Studio" and "Meet Android Studio".

Google search results for "jdk". The search bar shows "jdk". Below it, there are tabs for "Tous", "Vidéos", "Images", "Actualités", "Shopping", "Plus", "Paramètres", and "Outils". The main content area shows "Environ 21 300 000 résultats (0,41 secondes)". The first result is "Java SE Development Kit 8 - Downloads - Oracle" with a link to <https://www.oracle.com/.../java/.../jdk8-downloads-2133151.html>. Below it, there are other results for Java SE Downloads and Java SE Development Kit 11.



# Conditions initiales

- **Pour tout le monde :**
  - Minimum 16 Go de mémoire RAM (32 idéalement)
  - Plus de 30 Go d'espace disque
  - Pas de minima pour le processeur (émulation sur un cœur)
- **Spécificités par OS :**
  - Windows : Vista ou plus récent
  - Mac OS : Mac OS 10.8.5 ou plus récent
  - Linux : bibliothèque GNU C (glibc) version 2.15 ou plus récent



# Configuration du téléphone

- Par étapes :

1. Accéder aux **paramètres** du téléphone
2. Aller dans « **à propos du téléphone** »
3. Tout en bas, appuyer **7 fois sur « numéro de build »**
4. Retourner à nouveau dans le **menu principal** des paramètres du téléphone
5. Aller dans la nouvelle catégorie « **options pour les développeurs** »
6. Activer le « **débogage USB** »



# Se dokumenter

developer.android.com

The developer.android.com homepage features several sections: 'FEATURED New publishing features in Google Play Console' (with a 'PAIN MORE' button), 'FEATURED Apps, Games, & Insights Podcast' (with a 'LISTEN NOW' button), 'FEATURED What's new in Android games' (with a 'LEARN MORE' button), and a central yellow banner 'Learn how to publish your first app using Google Play'. Below these are 'Latest news' and 'MORE NEWS' sections.

The 'Kotlin' page on developer.android.com introduces the language with the heading 'Develop Android apps with Kotlin'. It includes a 'GET STARTED' button and four sections: 'Expressive and concise', 'Safer code', 'Interoperable', and 'Structured concurrency', each with a brief description and a 'LEARN MORE' button.

The 'News' page displays a grid of 12 news items from various categories: 'Kotlin', 'Java', 'Android', 'APIs', 'Design', 'Performance', 'Security', 'Testing', 'UI/UX', 'IDE', and 'Project'. Each item has a thumbnail, title, and a 'READ MORE' button.

The 'Jetpack' page highlights 'Android app quality: Visual experience' and features three cards: 'CameraX', 'DataStore in Alpha', and 'RecyclerView'. It also includes a section titled 'Why use Android Jetpack?'.

The 'Documentation for app developers' page provides links to 'Get started', 'Android devices', 'Best practices', 'Core developer topics', and 'Design guides'.

The 'Build anything on Android' page includes sections for 'Android Development Resources for Educators', 'Apps, Games, & Insights Podcast', and 'Apps, Games, & Insights'.



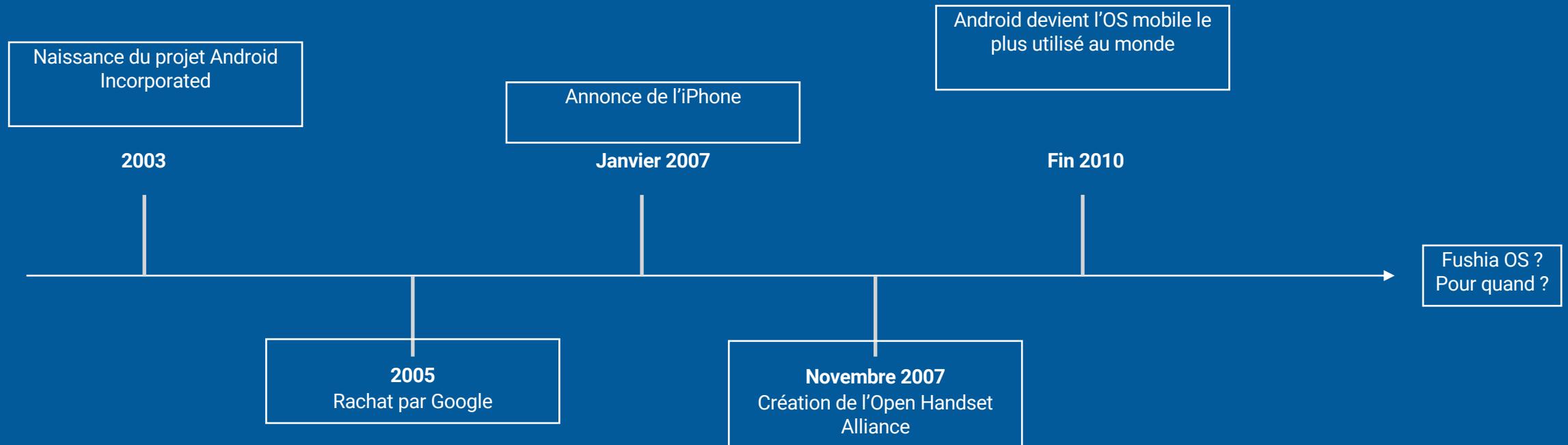
# Introduction à l'univers Android

---

Système d'exploitation mobile fondé sur le noyau  
Linux et développé actuellement par Google



# Les moments historiques



# Philosophie et avantages

## Open source

Vous pouvez à tout moment télécharger les sources et les modifier selon vos goûts. Android utilise aussi des bibliothèques open source puissantes, comme par exemple SQLite pour les bases de données et OpenGL.

## Facile à développer

Toutes les API mises à disposition sont complètes, faciles d'accès et d'utilisation. De manière un peu caricaturale, on peut dire que vous pouvez envoyer un SMS en seulement deux lignes de code.

## Gratuit (ou presque)

Android est gratuit. S'il vous prenait l'envie de produire votre propre téléphone sous Android, alors vous n'auriez même pas à payer une quelconque licence.

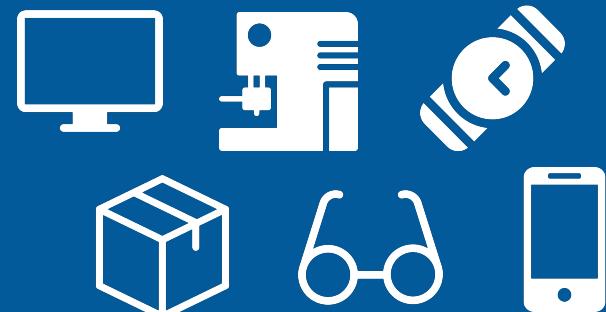
En revanche, pour poster autant d'applications sur le Play Store que vous le souhaitez, il vous en coûtera la modique somme de 25\$.

## Facile à vendre

Le Play Store est une plateforme immense et très visitée. C'est une grande opportunité pour quiconque veut y diffuser une application.

## Flexible

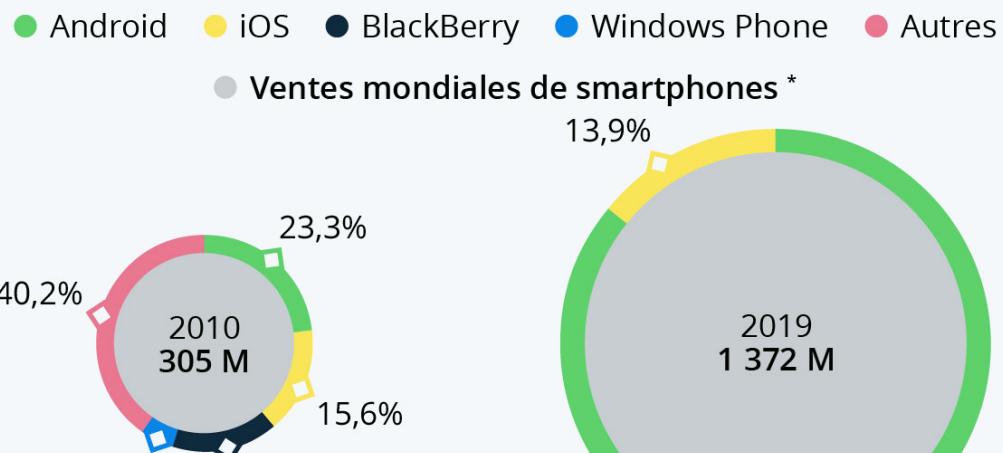
Le système est extrêmement portable, il s'adapte à beaucoup de structures différentes. De plus il est construit de manière à faciliter le développement et la distribution en fonction des composants en présence dans le terminal (si votre application nécessite d'utiliser le Bluetooth, seuls les terminaux équipés de Bluetooth pourront la voir sur le Play Store).



# Parts de marché des OS mobiles

## Android et iOS : un solide duopole

Estimation des parts de marché des systèmes d'exploitation pour smartphones



\* unités

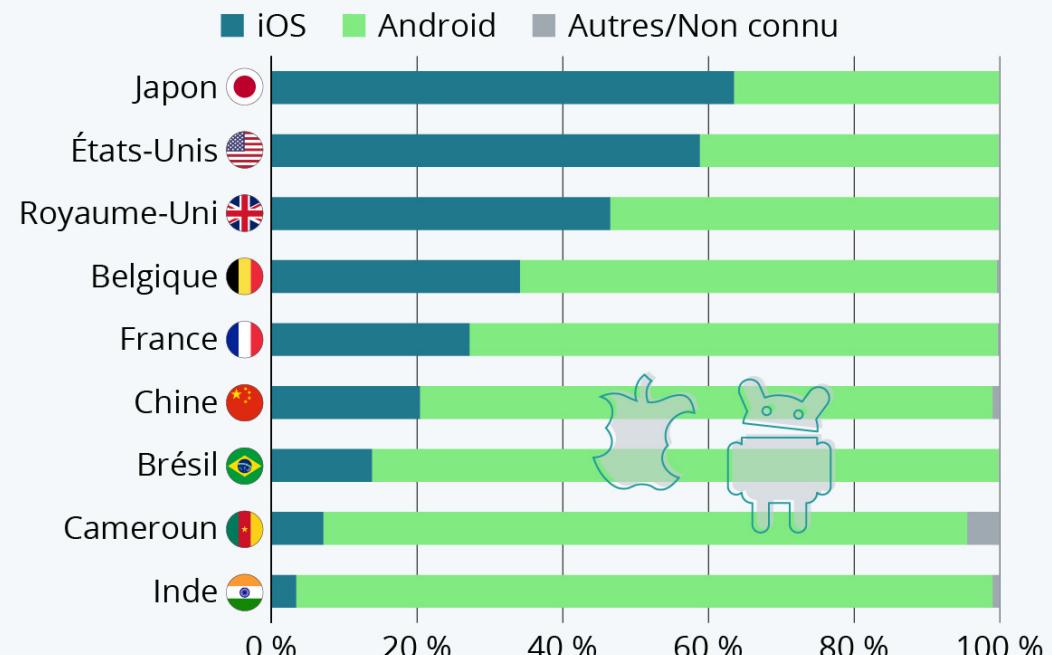
Source : IDC



statista

## Plutôt Apple ou Android ?

Part de marché des systèmes d'exploitation mobiles dans une sélection de pays (juillet 2020)



Source : StatCounter



statista



# Versions d'Android

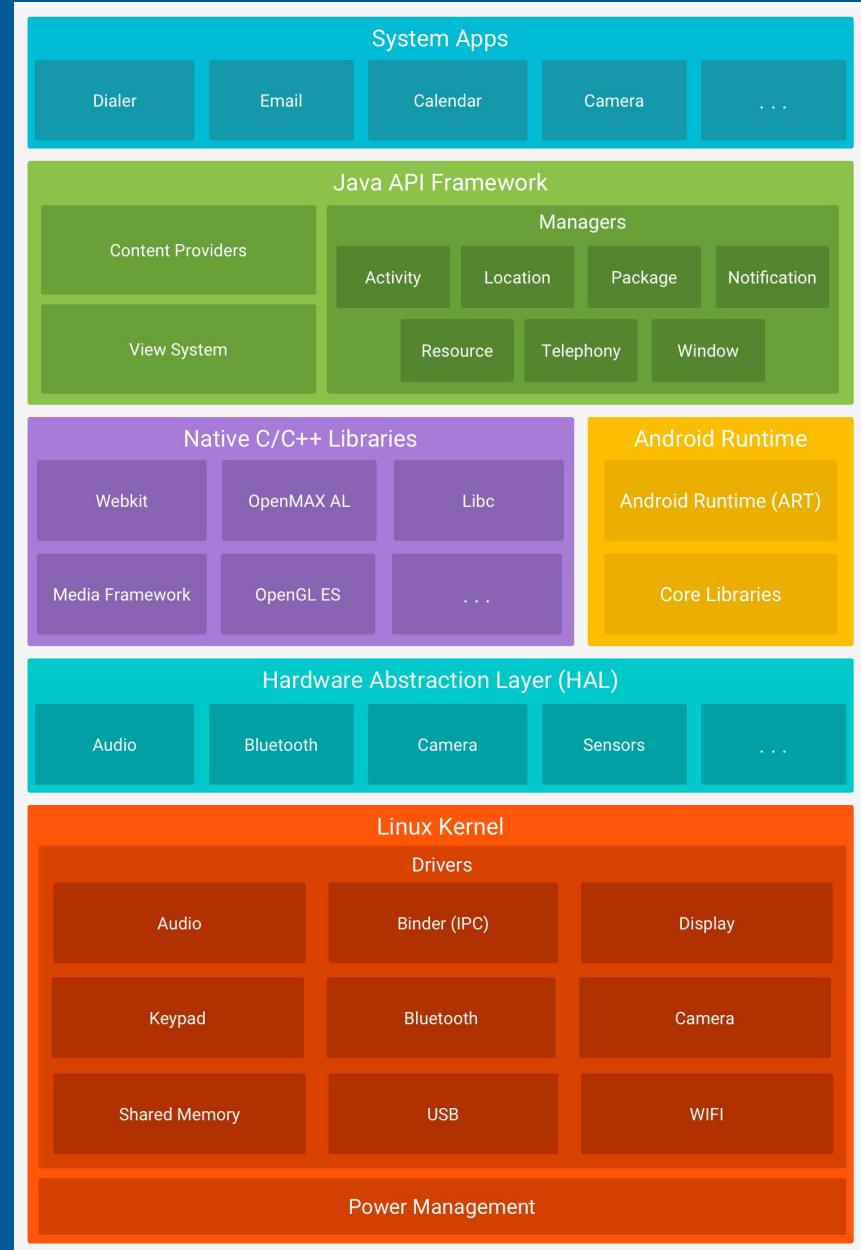


ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0 Ice Cream Sandwich	15	
4.1 Jelly Bean	16	99,8%
4.2 Jelly Bean	17	99,2%
4.3 Jelly Bean	18	98,4%
4.4 KitKat	19	98,1%
5.0 Lollipop	21	94,1%
5.1 Lollipop	22	92,3%
6.0 Marshmallow	23	84,9%
7.0 Nougat	24	73,7%
7.1 Nougat	25	66,2%
8.0 Oreo	26	60,8%
8.1 Oreo	27	53,5%
9.0 Pie	28	39,5%
10. Android 10	29	8,2%



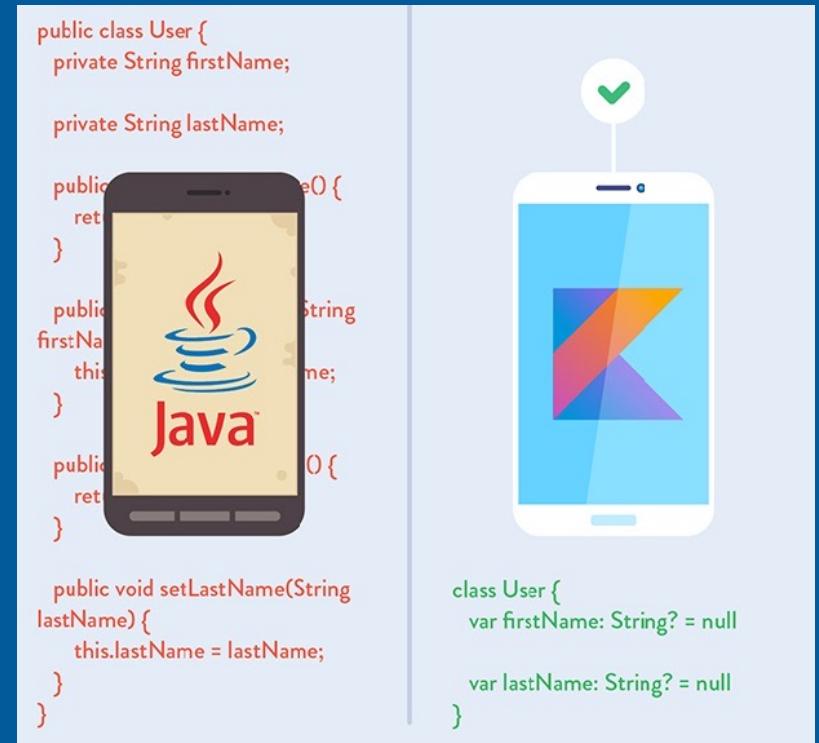
# Comment ça fonctionne Android ?

- **Linux Kernel** : la base de la plate-forme Android est le noyau Linux.
- **HAL** : La couche d'abstraction matérielle (HAL) fournit des interfaces standard permettant d'utiliser les capacités matérielles des périphériques à l'API Java de niveau supérieur.
- **ART** : chaque application s'exécute dans son propre processus et avec sa propre instance d'Android Runtime (ART). ART est écrit pour exécuter plusieurs machines virtuelles sur des périphériques à faible mémoire en exécutant des fichiers DEX, un format de bytecode conçu spécialement pour Android et optimisé pour une empreinte mémoire minimale.



# Code, compilation et exécution

- Avec Android Studio, on peut coder en :
  - Java
  - Kotlin
  - C++
- À la compilation, le code Java/Kotlin est traduit en **bytecode**
- Il est ensuite exécuté et lu pour la machine virtuelle **ART**



# Machine virtuelle ART

- ART est né à cause d'un procès opposant Oracle à Google (car **Dalvik** enfreindrait des brevets d'Oracle)
- Au contraire de **Dalvik**, **ART** utilise la compilation anticipée en compilant l'application à son installation, sans besoin ultérieur d'interprétation
- La compilation anticipée rend les apps un peu plus lourdes en terme de stockage et d'installation.
- Cependant, **ART** augmente les performances et donc la durée de vie de la batterie
- À partir de la version Android 5.0, **Dalvik** est entièrement remplacé par **ART**
- **ART** génère toujours du **bytecode Dalvik** mais les fichiers .odex sont remplacés par des **ELF** (*Executable and Linkable Format*)



# Environnement de développement

---

Android Studio est un environnement de développement pour développer des applications mobiles Android. Il est basé sur IntelliJ IDEA et utilise le moteur de production Gradle.

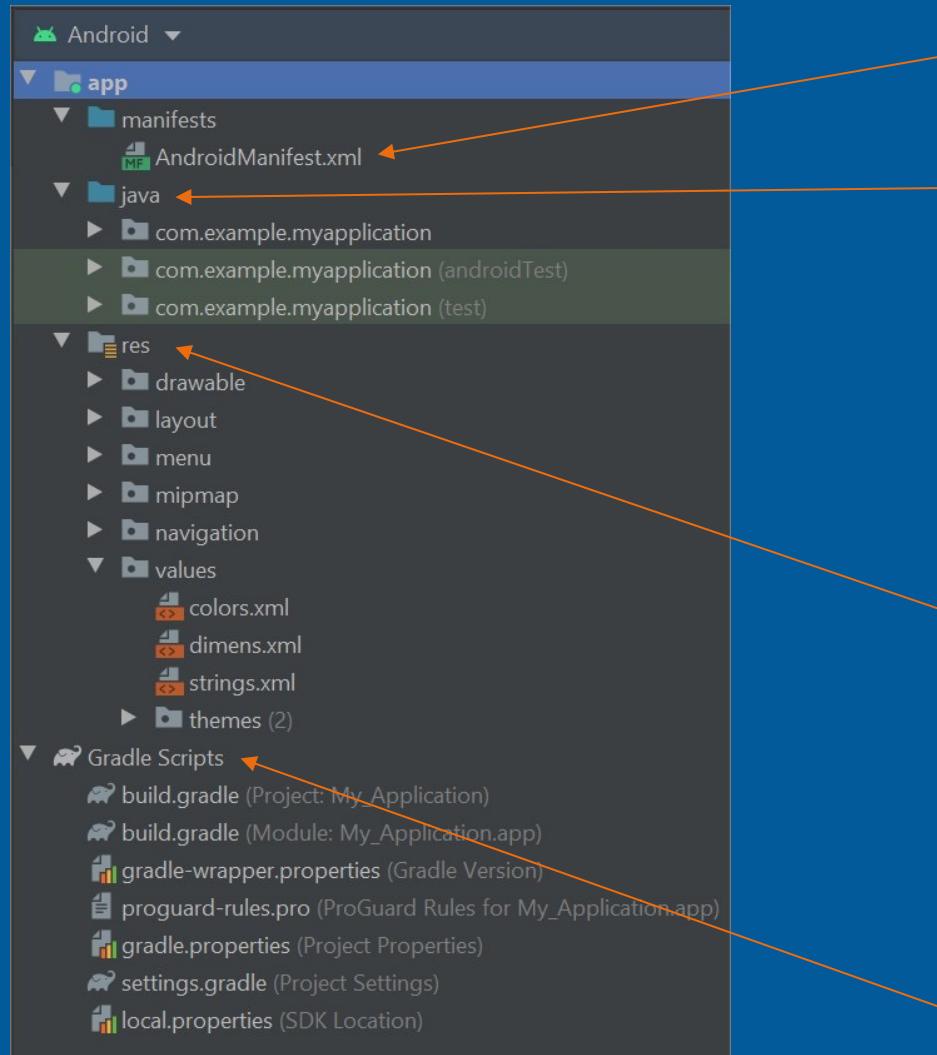


# Création d'un projet

- Premier écran
  - Application Name :
    - Nom qui apparaîtra sur l'appareil et sur Google Play
  - Company Domain :
    - Identifiant de l'application (ne peut être changé une fois l'app publiée)
  - Project Location :
    - Emplacement où les fichiers du projet seront créés
- Deuxième écran
  - Sélection de l'API
- Troisième écran
  - Création de la première Activity
- Dernier écran
  - Informations sur la première activité de notre application



# Composition d'un projet Android (vue simplifiée par Android Studio)



Déclaration d'informations liées à l'application (nom de l'app, icône, permissions, activités, services,...)

Code Java/Kotlin (les deux packages surlignés du dessous correspondent aux classes de test)

Ressources de l'app (liste non exhaustive) :

- **animator** : Fichiers XML pour les animations
- **anim** : Fichiers XML qui définissent des animations entre deux. (Les animations de propriété peuvent également être enregistrées dans ce répertoire, mais le répertoire animator/ est préférable pour les animations de propriété afin de distinguer les deux types.)
- **color** : Fichiers XML décrivant les couleurs
- **drawable** : Fichiers bitmaps (.png, .9.png, .jpg, .gif) ou XML qui font référence à des Bitmap files, SVG, Nine-Patches (re-sizeable bitmaps), State lists, Shapes, Animation drawables, autres drawables
- **mipmap** : Drawables des différentes launcher icons.
- **layout** : XMLs des vues
- **menu** : XMLs de définition des menus
- **navigation** : XMLs de définition de la navigation dans l'app
- **raw** : Tout ce qui ne rentre pas dans les autres dossiers (musiques, vidéos, Jsons,...). Si vous souhaitez mettre en place une hiérarchie de fichiers, utilisez assets/ (à la racine de app/).
- **values** : XMLs de valeurs comme les strings, integers, dimens, styles, colors,...
- **xml** : XMLs ne rentrant pas dans les autres dossiers
- **font** : Polices

Gradle est utilisé pour construire et gérer des projets Android avec le langage Groovy.



# Android Manifest

Le fichier `AndroidManifest.xml` décrit les informations essentielles de votre application. Ce fichier est utilisé par :

- les outils de construction Android
- le système d'exploitation
- le Google Play

```
<manifest ... >
    <application ... >
        <activity android:name="com.example.myapp.MainActivity" ... >
            </activity>
        </application>
    </manifest>
```

Dans ce fichier, on peut y déclarer de nombreuses informations (activités, permissions, services, receivers, intent-filters, actions, etc.). La liste complète est disponible à cette adresse :  
<https://developer.android.com/guide/topics/manifest/manifest-intro#reference>

Exemple de déclaration d'une permission dans le fichier `AndroidManifest.xml` :

```
<manifest ... >
    <uses-permission android:name="android.permission.SEND_SMS" />
    ...
</manifest>
```

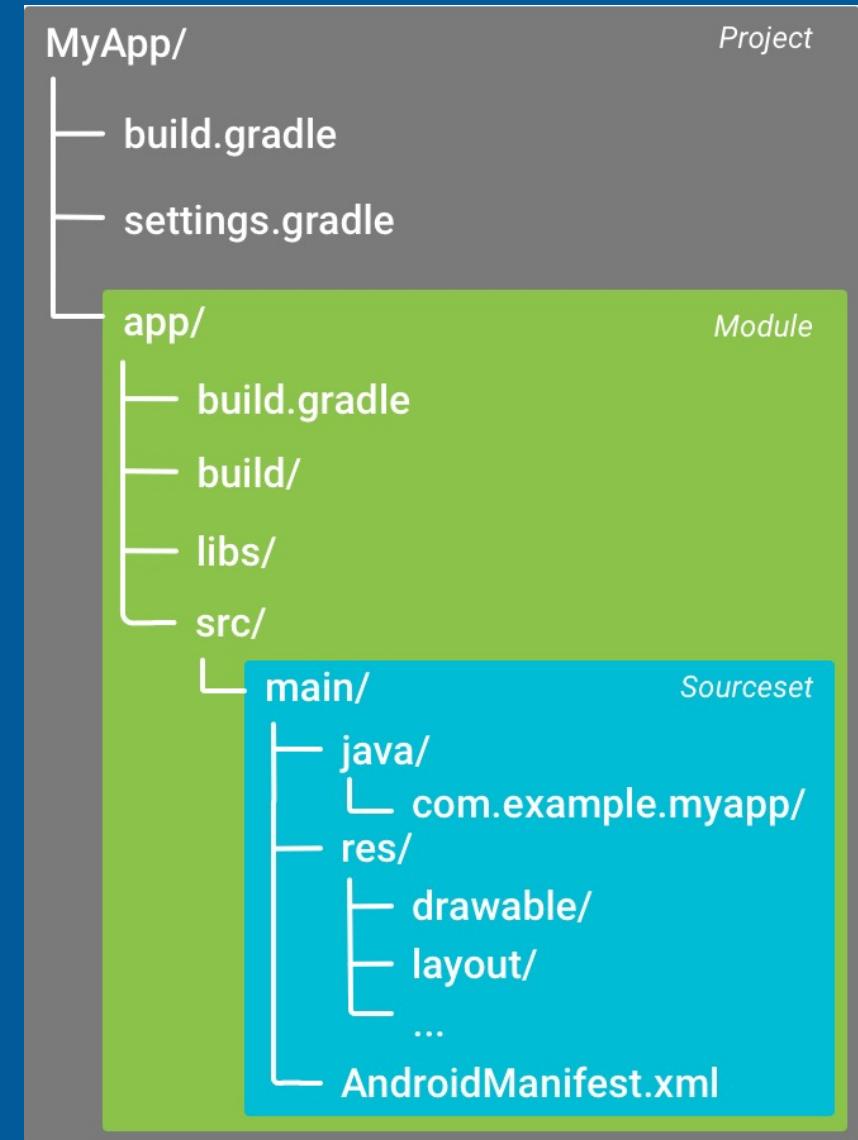


# Structure par défaut d'un projet

La création de configurations de build personnalisées nécessite que vous apportiez des modifications à un ou plusieurs fichiers de configuration de build ou aux fichiers build.gradle.

Ces fichiers de texte brut utilisent le langage DSL (Domain Specific Language) pour décrire et manipuler la logique de construction à l'aide de Groovy, qui est un langage dynamique pour la machine virtuelle Java (JVM).

Vous n'avez pas besoin de connaître Groovy pour commencer à configurer votre build, car le plug-in Android pour Gradle introduit la plupart des éléments DSL dont vous avez besoin.



Lors du démarrage d'un nouveau projet, Android Studio crée automatiquement certains fichiers de configuration.



# build.gradle

Le système de build Android compile les ressources de l'application et le code source, et les conditionne dans des APK que vous pouvez tester, déployer, signer et distribuer.

Android Studio utilise Gradle, une boîte à outils de construction avancée, pour automatiser et gérer le processus de construction, tout en vous permettant de définir des configurations de construction personnalisées flexibles. Chaque configuration de build peut définir son propre ensemble de code et de ressources, tout en réutilisant les parties communes à toutes les versions de votre application. Le plugin Android pour Gradle fonctionne avec la boîte à outils de construction pour fournir des processus et des paramètres configurables spécifiques à la création et au test d'applications Android.

```
1 apply plugin: 'com.android.application' ←
2
3 android {
4     compileSdkVersion 24
5     buildToolsVersion "23.0.3"
6     defaultConfig {
7         applicationId "app.myapplication"
8         minSdkVersion 19
9         targetSdkVersion 24
10        versionCode 1
11        versionName "1.0"
12        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
13    }
14    buildTypes {
15        release {
16            minifyEnabled false
17            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
18        }
19    }
20 }
21
22 dependencies {
23     compile fileTree(dir: 'libs', include: ['*.jar'])
24     androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
25         exclude group: 'com.android.support', module: 'support-annotations'
26     })
27     compile 'com.android.support:appcompat-v7:24.1.0'
28     testCompile 'junit:junit:4.12'
29 }
30 }
```

Permet d'appliquer des plugins au module

- **compileSdkVersion** : niveau d'API Android que Gradle doit utiliser pour compiler (l'app a accès aux fonctionnalités de l'API incluses dans ce niveau d'API et inférieur)
- **minSdkVersion** : niveau d'API minimum requis pour exécuter l'application
- **targetSdkVersion** : niveau d'API utilisé pour tester l'application
- **versionCode** : numéro de version de votre application
- **versionName** : nom de version convivial pour votre application
- **applicationId** : identifiant unique de votre app

Le bloc **buildTypes** est l'endroit où vous pouvez configurer plusieurs types de build. Par défaut, le système de construction définit deux types : **debug** et **release**. Le type de build de débogage n'est pas explicitement affiché dans la configuration de build par défaut, mais il inclut des outils de débogage et est signé avec la clé de débogage.

Le bloc de dépendances dans le fichier de configuration de construction au niveau du module spécifie les dépendances requises pour créer uniquement le module lui-même.

Pour en savoir plus: <https://developer.android.com/studio/build/dependencies>

Le bloc **defaultConfig** encapsule les paramètres par défaut et les entrées pour toutes les variantes de build, et peut remplacer certains attributs dans main/AndroidManifest.xml de manière dynamique à partir du système de build.



# Ressource : Values

strings.xml :

```
1 <resources>
2
3     <string name="app_name">My Application</string>
4     <string name="hello">Hello</string>
5     <string name="sure">Are you sure ?</string>
6
7 </resources>
8
```

colors.xml :

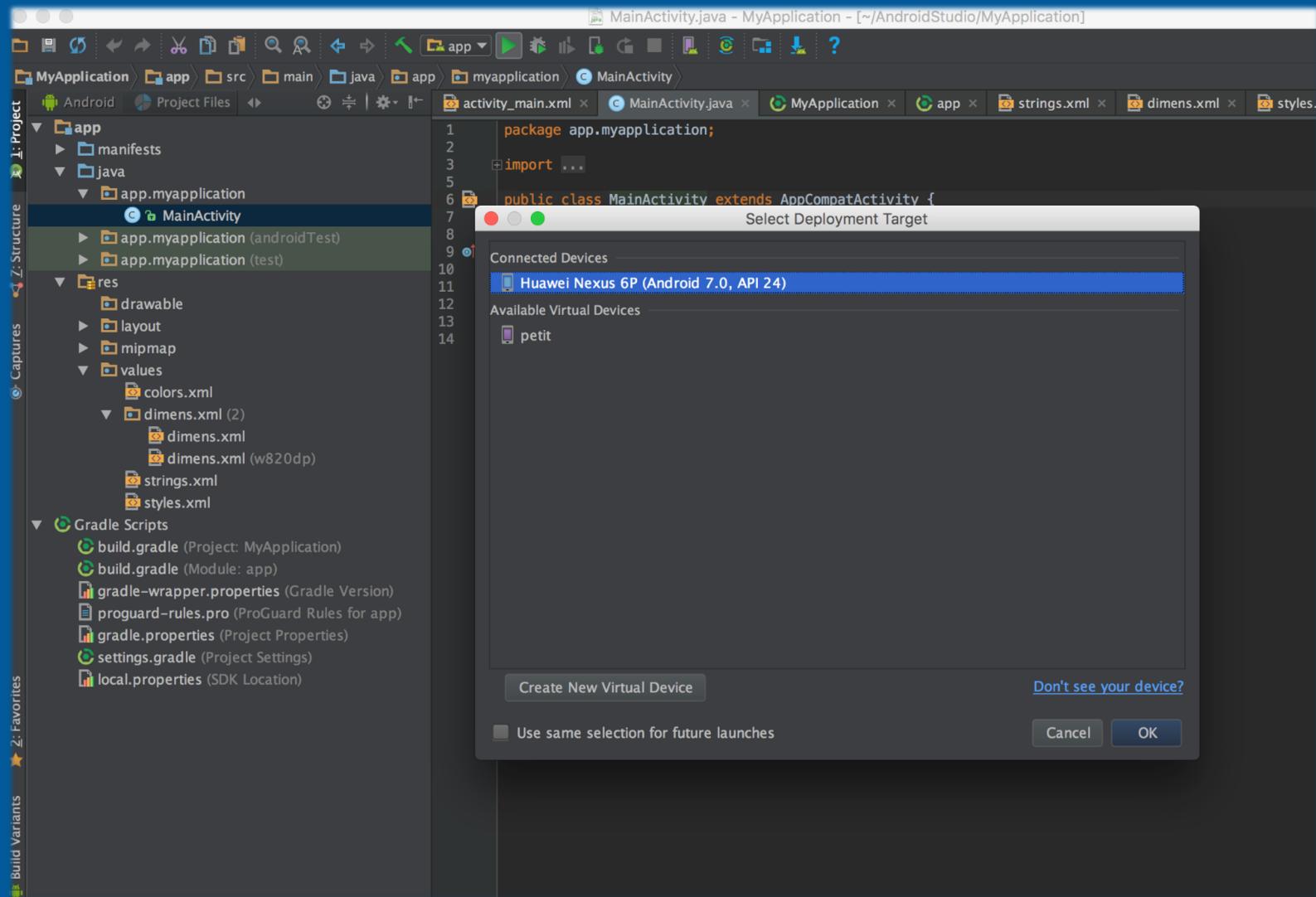
```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <color name="colorPrimary">#3F51B5</color>
4     <color name="colorPrimaryDark">#303F9F</color>
5     <color name="colorAccent">#FF4081</color>
6
7 </resources>
```

dimens.xml :

```
1 <resources>
2     <!-- Default screen margins, per the Android Design guidelines. -->
3     <dimen name="activity_horizontal_margin">16dp</dimen>
4     <dimen name="activity_vertical_margin">16dp</dimen>
5
6 </resources>
```

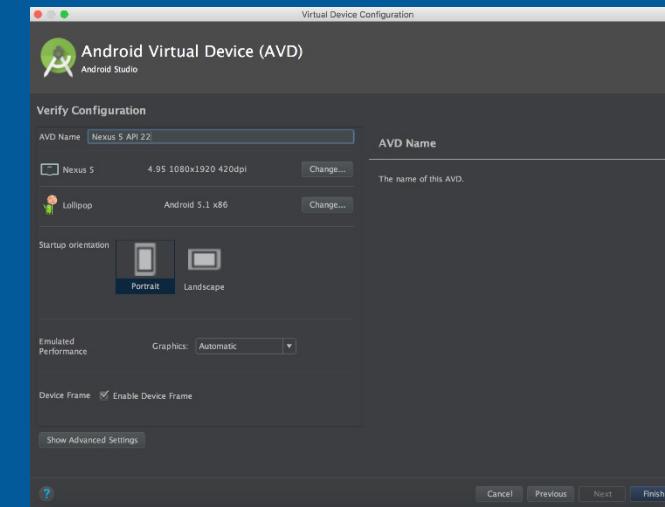
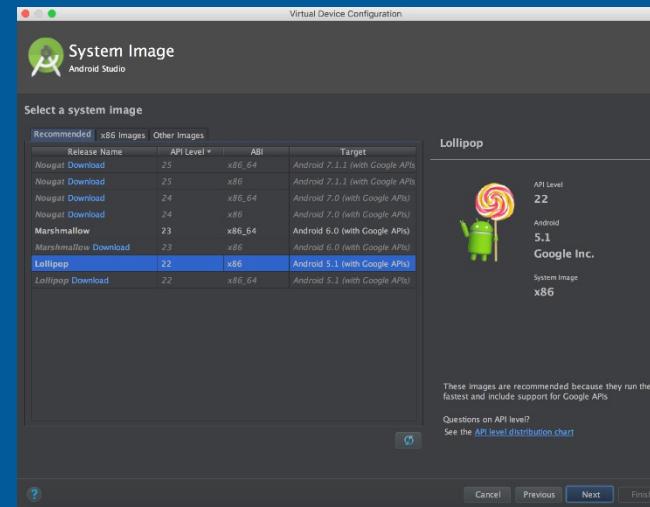
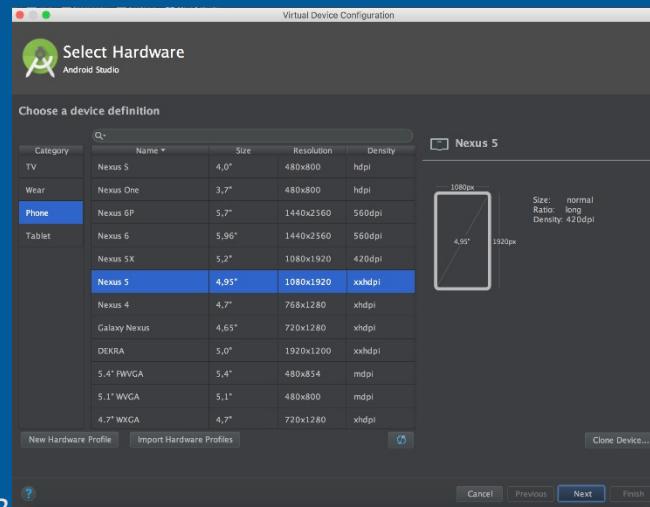
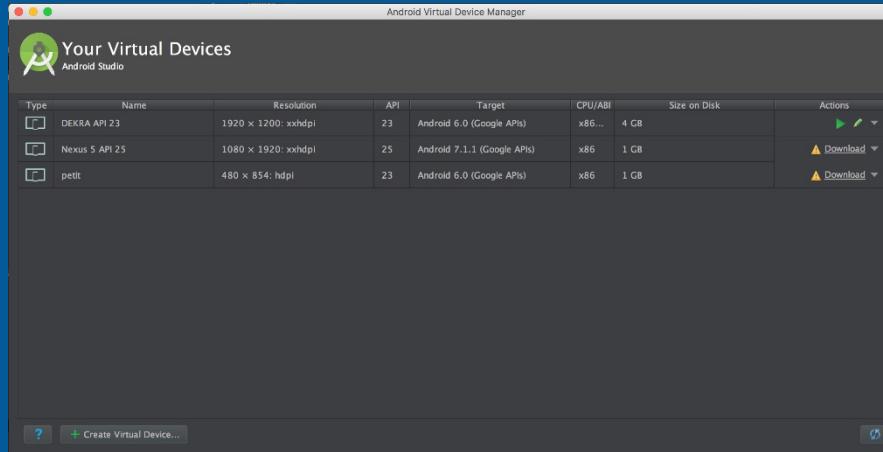


# Compiler et lancer son app



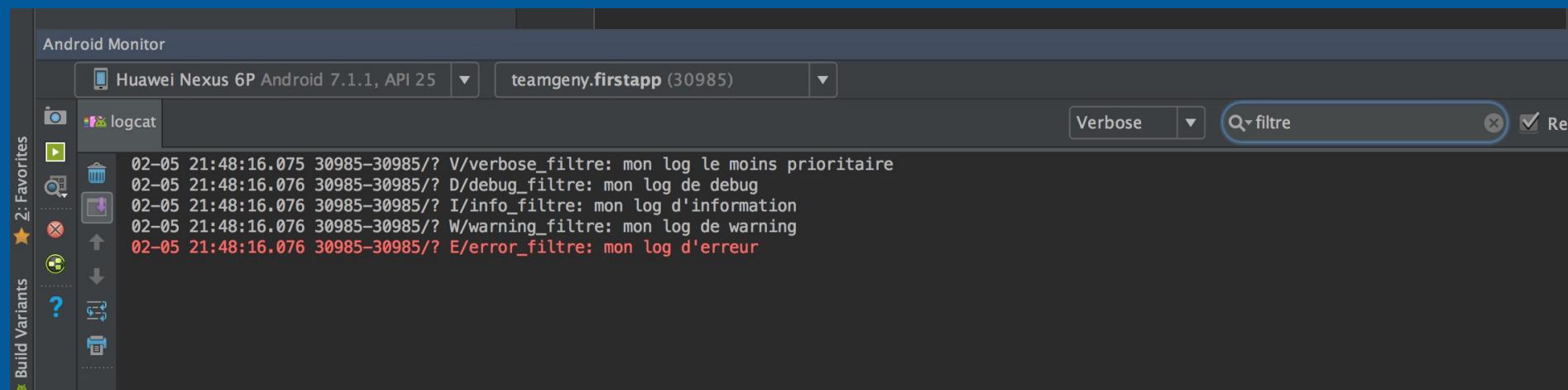
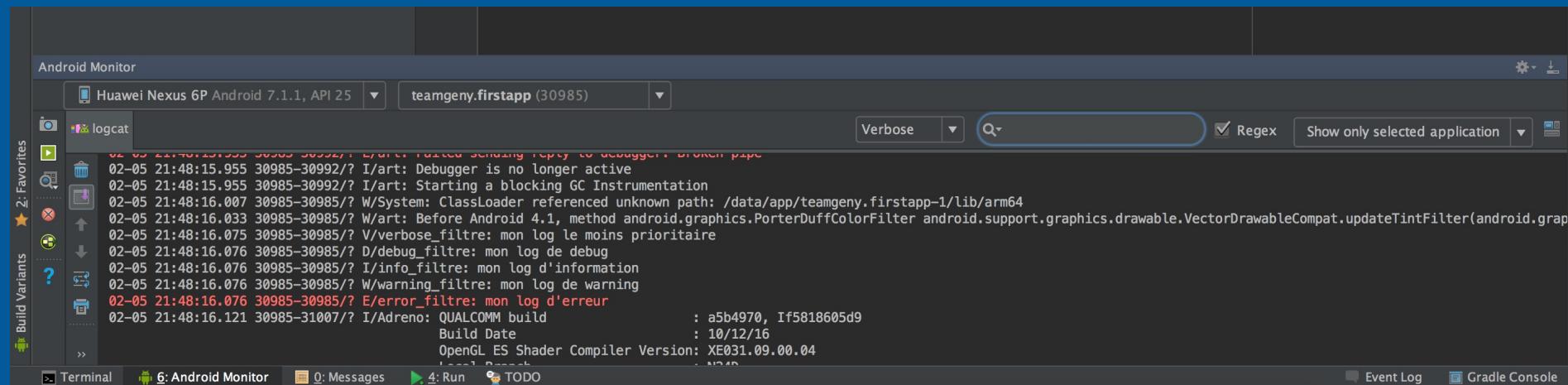
# Android Device Manager

- AVD Manager permet d'émuler des devices Android
- Il est possible de choisir des modèles existants ou de créer son propre modèle



# Logs

```
Log.v("verbose_filtre", "mon log le moins prioritaire");
Log.d("debug_filtre", "mon log de debug");
Log.i("info_filtre", "mon log d'information");
Log.w("warning_filtre", "mon log de warning");
Log.e("error_filtre", "mon log d'erreur");
```



# Debug

The screenshot shows the Android Studio interface with the following details:

- Code Editor:** The main window displays `MainActivity.java` from the `FirstApp` project. The code is as follows:

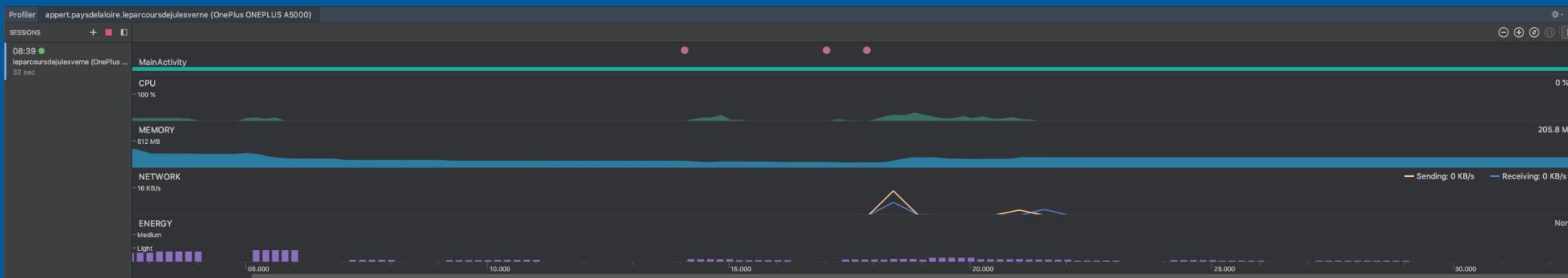
```
1 package teamgeny.firstapp;
2
3 import ...
4
5 public class MainActivity extends AppCompatActivity {
6
7     @Override
8     protected void onCreate(Bundle savedInstanceState) {
9         super.onCreate(savedInstanceState);
10        setContentView(R.layout.activity_main);
11
12        int i = 123; i: 123
13        String str = "abc"; str: "abc"
14
15        Log.d("debug", "test : " + i + str); i: 123 str: "abc"
16
17    }
18
19
20 }
```

- Debugger:** The bottom panel shows the **Debug** tab selected. The **Frames** tool window lists the current stack frames, with `onCreate:17, MainActivity (teamgeny.firstapp)` highlighted. The **Variables** tool window shows local variables: `this`, `savedInstanceState`, `i` (value 123), and `str` (value "abc").
- Bottom Bar:** The toolbar includes icons for Terminal, Android Monitor, Messages, Run, Debug, TODO, Event Log, and Gradle Console. A message at the bottom states: `Gradle build finished in 1s 397ms (moments ago)`.



# Performances avec le Profiler

L'ensemble des outils de monitoring ont été réécrits pour offrir une ergonomie améliorée et des nouvelles fonctionnalités. Notons par exemple la possibilité de lister les requêtes réseau, voir les entêtes et le résultat reçu.



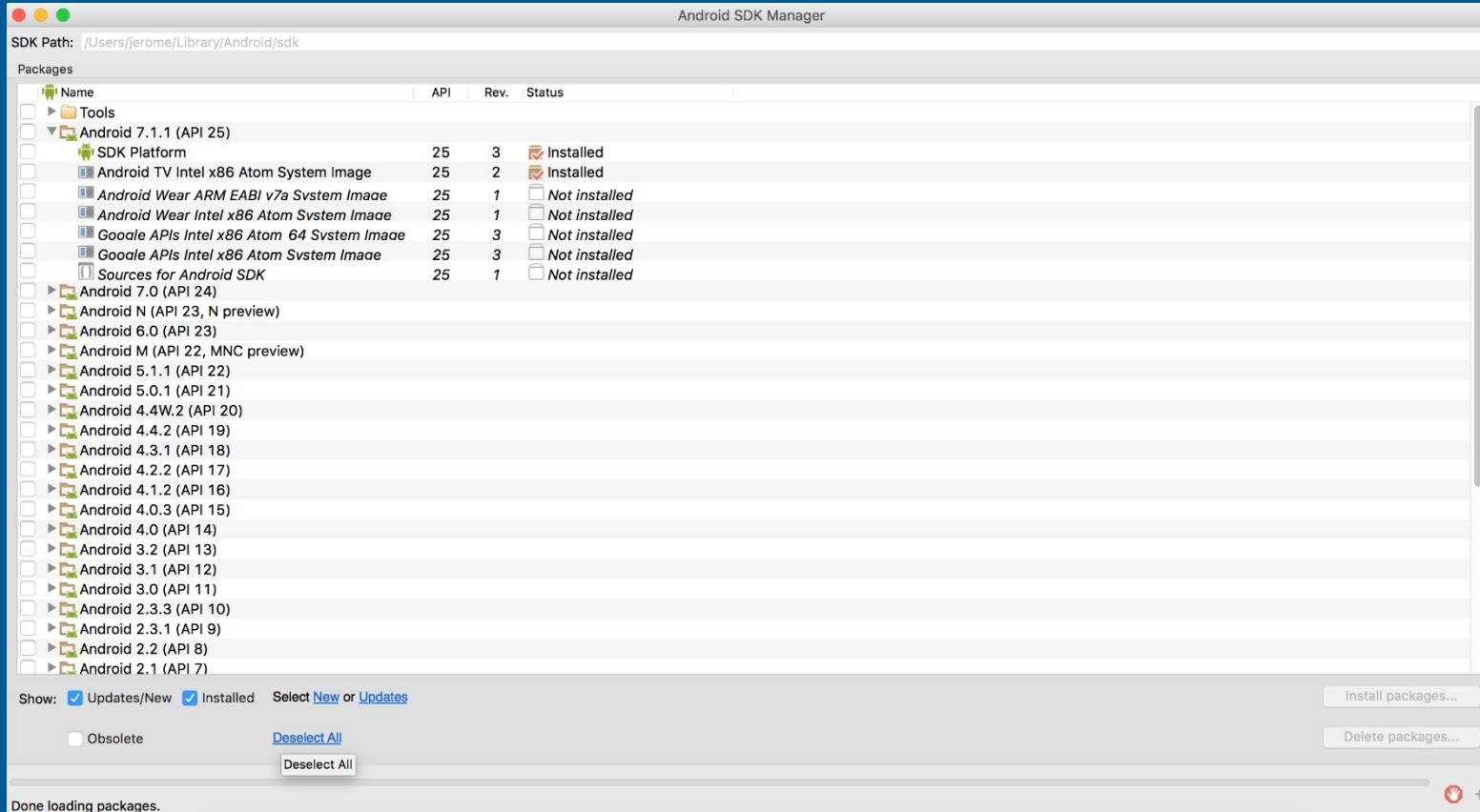
# Inspector son code

The screenshot illustrates the inspection process in Android Studio:

- Left Panel:** Shows the project structure for a "FirstApp" project. The "app" module is selected. The "teamgeny.firstrapp" package contains the "MainActivity" and "ExampleInstrumentedTest" classes.
- Middle Panel:** The "Code" menu is open, and the "Analyze" submenu is selected. The "Inspect Code..." option is highlighted.
- Right Panel:** A "Specify Inspection Scope" dialog is open. The "Inspection scope" section has "Whole project" selected. The "Custom scope" dropdown shows "Project Files". The "Include test sources" checkbox is checked. The "Inspection profile" dropdown shows "Project Default".
- Bottom Panel:** The "Inspection Results" tool window displays the results for the "Project Default" profile. It lists findings under categories like "FirstApp", "Android > Lint > Security", "Android > Lint > Usability", and "Data flow issues". One specific issue is highlighted: "Not all execution paths return a value" at line 20 of the "build.gradle" file.



# SDK Manager



# Quelques plugins pour Android Studio

- **Markdown** : visualisation graphique de vos fichiers écrits en Markdown
- **Rainbow Brackets** : des parenthèses et accolades en couleur
- **JSON To Kotlin Class** : convertir des JSON en classes Kotlin
- **Key Promoter X** : apprenez à utiliser vos raccourcis clavier
- BashSupport



# Ressources

---

Les ressources sont des fichiers qui contiennent des informations capitales pour l'application, telles que des images, des chaînes de caractères, des couleurs ou des styles.



# Les ressources, c'est quoi ?

- Les ressources sont des fichiers qui contiennent des informations qui ne sont :
- Pas en Java (ce n'est donc pas du code).
- Pas dynamique (le contenu d'une ressource restera inchangé entre le début de l'exécution de votre application et la fin de l'exécution).
- Android est destiné à être utilisé sur un très grand nombre de supports différents, et il faut par conséquent s'adapter à ces supports. L'avantage des ressources, c'est qu'elles nous permettent de nous adapter facilement à toutes ces situations différentes.



# Types de ressources

<b>animator/</b>	Fichiers XML pour les animations
<b>anim/</b>	Fichiers XML qui définissent des animations entre deux. (Les animations de propriété peuvent également être enregistrées dans ce répertoire, mais le répertoire animator/ est préférable pour les animations de propriété afin de distinguer les deux types.)
<b>color/</b>	Fichiers XML décrivant les couleurs
<b>drawable/</b>	Fichiers bitmaps (.png, .9.png, .jpg, .gif) ou XML qui font référence à des : •Bitmap files •Nine-Patches (re-sizable bitmaps) •State lists •Shapes •Animation drawables •Autres drawables
<b>mipmap/</b>	Drawables des différentes launcher icônes.
<b>layout/</b>	XMLs des vues
<b>menu/</b>	XMLs de définition des menus
<b>raw/</b>	Tout ce qui ne rentre pas dans les autres dossiers (musiques, vidéos, Jsons,...). Si vous souhaitez mettre en place une hiérarchie de fichiers, utilisez assets/ (à la racine de app/).
<b>values/</b>	XMLs de valeurs comme les strings, integers, dimens, styles, colors,...
<b>xml/</b>	XMLs ne rentrant pas dans les autres dossiers
<b>font/</b>	Police



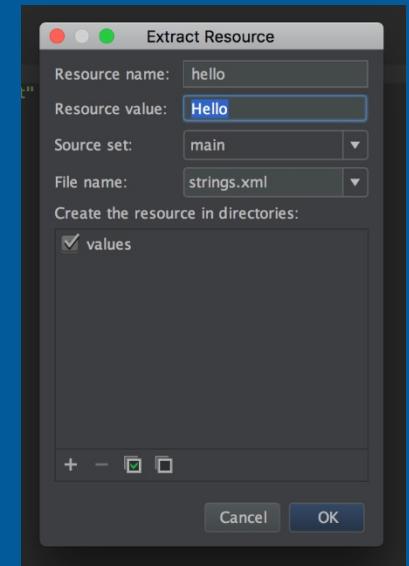
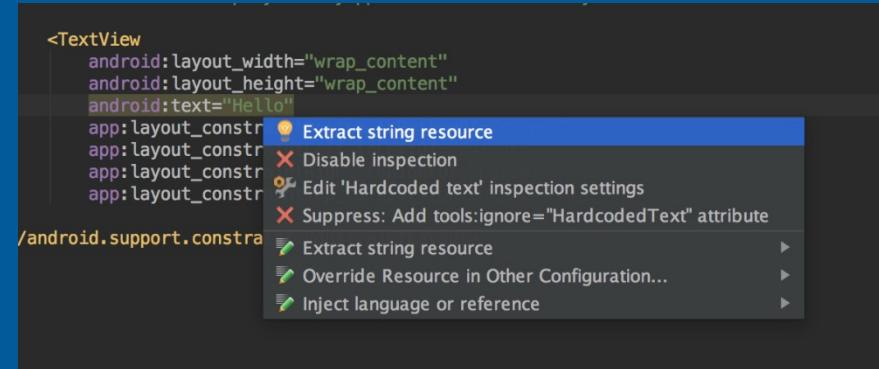
# Hiérarchie des fichiers

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            graphic.png
        layout/
            main.xml
            info.xml
        mipmap/
            icon.png
        values/
            strings.xml
```



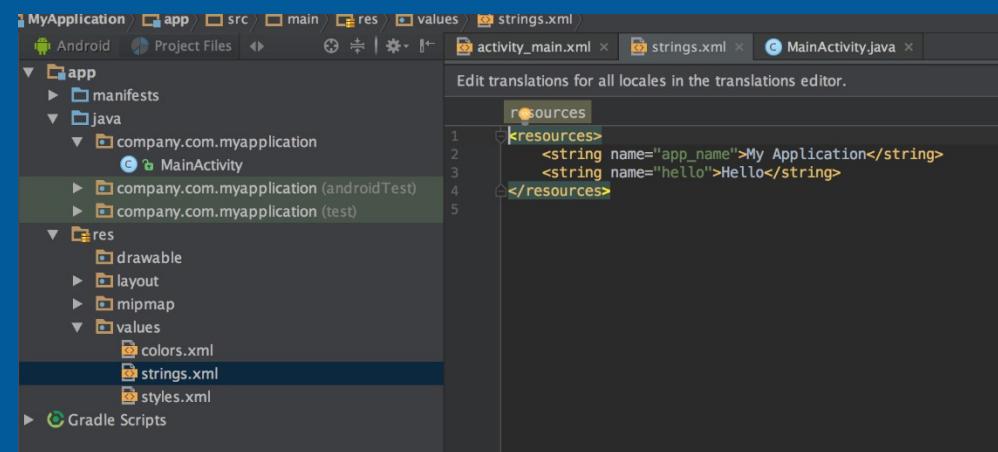
# Les chaînes de caractère

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Hello"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintLeft_toLeftOf="parent"  
    app:layout_constraintRight_toRightOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```

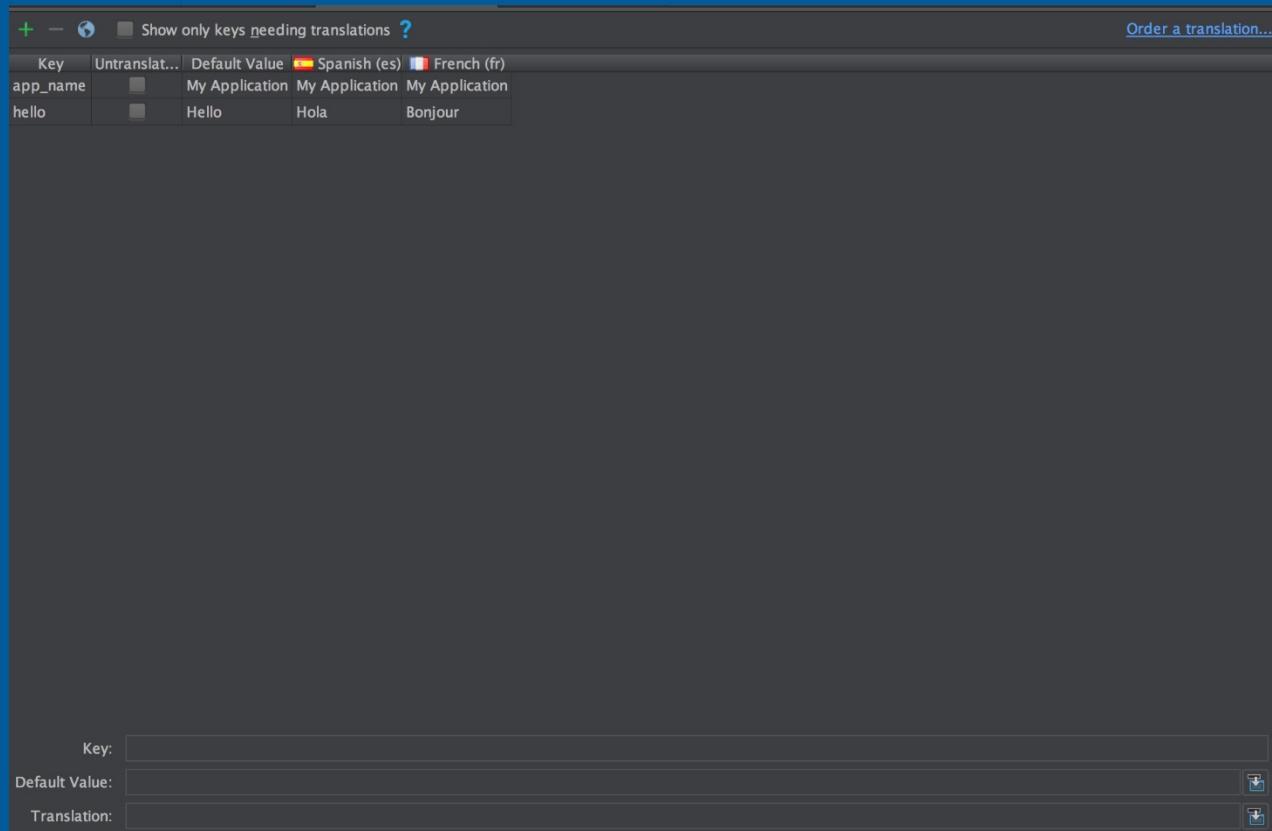


Résultat :

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/hello"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintLeft_toLeftOf="parent"  
    app:layout_constraintRight_toRightOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```

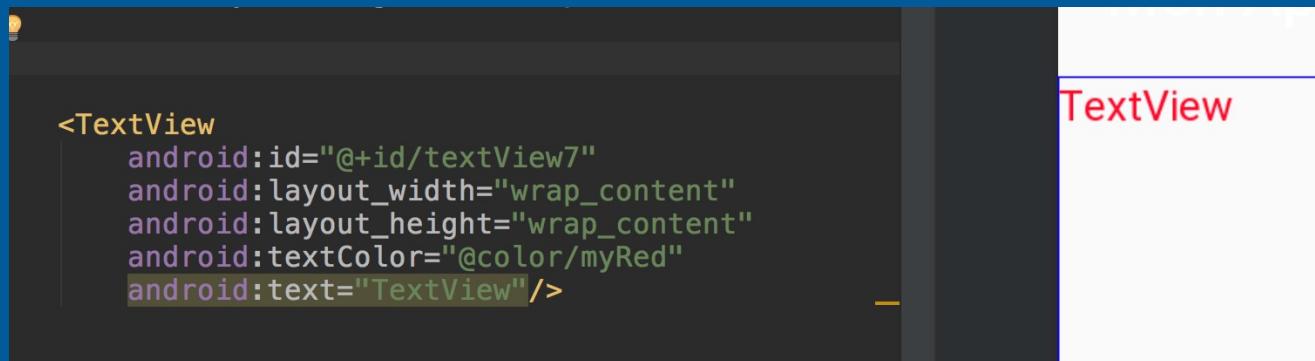
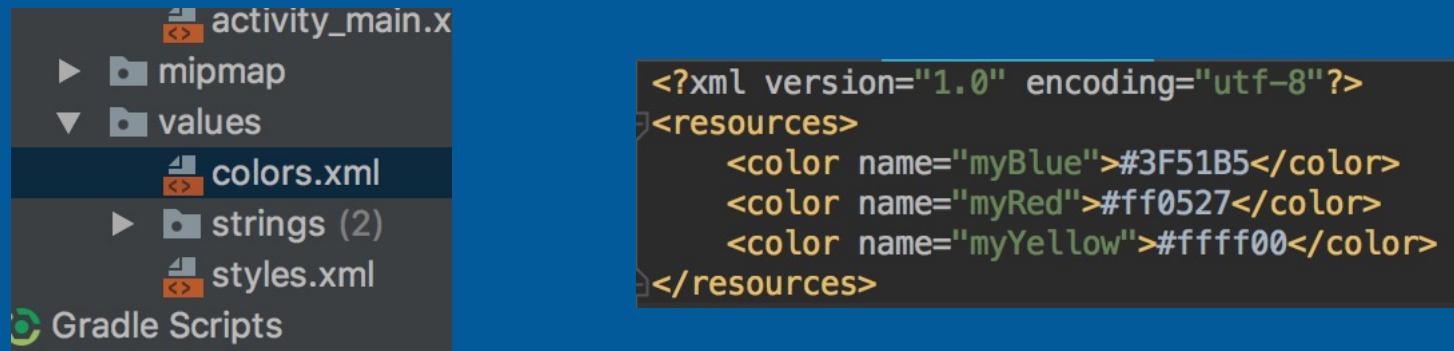


# Internationalisation des applications



Translation Editor

# Les couleurs



# Styles

---

Styliser son app



# Créer et appliquer un style

Dans style.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="GreenText" parent="TextAppearance.AppCompat">
        <item name="android:textColor">#00FF00</item>
    </style>
</resources>
```

```
<TextView
    style="@style/GreenText"
    ... />
```



# Étendre un style

```
<style name="GreenText" parent="TextAppearance.AppCompat">
    <item name="android:textColor">#00FF00</item>
</style>

<style name="GreenText.Large">
    <item name="android:textSize">22sp</item>
</style>
```

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    style="@style/GreenText.Large"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```



# Theme par defaut

Vous pouvez créer un thème de la même manière que vous créez des styles. La différence est la façon dont vous l'appliquez: au lieu d'appliquer un style avec l'attribut style sur une vue, vous appliquez un thème avec l'attribut android:theme sur la balise **<application>** ou une balise **<activity>** dans le fichier **AndroidManifest.xml**.

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>
```

```
<manifest ... >
    <application android:theme="@style/Theme.AppCompat" ... >
        </application>
    </manifest>
```



# Customiser le style des widgets

On peut définir le style d'un widget de manière globale de sorte à ne pas à avoir à le redéfinir dans chaque vues :

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <item name="buttonStyle">@style/Widget.AppCompat.Button.Borderless</item>
    ...
</style>
```

Exemple :

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
    <item name="android:textViewStyle">@style/GreenText.Large</item>
</style>

<style name="GreenText" parent="TextAppearance.AppCompat">
    <item name="android:textColor">#00FF00</item>
</style>

<style name="GreenText.Large">
    <item name="android:textSize">22sp</item>
</style>
```



# Les layouts

Un layout définit la structure d'une interface utilisateur dans votre application.



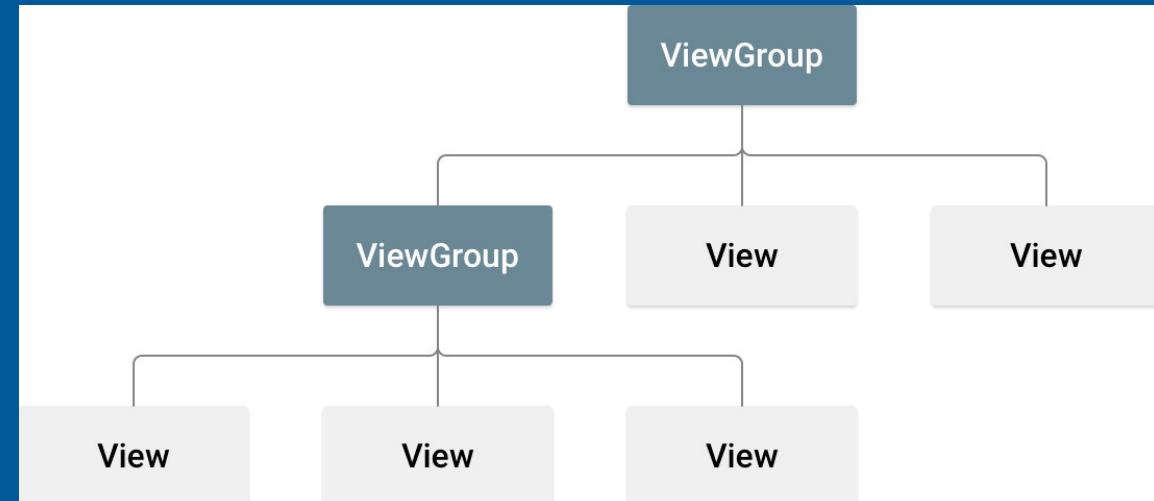
# ViewGroup / View

Tous les éléments de la mise en page sont construits à l'aide d'une hiérarchie d'objets View et ViewGroup.

Une vue dessine généralement quelque chose que l'utilisateur peut voir et avec lequel il peut interagir. Tandis qu'un ViewGroup est un conteneur invisible qui définit la structure de présentation de View et d'autres objets ViewGroup.

Les objets View sont généralement appelés "widgets" et peuvent être l'une des nombreuses sous-classes, telles que Button ou TextView.

Les objets ViewGroup, généralement appelés "layout", peuvent être l'un des nombreux types offrant une structure de présentation différente, tels que LinearLayout ou ConstraintLayout.



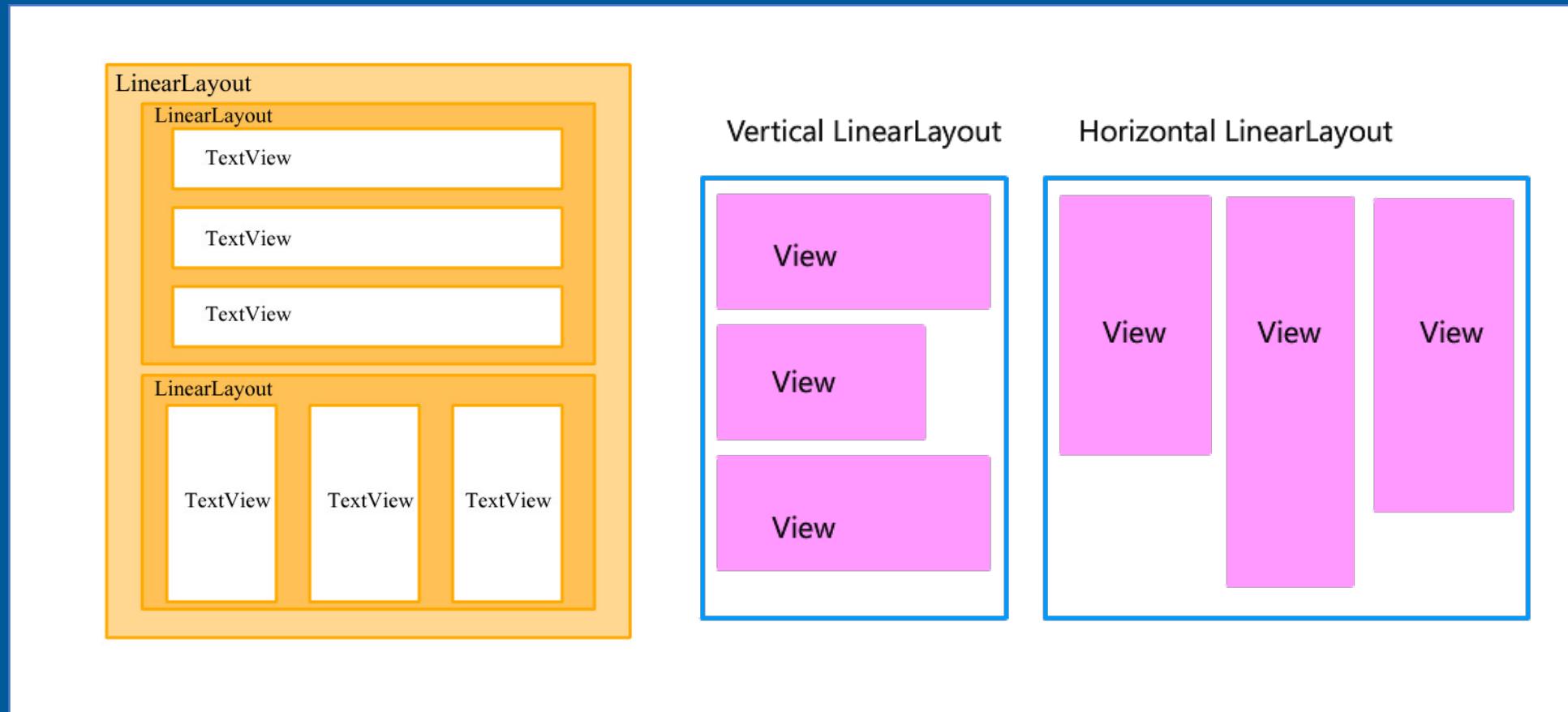
# Exemples de layouts (ViewGroup)

- **LinearLayout**
  - Layout avec orientation unique (horizontal et vertical)
- **RelativeLayout**
  - Layout avec positionnement des éléments
- **GridLayout**
  - Position des éléments dans un tableau
- **FrameLayout**
  - Elément unique
- **CoordinatorLayout**
  - Layout avec dépendance entre les éléments
- **ConstraintLayout**
  - Équivalent au RelativeLayout mais plus flexible et plus simple
- **TextInputLayout**
  - Ne sert que pour les champs d'entrée de texte

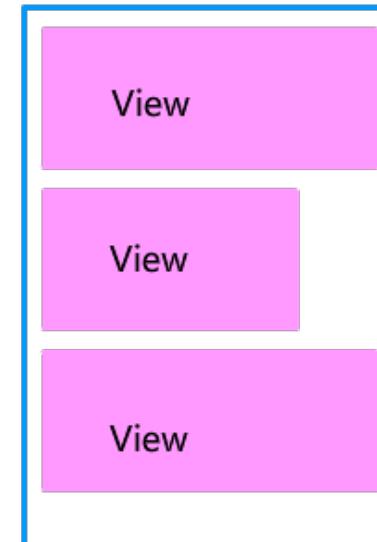


# LinearLayout

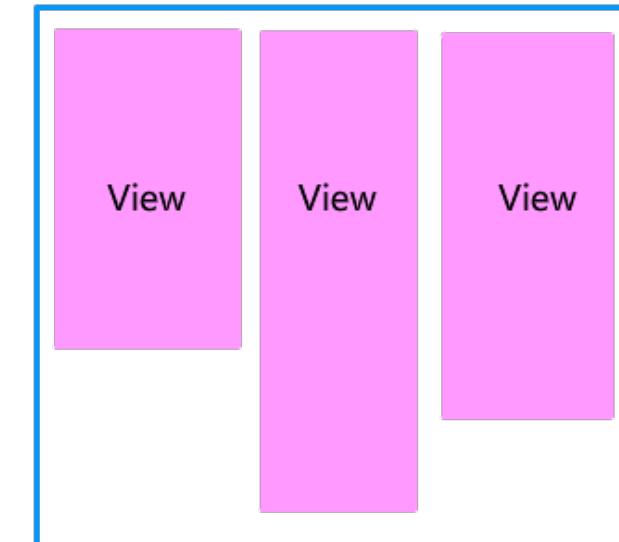
Les widgets sont placés d'une manière linéaire, soit verticalement, soit horizontalement  
(soit en lignes, soit en colonnes)



Vertical LinearLayout

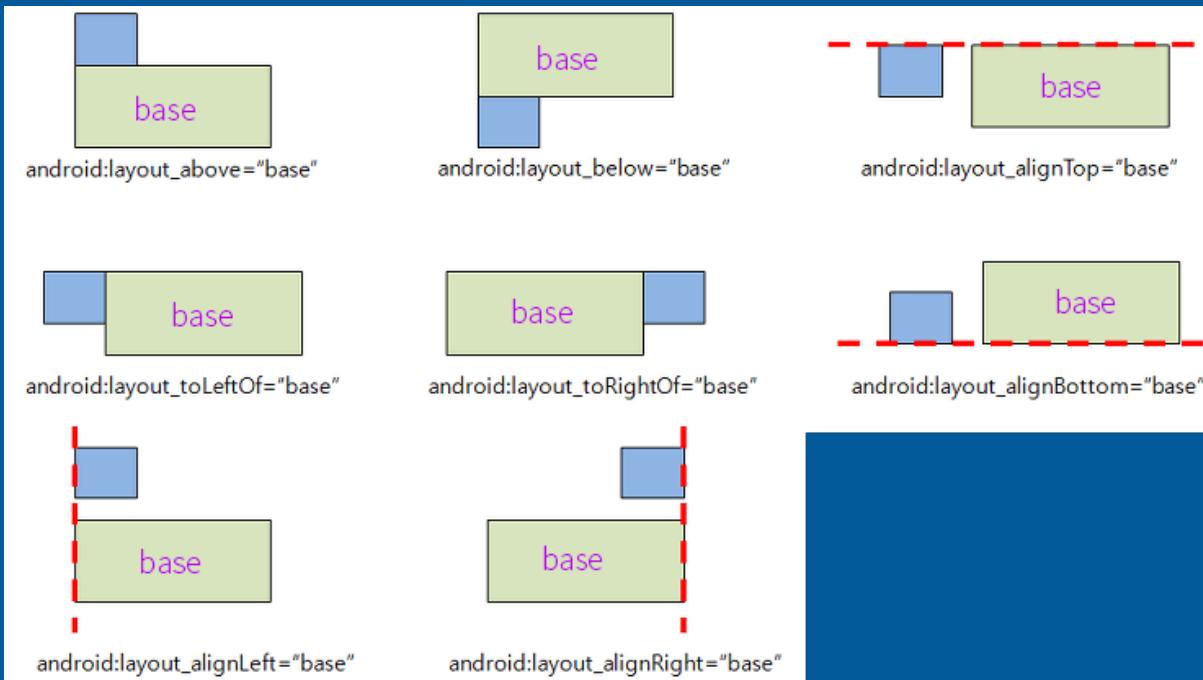
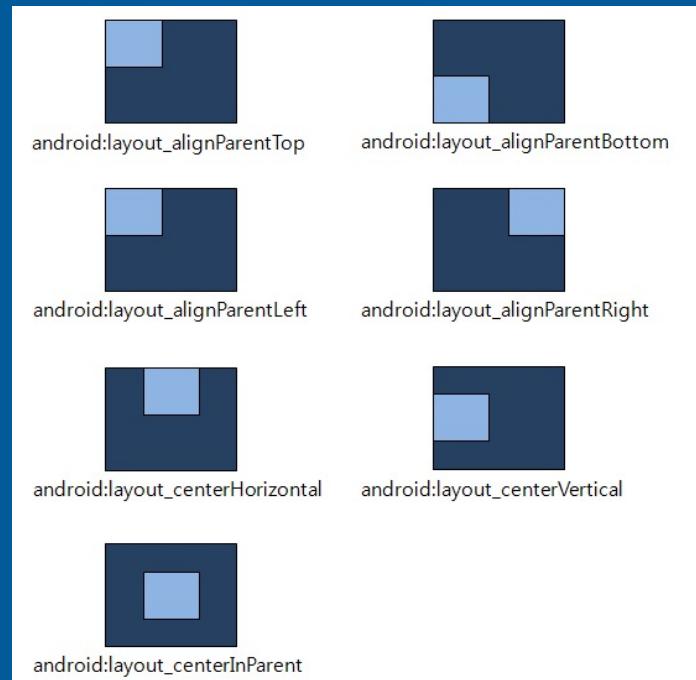
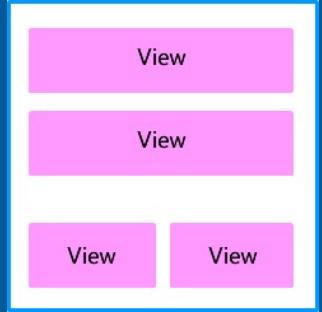


Horizontal LinearLayout



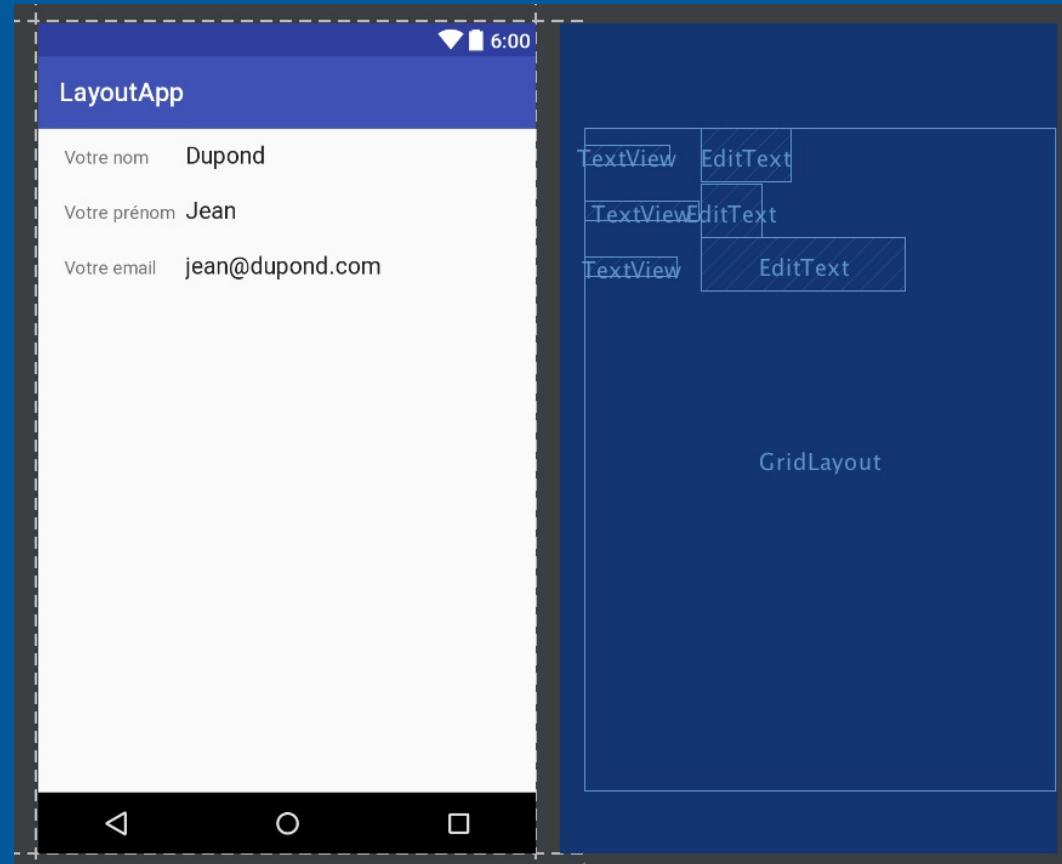
# RelativeLayout

Permet de placer les éléments d'une manière relative. C'est-à-dire que les éléments peuvent être positionnés les uns par rapport aux autres.



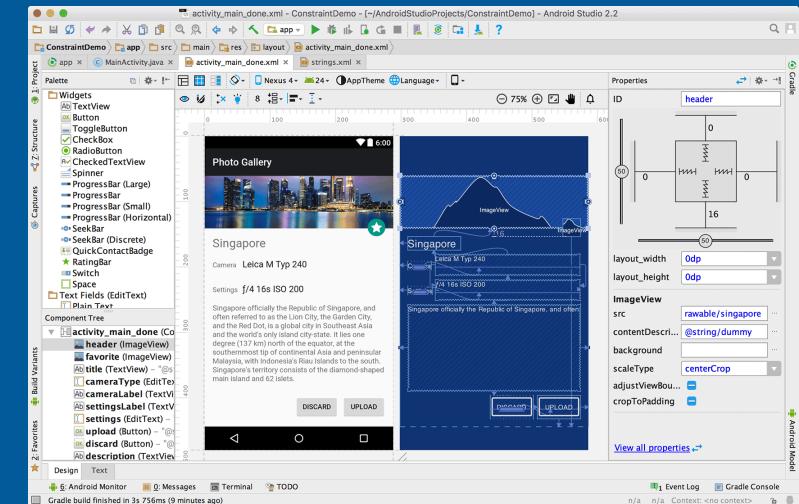
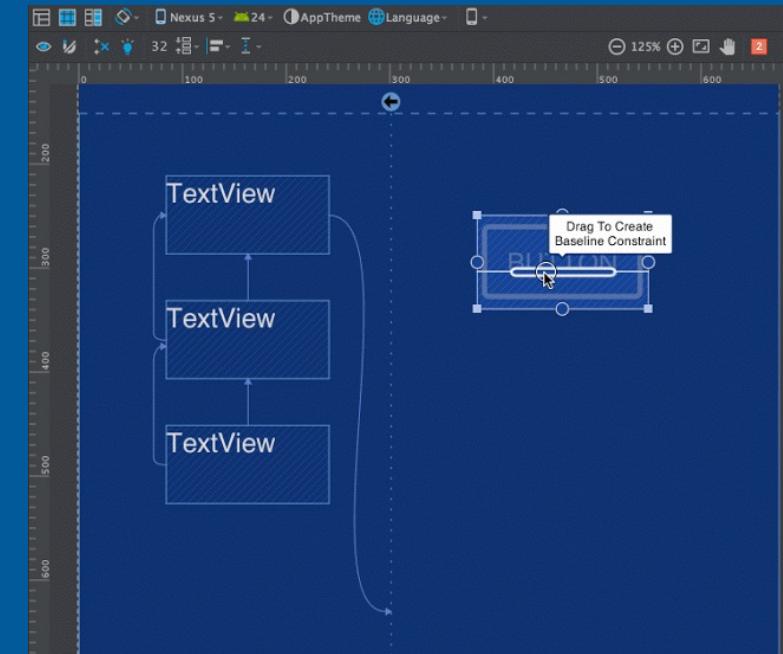
# GridLayout

```
GridLayout
1  <?xml version="1.0" encoding="utf-8"?>
2  <GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:columnCount="2"
6      android:layout_marginLeft="20dp"
7      android:orientation="horizontal" >
8
9      <TextView
10         android:text="Votre nom" />
11
12      <EditText
13         android:text="Dupond" />
14
15      <TextView
16         android:text="Votre prénom " />
17
18      <EditText
19         android:text="Jean" />
20
21      <TextView
22         android:text="Votre email" />
23
24      <EditText
25         android:text="jean@dupond.com" />
26
27  </GridLayout>
```



# ConstraintLayout

- Le ConstraintLayout est un nouveau type de ViewGroup présenté en **mai 2016** par Google à la Google I/O.
- Son objectif est de vous permettre de **créer vos layouts avec plus de facilité**, ainsi que de vous fournir des outils pour placer au maximum vos widgets sur une hiérarchie plate.
- Dans les grandes lignes, le ConstraintLayout reprend le fonctionnement du RelativeLayout. Il permet de placer ses widgets de façon relative aux autres. Ici, cela se fait à l'aide de "**contraintes**" de positionnement, similaires aux règles du RelativeLayout (*layout\_alignTop*, *layout\_toRightOf* etc.).
- Sauf que ça ne s'arrête pas là. Car en effet, il existe d'autres contraintes que celles héritées du RelativeLayout. Certaines sont là pour calculer un ratio, placer un élément sur un pourcentage précis de l'écran ou même pouvoir créer des chaînes de widgets qui permettront, par exemple, d'utiliser les poids comme on le ferait avec un LinearLayout. En cela, le ConstraintLayout permet de répondre à plusieurs problématiques qui nécessitaient jusqu'ici l'usage de ViewGroup imbriqués.



# Les dimensions

- **dp** : Density independent Pixel ( Densité de pixels indépendant) - Unité abstraite qui est basés sur la densité physique de l'écran.
- **dpi** = 1/160 dp
- **sp** : Scale independent Pixel (Echelle de pixels indépendant) - Utilisé pour les tailles de polices. On pourrait comparer cette unité aux em du développement web. La police peut être plus ou moins grosse suivant les préférences utilisateurs
- **pt** : Point - 72 points par pouces. basé sur la taille physique de l'écran.
- **px** : Pixels - Corresponds aux pixels réels de l'écran. Cette unité de mesure n'est pas recommandées car le rendu sur les différents types d'écran peut être différents. Le nombre de pixels par pouce peut varier suivant les appareils.
- **mm** : Millimètre - basée sur la taille physique de l'écran
- **in** : Inches (Pouces) - basée sur la taille physique de l'écran



# Exemples de widgets

- TextView
- EditText
- Button
- RadioButton
- CheckBox
- AutoCompleteTextView
- ImageView
- WebView
- ProgressBar
- ListView / RecyclerView
- *et beaucoup d'autres...*

# TextView

The screenshot shows the Android Studio interface with the XML code for a layout and its corresponding preview.

**XML Code:**

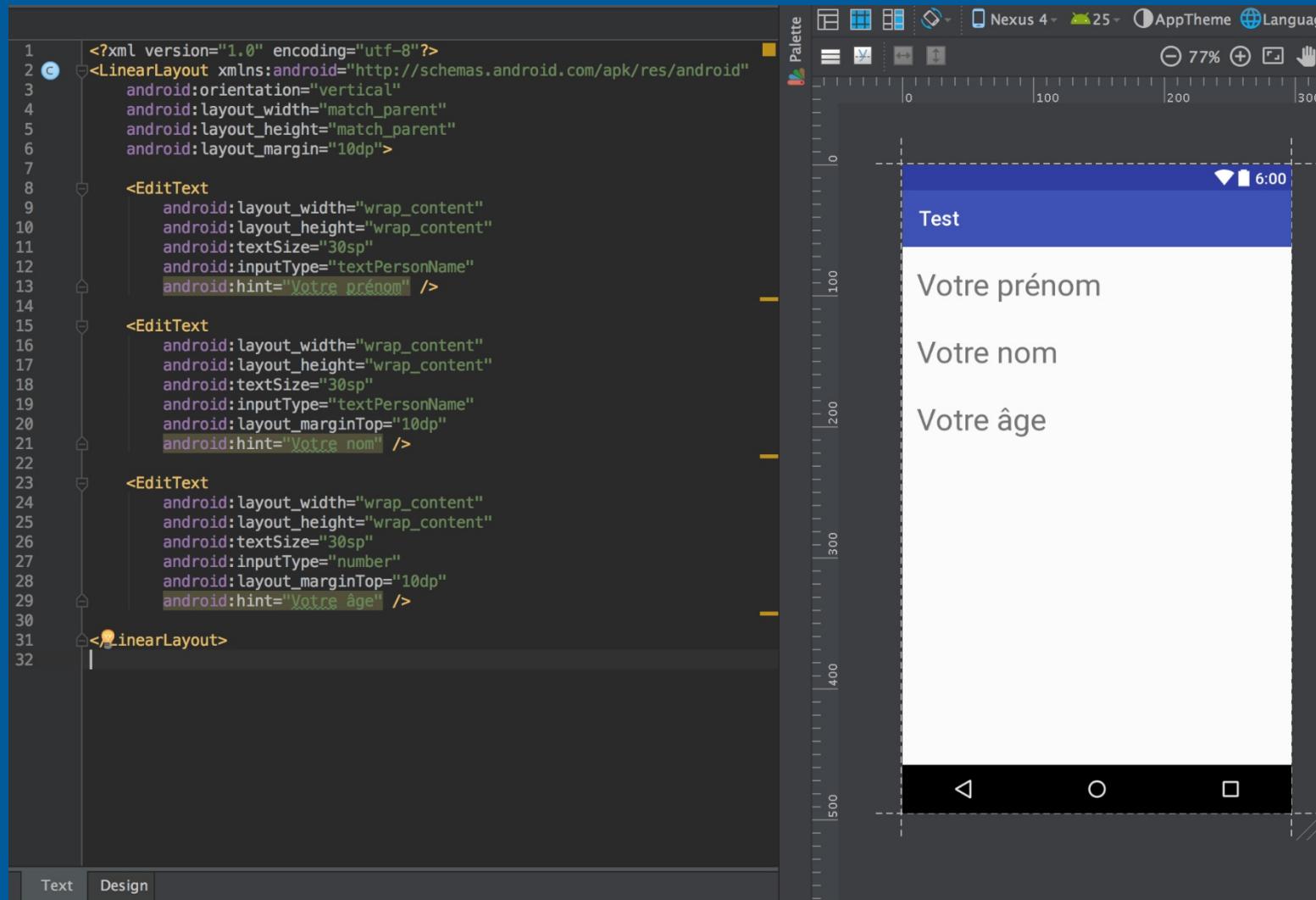
```
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="vertical"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     android:background="#60ffbf">
7
8     <TextView
9         android:layout_width="wrap_content"
10        android:layout_height="wrap_content"
11        android:textSize="30sp"
12        android:layout_marginLeft="70dp"
13        android:layout_marginBottom="40dp"
14        android:text="BONJOUR" />
15
16     <TextView
17         android:layout_width="wrap_content"
18         android:layout_height="wrap_content"
19         android:text="LE MONDE" />
20
21     <LinearLayout
22         android:layout_width="match_parent"
23         android:layout_height="wrap_content"
24         android:orientation="horizontal"
25         android:background="#a5b8ff"
26         android:layout_marginTop="30dp">
27
28         <TextView
29             android:layout_width="wrap_content"
30             android:layout_height="wrap_content"
31             android:textSize="20sp"
32             android:text="HELLO" />
33
34         <TextView
35             android:layout_width="wrap_content"
36             android:layout_height="wrap_content"
37             android:textSize="40sp"
38             android:text="THE WORLD"
39             android:layout_marginLeft="50dp" />
40
41     </LinearLayout>
42
43 </LinearLayout>
```

**Preview:**

The preview shows a vertical stack of views. At the top is a blue header bar with the text "Test". Below it is a green view containing the text "BONJOUR". A purple horizontal bar follows, displaying "LE MONDE" on the left and "THE WORLD" on the right. At the bottom is a black navigation bar with three icons. The entire layout has a light blue background.



# EditText



The screenshot shows the Android Studio interface with the XML layout editor on the left and the design preview on the right.

**XML Layout Editor (Text Tab):**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="10dp">

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        android:inputType="textPersonName"
        android:hint="Votre prénom" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        android:inputType="textPersonName"
        android:layout_marginTop="10dp"
        android:hint="Votre nom" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        android:inputType="number"
        android:layout_marginTop="10dp"
        android:hint="Votre âge" />

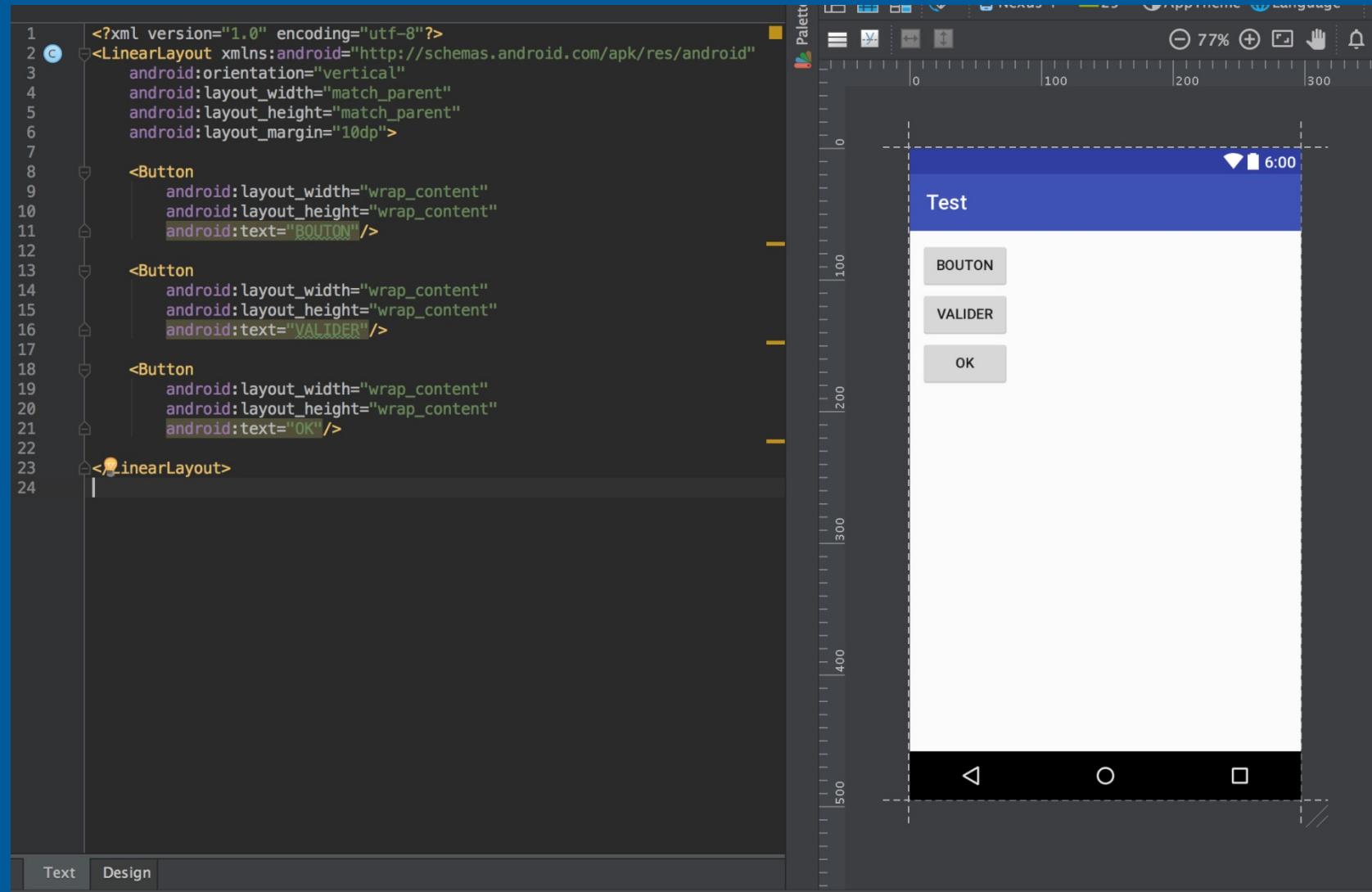
</LinearLayout>
```

**Design Preview:**

The design preview shows a mobile application interface with a blue header bar containing the text "Test". Below the header, there are three text input fields stacked vertically. The first field has the placeholder "Votre prénom", the second has "Votre nom", and the third has "Votre âge". The layout uses a vertical LinearLayout with a 10dp margin between the children.



# Button



The screenshot shows the Android Studio interface with the XML code for a layout and its corresponding preview.

**XML Code:**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="10dp">

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

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

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

</LinearLayout>
```

**Preview:**

The preview window shows a vertical layout with three buttons. The top button is labeled "BOUTON", the middle one "VALIDER", and the bottom one "OK". The layout has a blue header bar with the text "Test". The bottom navigation bar shows icons for back, home, and recent apps.

# Exemples d'attributs de View

- Id
- Width / Height
- WeightSum et Weight
- Margin / Padding
- Gravity
- Visibility
- Src
- Text
- Color
- etc...

# Identifiant d'une View

- Définir un **id** permet d'utiliser un élément depuis un Layout ou depuis une classe
- Un **id** se définit de cette manière : **@+id/nom\_id**

Exemple avec la création d'une TextView qui a pour id myTextView :

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/myTextView"  
    android:text="Hello" />
```

On peut ensuite utiliser cette TextView dans notre classe en Kotlin pour changer son texte par exemple :

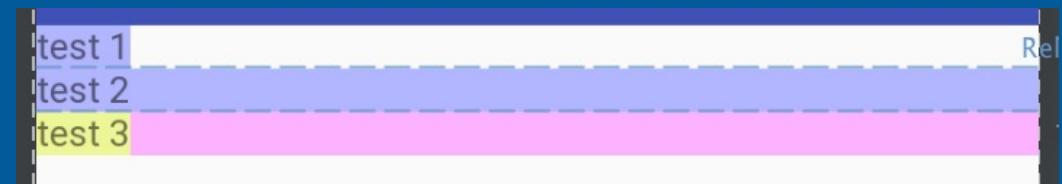
```
myTextView.text = "Hello"
```



# Width / Height

- Les attributs **width** et **height** définissent la longueur et largeur d'un élément
- Il y a plusieurs possibilités :
  - **match\_parent** : prend toute la taille du layout parent
  - **wrap\_content** : prend toute la taille du contenu de l'élément
  - taille fixe (en dp / px / in / mm / pt)

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:orientation="vertical">
6
7     <RelativeLayout
8         android:layout_width="wrap_content"
9         android:layout_height="wrap_content"
10        android:background="#b2b6ff">
11
12         <TextView
13             android:layout_width="wrap_content"
14             android:layout_height="wrap_content"
15             android:text="test 1"/>
16
17     </RelativeLayout>
18
19     <TextView
20         android:layout_width="match_parent"
21         android:layout_height="wrap_content"
22         android:background="#b2b6ff"
23         android:text="test 2"/>
24
25     <RelativeLayout
26         android:layout_width="match_parent"
27         android:layout_height="wrap_content"
28         android:background="#fcb2ff">
29
30         <TextView
31             android:layout_width="wrap_content"
32             android:layout_height="wrap_content"
33             android:background="#eef798"
34             android:text="test 3"/>
35
36     </RelativeLayout>
37
38 </LinearLayout>
```



# Margin / Padding

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:orientation="vertical"
6      android:background="#9c9c9c">
7
8      <TextView
9          android:layout_width="wrap_content"
10         android:layout_height="wrap_content"
11         android:layout_margin="30dp"
12         android:background="#ff5656"
13         android:text="test 1"/>
14
15     <TextView
16         android:layout_width="wrap_content"
17         android:layout_height="wrap_content"
18         android:padding="30dp"
19         android:background="#ff5656"
20         android:text="test 1"/>
21
22     </LinearLayout>
23
24
25
26
27
28
```



# Include

- Les **include** permettent d'insérer le contenu d'un **Layout** dans un autre **Layout**.
- Il est possible de donner un id à son **include**
- Il est aussi possible de définir la longueur et la largeur d'un **include**

```
<include layout="@layout/titlebar"/>

<include android:id="@+id/new_title"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    layout="@layout/titlebar"/>
```



# Présentation de iPlant



Vous allez commencer la création d'une app de A à Z : iPlant.

Cette application communiquera avec des webservices de l'API Trefle ( plus d'infos sur [trefle.io](https://trefle.io) ).

A la fin de la formation, votre app devrait permettre de :

- Chercher une plante, et afficher les résultats dans une liste
- Afficher les détails d'une plante
- Mettre en favoris une plante
- Permettre à l'utilisateur de changer quelques paramètres de votre application

Vous pouvez dès à présent faire un Fork du projet de base : [github.com/FormationAndroid/iPlant-base](https://github.com/FormationAndroid/iPlant-base)

Celui-ci contient les différents fragments dont vous aurez besoin ainsi que la navigation qui y est partiellement implémentée. Ce projet utilise aussi une architecture de type MVVM avec du databinding.



# TP – iPlant (1/10)



## Création du Layout des paramètres

Notre application proposera quelques options dans les paramètres :

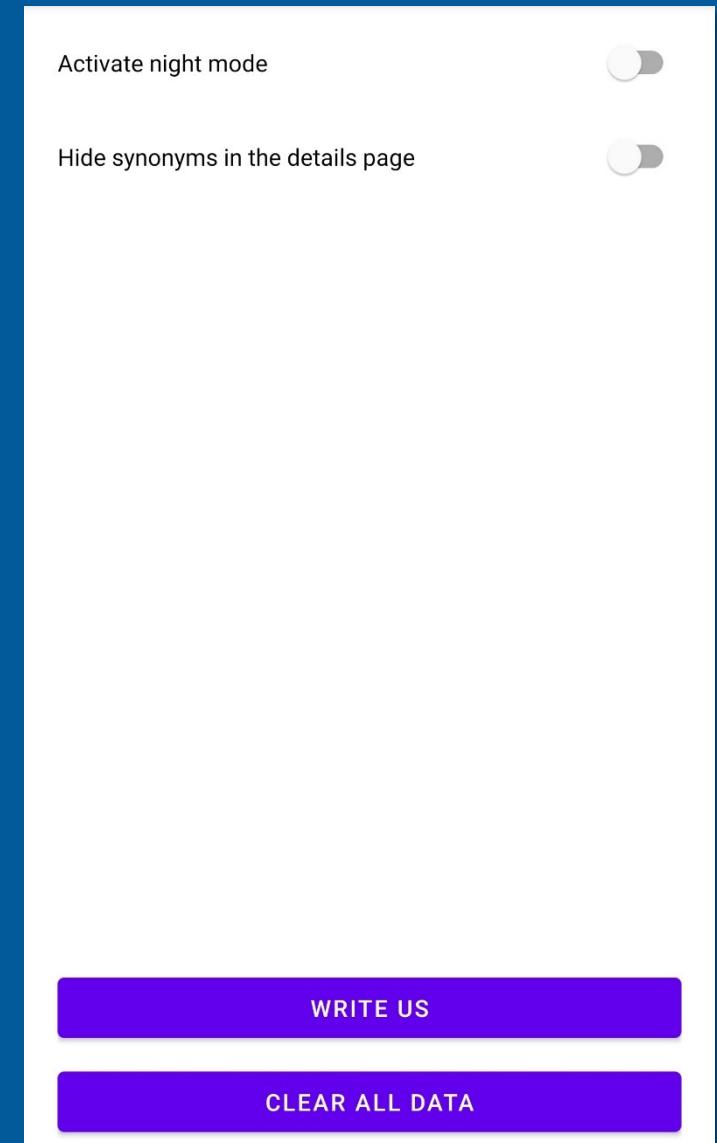
- Activer un mode nuit
- Cacher les synonymes (parfois trop nombreux) des plantes lorsque l'on consulte une fiche détaillée
- Envoyer un mail à l'équipe de développement
- Effacer toutes les données de l'application

Pour le moment, nous allons nous focaliser sur la partie graphique. Nous reviendrons plus tard sur l'implémentation des fonctionnalités.

Reproduire à l'identique le layout de droite dans le fichier **fragment\_settings.xml** (dans le dossier **res/layout**)

Pour info, ce layout est composé de ces views :

- SwitchMaterial
- Button



# CONSTRAINT LAYOUT

---

Un ConstraintLayout est un ViewGroup qui vous permet de positionner et de dimensionner les widgets de manière flexible.



# Avantages du ConstraintLayout

- Responsive UI
- Performances
- Animations
- Pas besoin du XML pour placer ses vues

Une seule règle :

- Il faut minimum une contrainte horizontale et une contrainte verticale

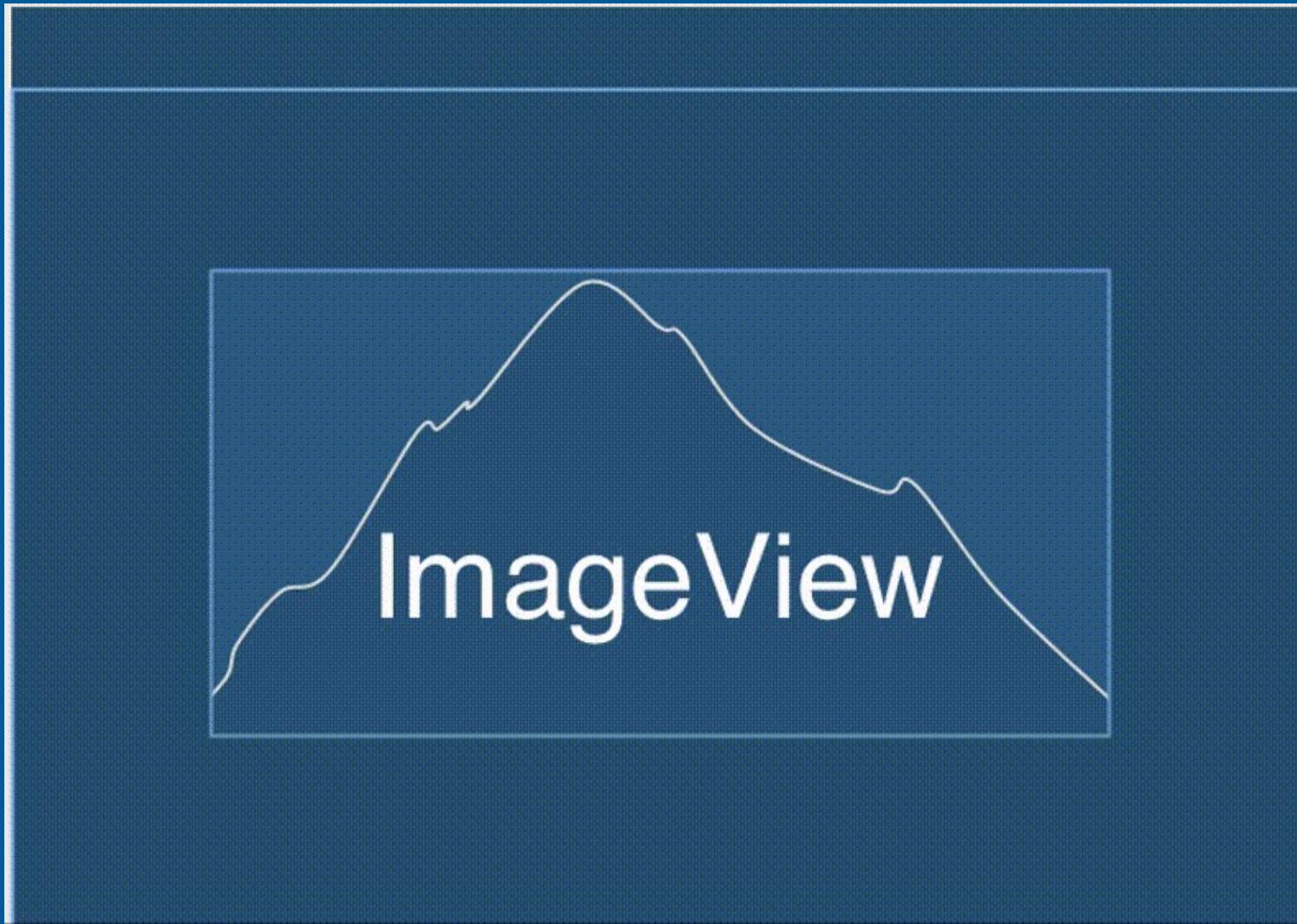


# Bases

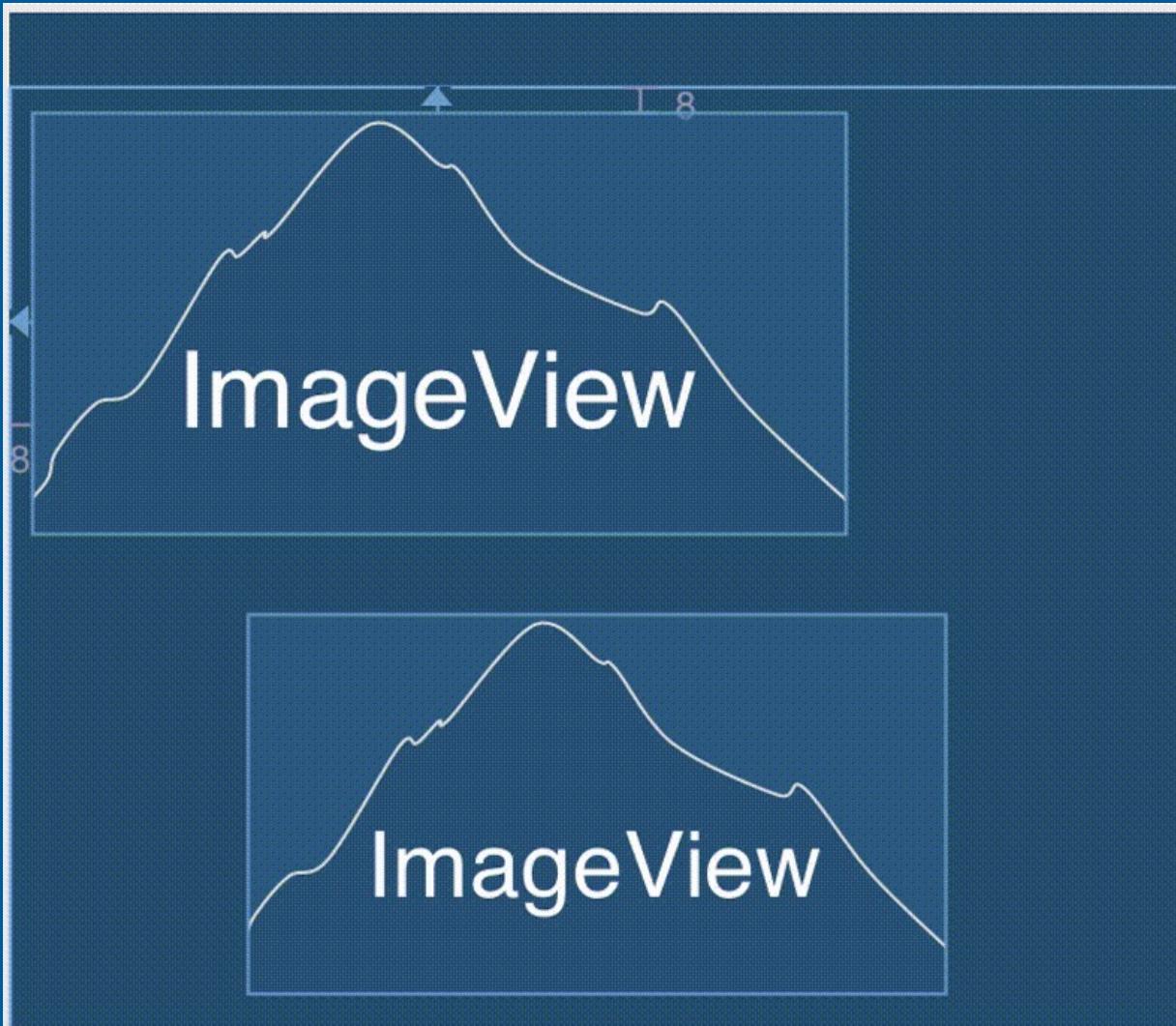
- Placement
- Contrainte
- Centrer
- Baseline



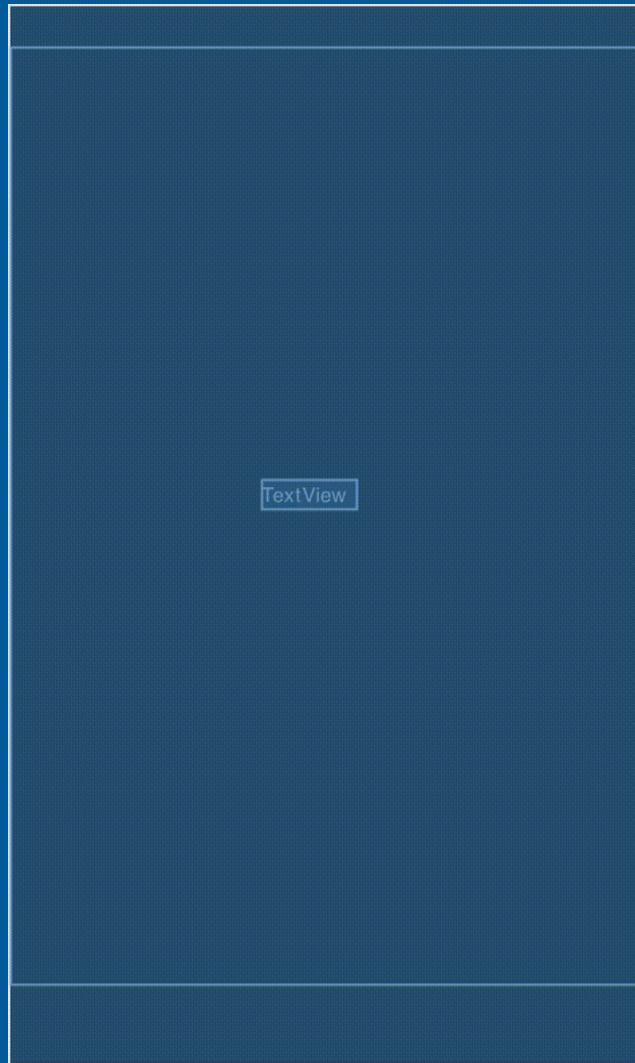
# Placement d'une vue



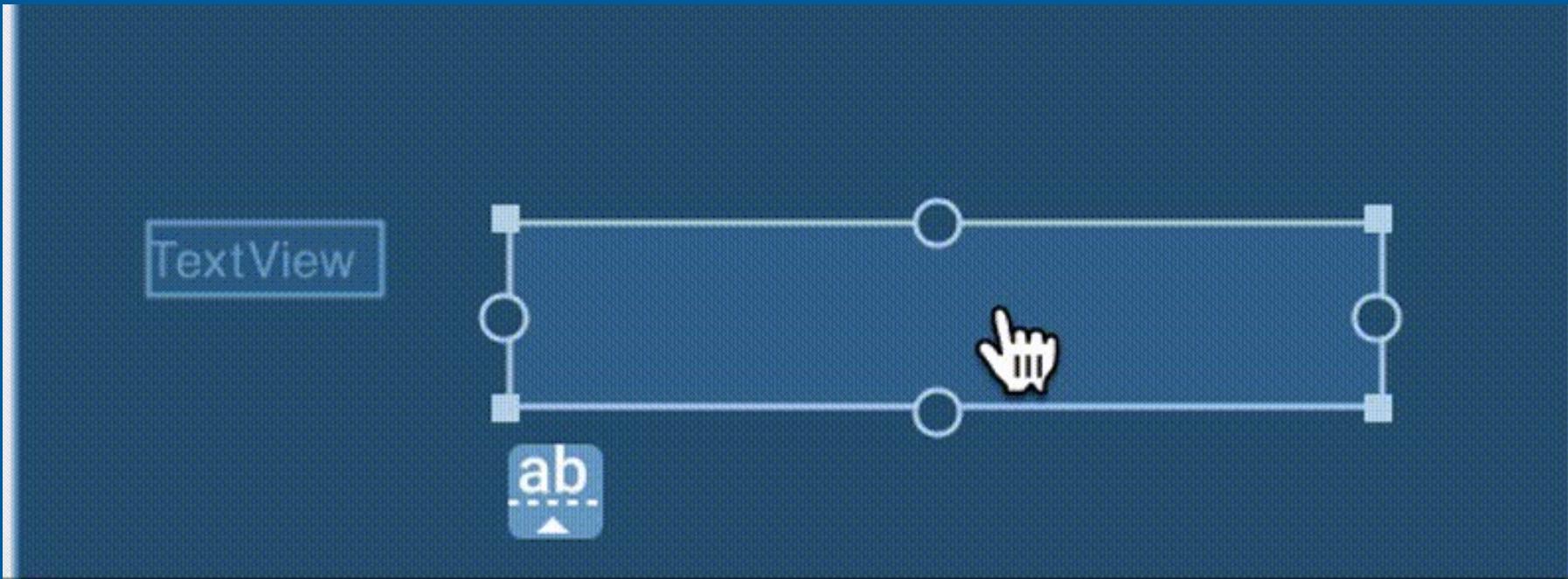
# Contrainte sur une vue



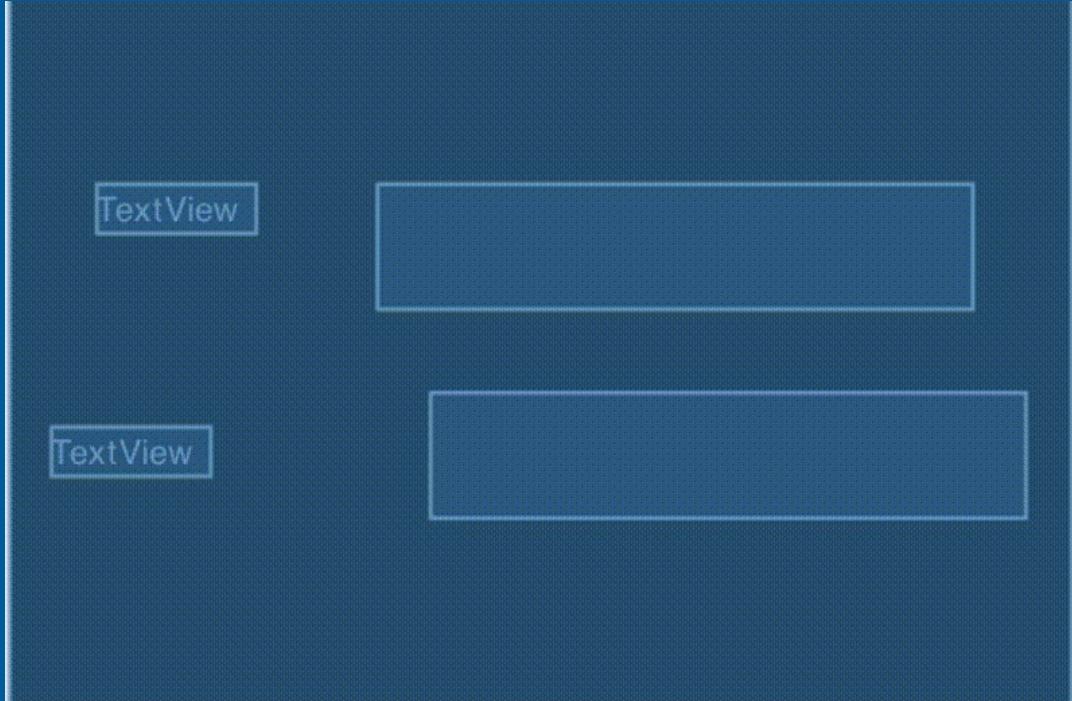
# Centrer une vue



# Aligner sur la baseline



# Exemple



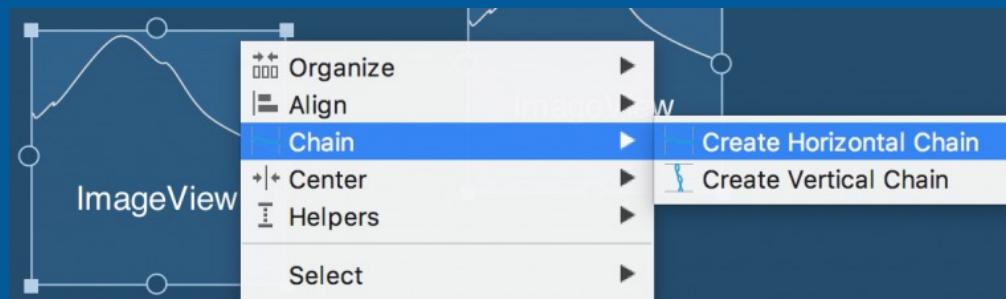
# Chaîne

- A la manière d'un `LinearLayout` avec l'attribut `gravity='center'`, les chaînes vont lier les vues et les répartir sur un axe donné.
- Derrière ce mécanisme se cache une notion de contrainte réciproque
- Il existe 4 *Cycle Chain Mode* (manières de repartir une chaîne)

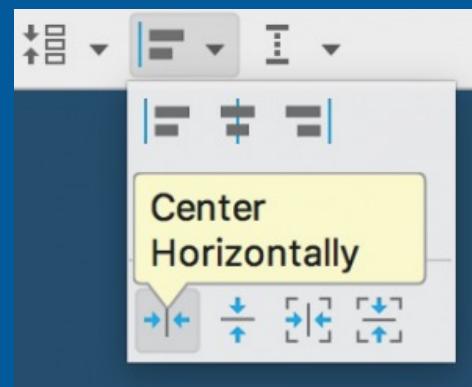


# Ajouter une chaîne

Pour ajouter une chaîne, sélectionnez les widgets à lier puis :

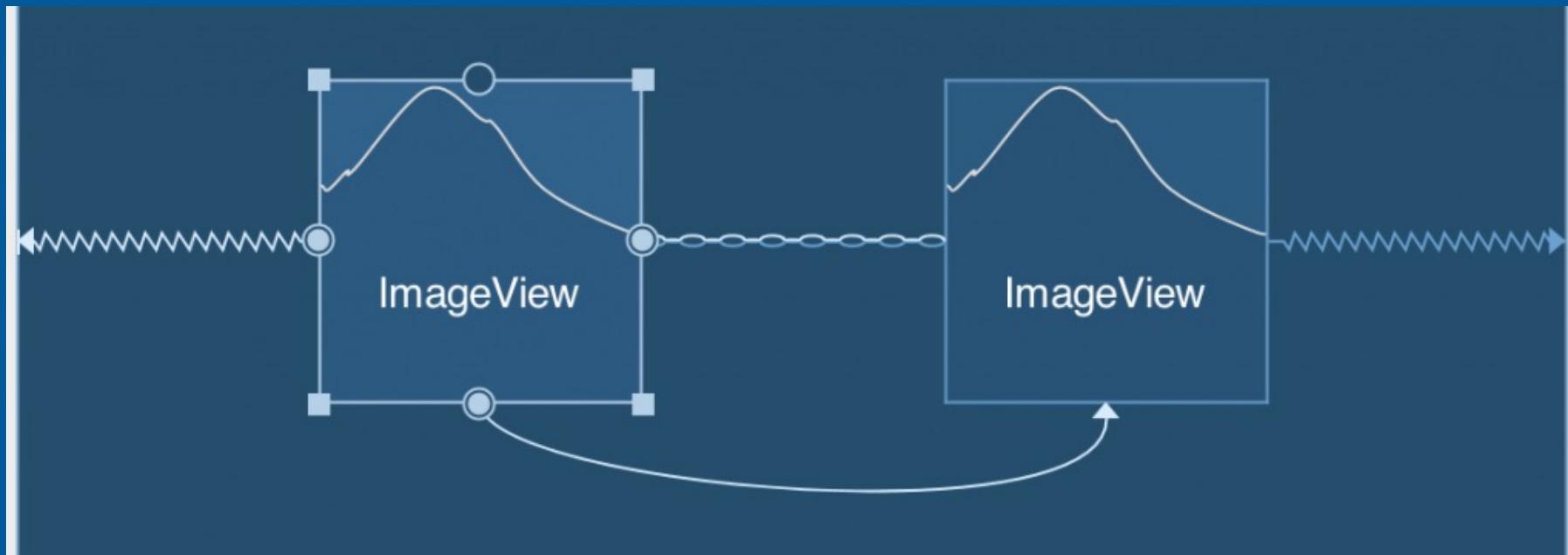


OU



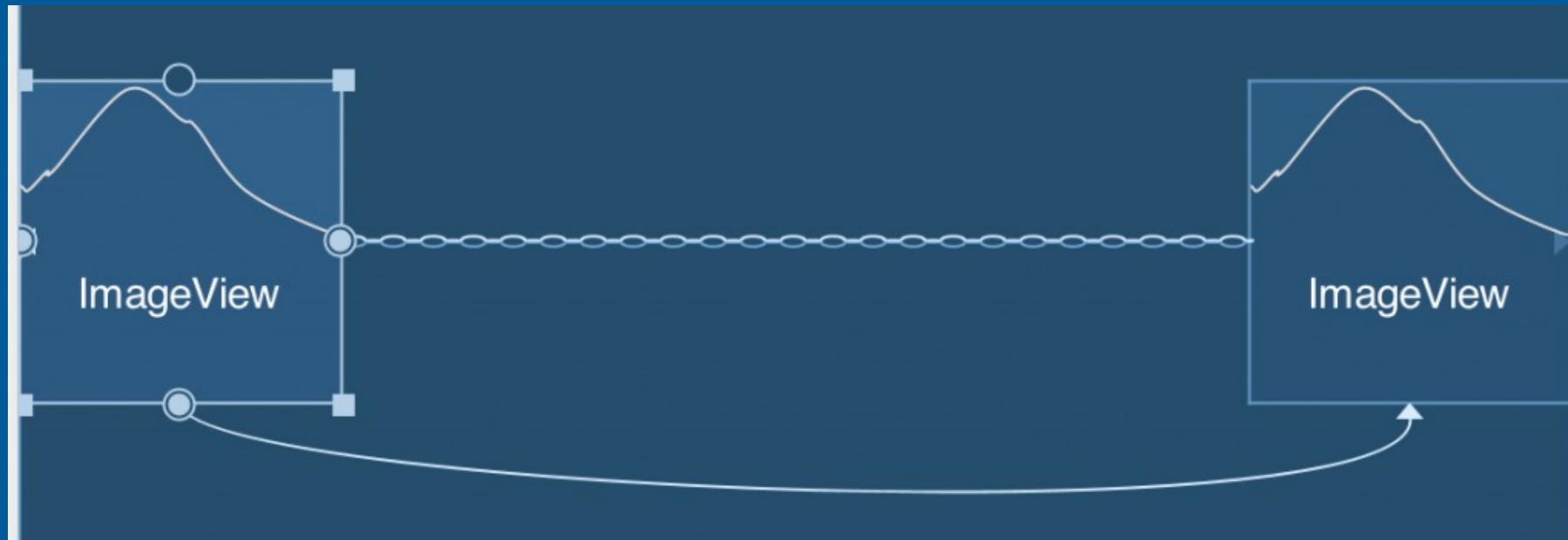
# Spread (par défaut)

Les widgets sont répartis uniformément sur l'axe.



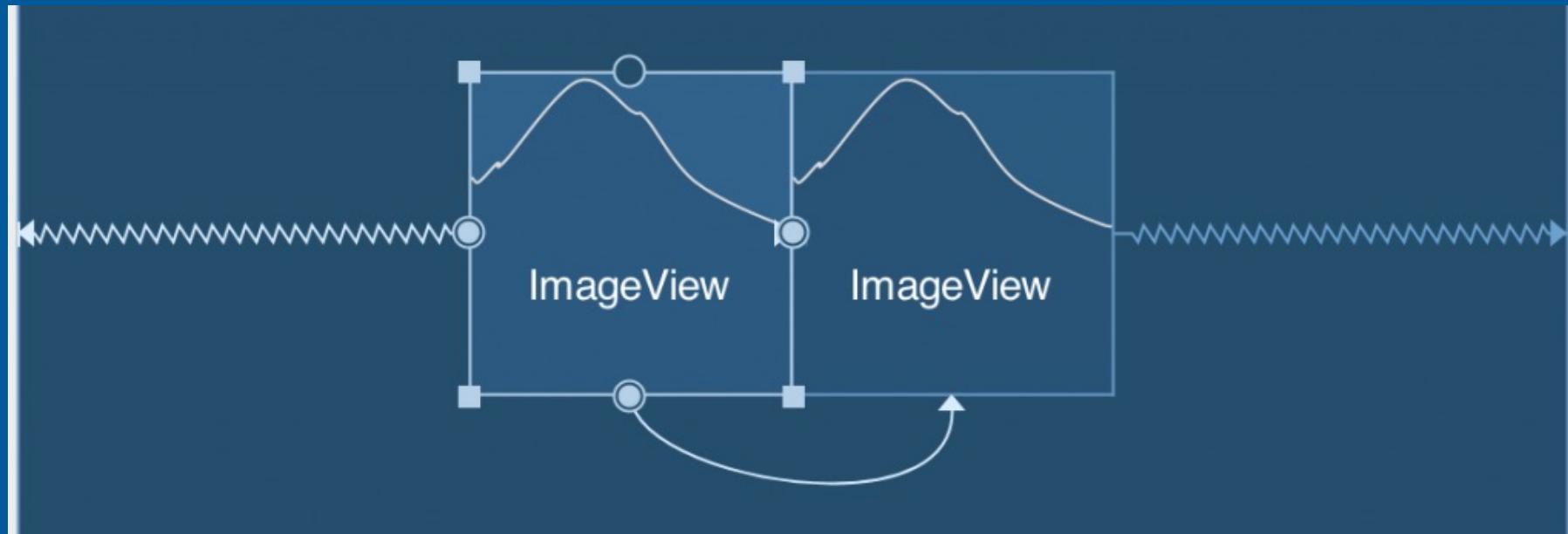
# Spread inside

Le premier et dernier widget sont alignés respectivement à gauche (ou haut) et à droite (ou bas), le reste de l'espace est réparti uniformément.



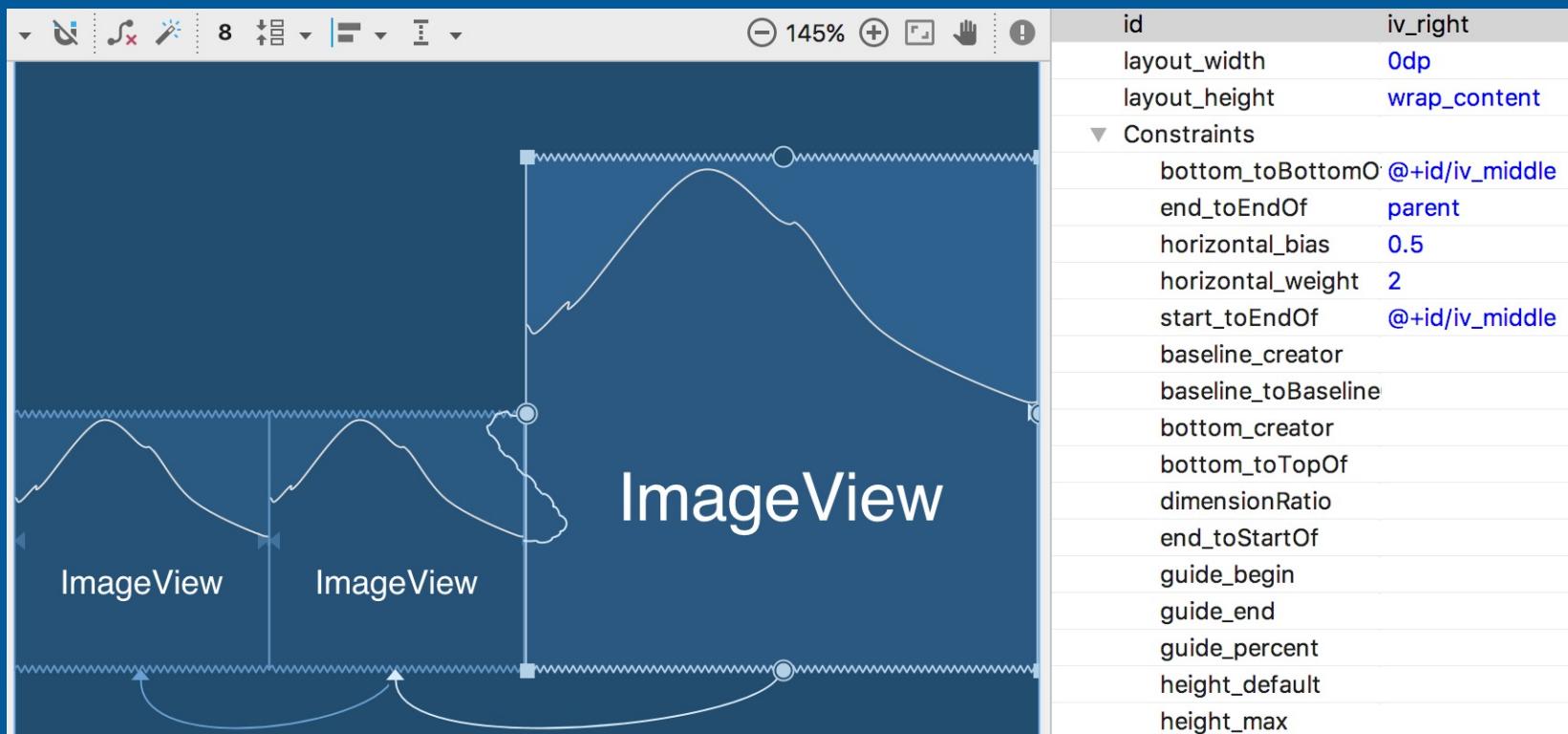
# Packed

Les widgets sont regroupés. Il est possible dans ce mode de déplacer la chaîne en ajustant le bias d'un des widgets.



# Weighted

Avec le mode Spread et Spread inside vous pouvez, à la manière du LinearLayout, attribuer un « poids » aux différents widgets de la chaîne pour qu'ils occupent plus ou moins de place. Dans l'exemple ci-dessous l'ImageView de droite possède un poids de 2 (contrainte « horizontal\_weight ») quand les 2 précédentes ont un poids de 1.



# Les biais

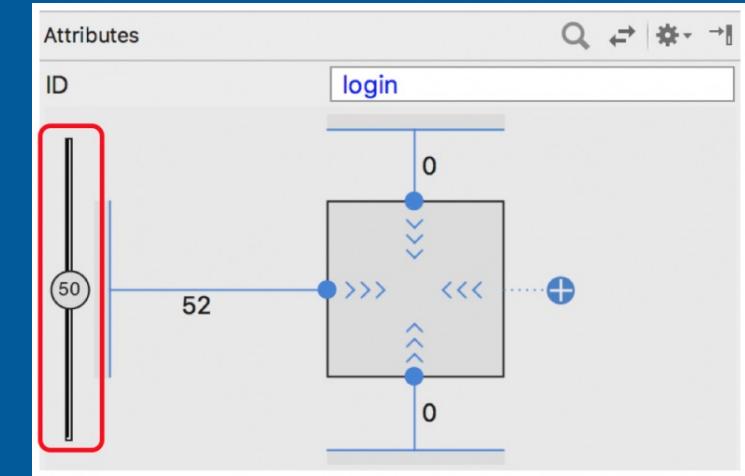
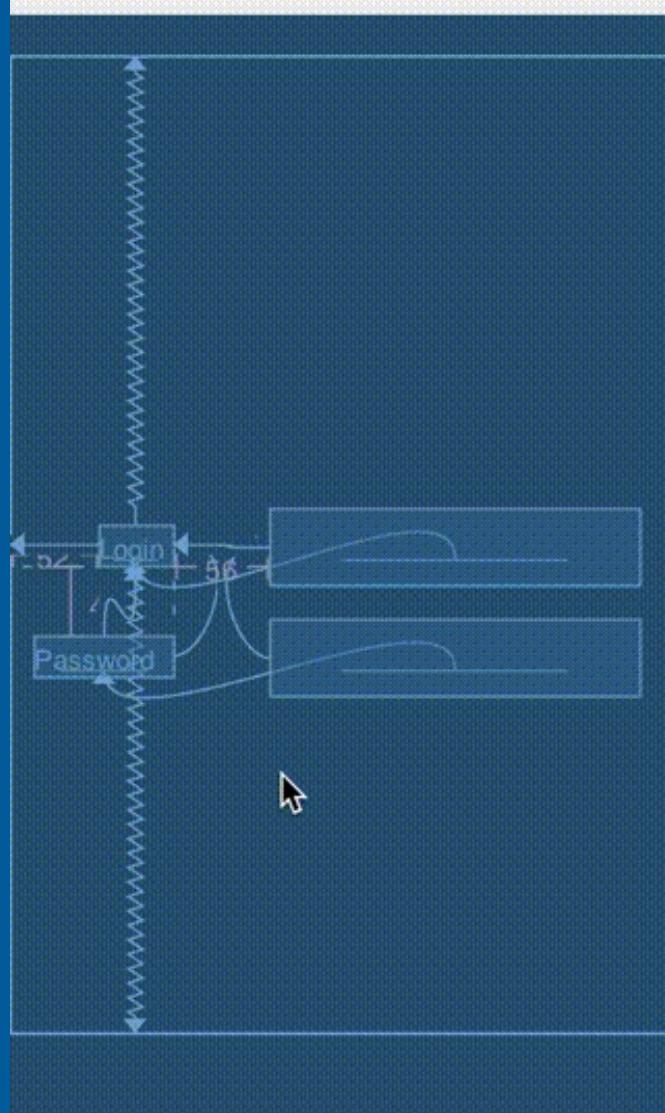
- Permet de déplacer une vue à un pourcentage près entre deux contraintes opposées (haut/bas ou gauche/droite)
- Il existe donc deux types de biais : vertical et horizontal



# Ajouter un biais



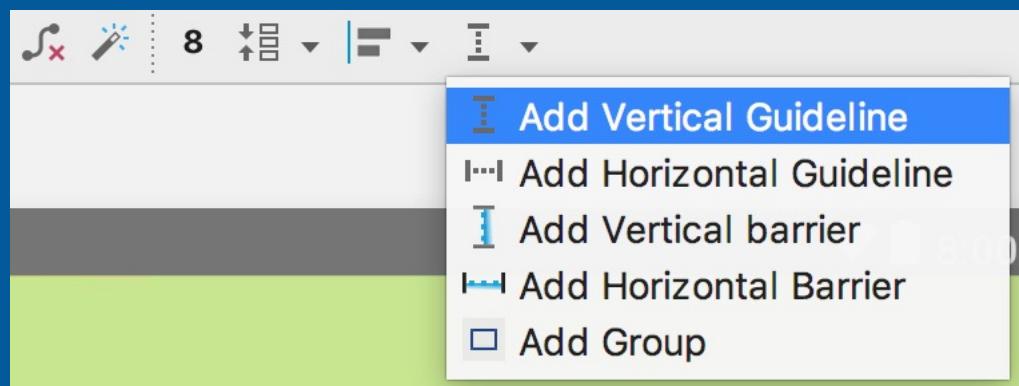
# Configurer un biais



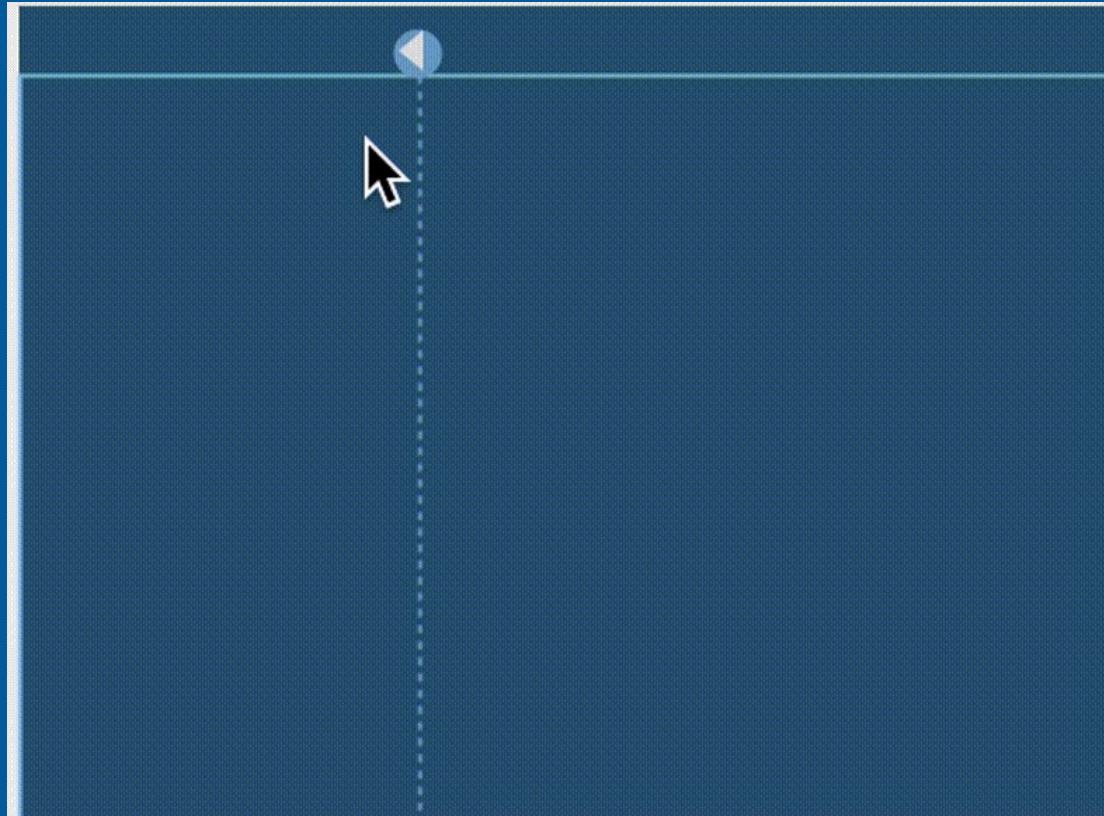
# Les Guidelines

- Une Guideline est un marquage (qui n'apparaît pas sur l'écran final) représentant une ligne de référence
- Elle peut être verticale ou horizontale
- Nos vues pourront venir s'y ancrer.
- Elle peut être utilisée, par exemple, pour éviter les répétitions de « Margin »

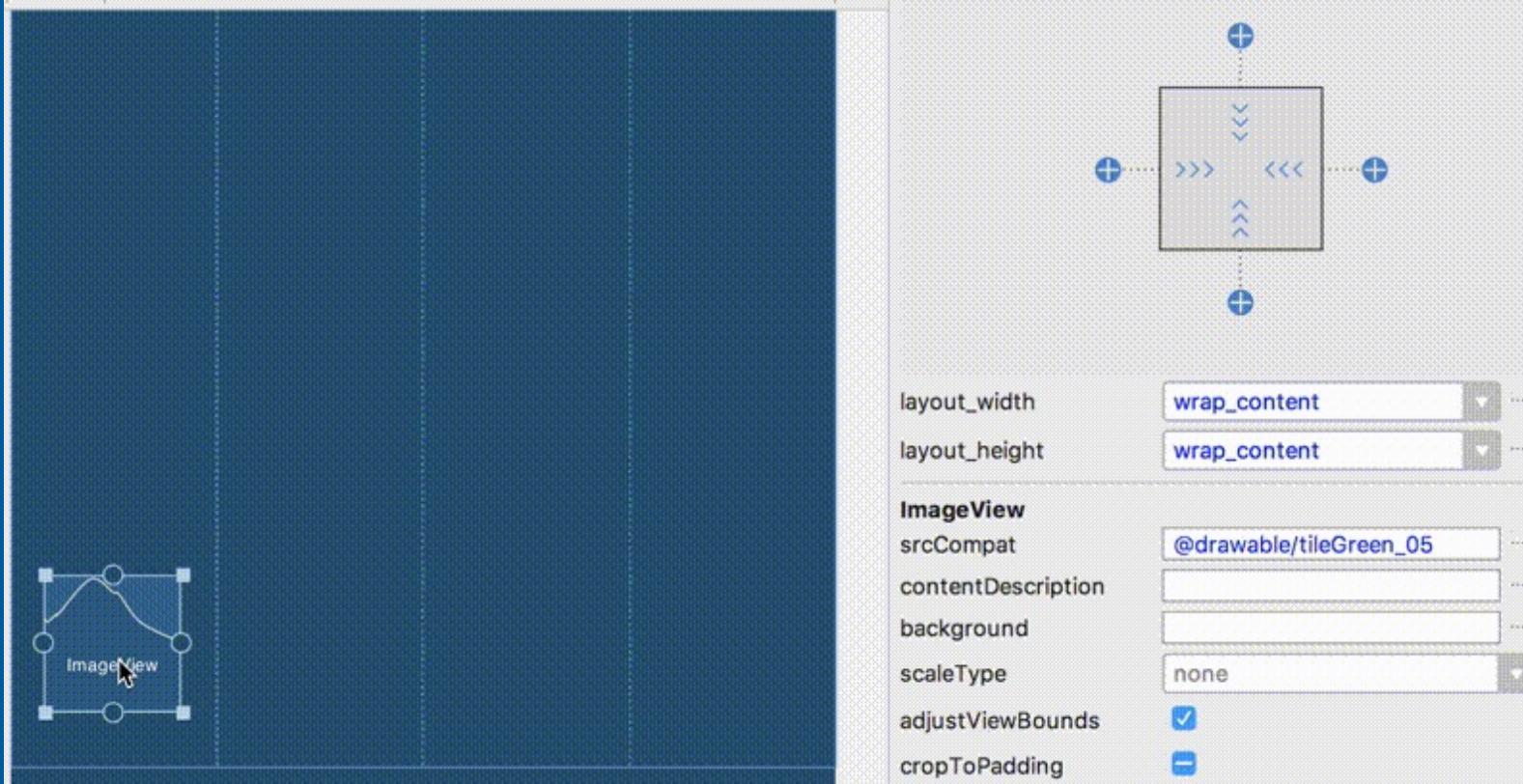
# Ajouter une Guideline



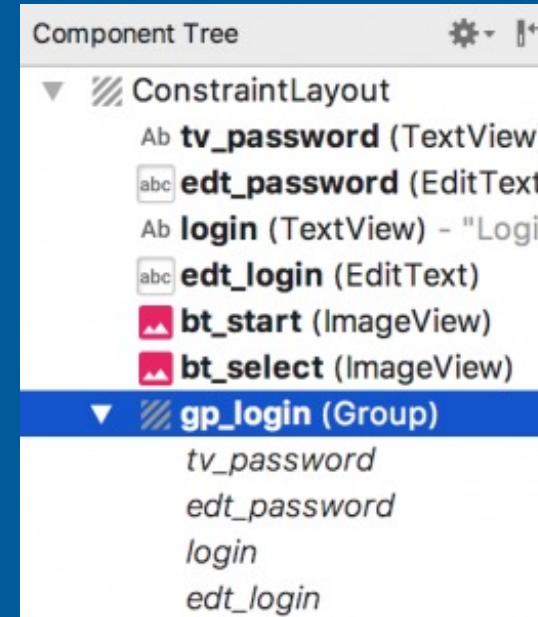
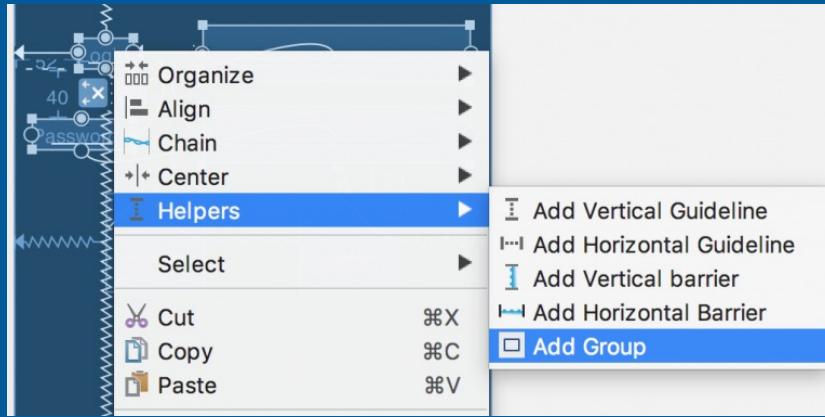
# Configurer une Guideline



# Placer des vues avec des Guidelines

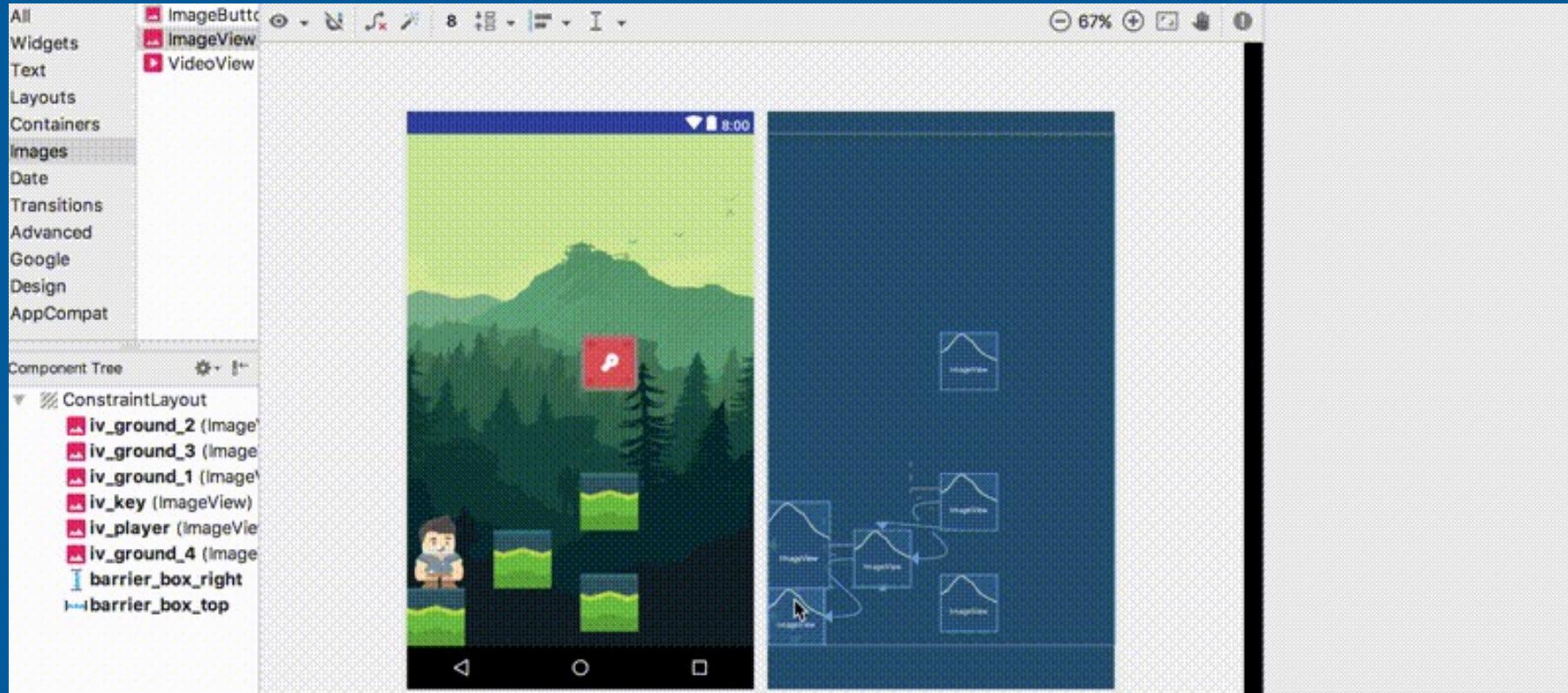


# Les Groups



```
<android.support.constraint.Group  
    android:id="@+id/gp_login"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    app:constraint_referenced_ids="tv_password,edt_password,login,edt_login" />
```

# Les Barriers



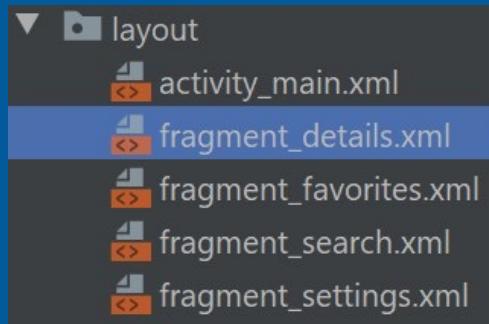
# TP – iPlant (2/10)



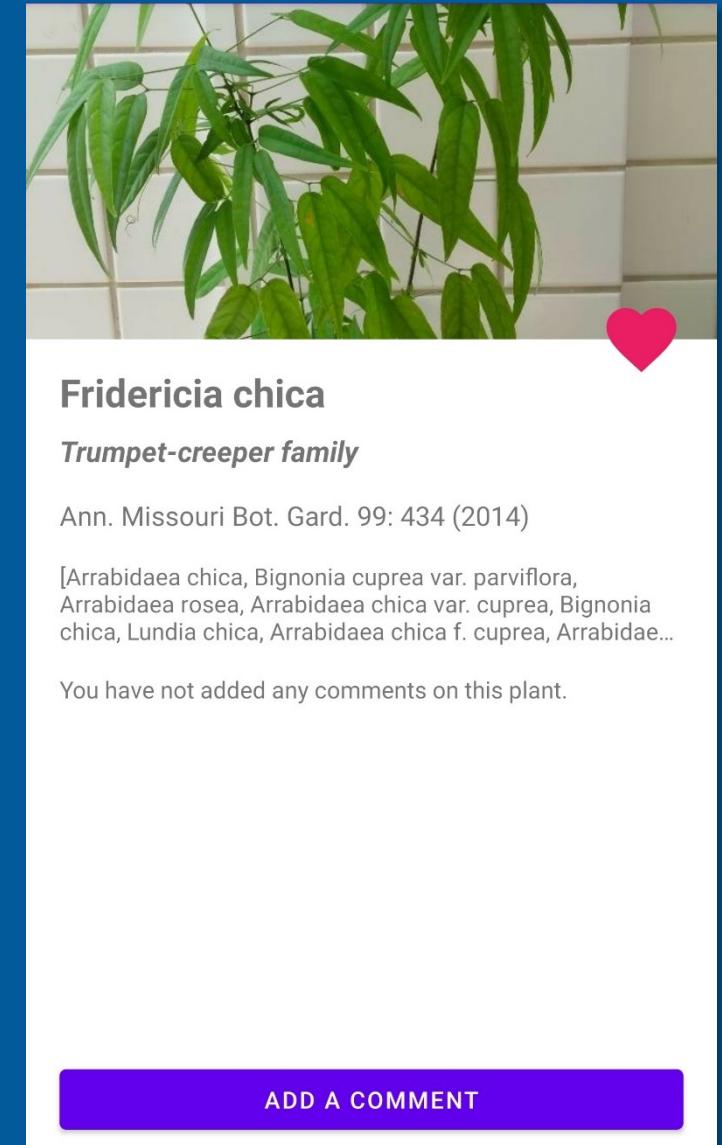
## Création du Layout de détails

Le layout visible à droite nous servira à afficher le détail d'une plante.

Créer un layout similaire dans le fichier fragment\_details.xml du projet :



Ce layout devra être réalisé grâce à ConstraintLayout.



# Interagir avec les vues

---

Assesseurs, mutateurs et listeners



# Databinding

- Databinding (« liaison de données » en anglais) est une bibliothèque de prise en charge qui vous permet de lier des composants d'interface utilisateur dans vos layouts aux sources de données de votre application à l'aide d'un format déclaratif plutôt que par programme.
- Les mises en page sont souvent définies dans les activités avec du code qui appelle les méthodes du framework d'interface utilisateur. Par exemple, le code ci-dessous appelle **findViewById()** pour rechercher un widget **TextView** et le lier à la propriété **userName** de la variable **viewModel**:

```
findViewById<TextView>(R.id.sample_text).apply {  
    text = viewModel.userName  
}
```

- L'exemple suivant montre comment utiliser le databinding pour affecter du texte au widget **TextView** directement dans le layout. Cela supprime la nécessité d'appeler le code Kotlin indiqué ci-dessus. Notez l'utilisation de la syntaxe **@{}** dans l'expression d'affectation:

```
<TextView  
    android:text="@{viewmodel.userName}" />
```

- La databinding vous permet de supprimer de nombreux appels de l'interface utilisateur dans vos classes (activités, fragments,...), ce qui les rend plus simples et plus faciles à gérer. Cela peut également améliorer les performances de votre application et éviter les fuites de mémoire et les « null pointer exceptions ».



# Activer le databinding dans son projet

- Pour commencer avec le databinding, activez l'option de génération dataBinding dans votre fichier build.gradle (module d'application) :

```
android {  
    ...  
    buildFeatures {  
        dataBinding true  
    }  
}
```

- Ajoutez aussi le plugin « kotlin-kapt » pour générer du code annoté au moment de la compilation :

```
plugins {  
    id 'com.android.application'  
    id 'kotlin-android'  
    id 'kotlin-kapt'  
}
```



# Premiers pas

- Le databinding génère automatiquement les classes requises pour lier les vues de la mise en page avec vos objets de données.
- Les layouts de databindings sont légèrement différents et commencent par une balise layout qui contient :
  - Une balise data qui contiendra vos imports et vos variables.
  - Puis à la suite, la balise d'une ViewGroup (LinearLayout, ConstraintLayout,...). Cet élément de vue est ce que serait votre racine dans un layout classique.

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable name="user" type="com.example.User" /> ←
    </data>
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{user.firstName}" /> ←
        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{user.lastName}" />
    </LinearLayout>
</layout>
```

La variable **user** dans **data** décrit une propriété qui peut être utilisée dans ce **layout**.

Les expressions sont écrites dans les propriétés d'attribut à l'aide de la syntaxe "@{}". Ici, le texte de la **TextView** est défini sur la propriété **firstName** de la variable **user**.



# Génération de classe en databinding

- Une classe de binding est générée pour chaque fichier de **layout**.
- Par défaut, le nom de la classe est basé sur le nom du fichier avec le suffixe Binding.
  - Par exemple, pour le layout **activity\_main.xml**, la classe générée correspondante est **ActivityMainBinding**.
- Cette classe contient toutes les liaisons des propriétés du **layout** (par exemple, la variable **user**) aux vues du layout et sait comment attribuer des valeurs aux expressions de liaison.
- La méthode recommandée pour créer les liaisons (bindings) est de le faire tout en gonflant (inflating) le layout :

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
  
    val binding: ActivityMainBinding = DataBindingUtil.setContentView(  
        this, R.layout.activity_main)  
  
    binding.user = User("Test", "User")  
}
```



# Databinding et LayoutInflater

Vous pouvez également obtenir la vue à l'aide d'un LayoutInflater:

```
val binding: ActivityMainBinding = ActivityMainBinding.inflate(LayoutInflater())
```

Si vous utilisez des éléments de databinding dans un Fragment, un adaptateur de ListView ou RecyclerView, préférez utiliser les méthodes inflate() des classes de binding ou de la classe DataBindingUtil:

```
val listItemBinding = ListItemBinding.inflate(LayoutInflater, viewGroup, false)
// or
val listItemBinding = DataBindingUtil.inflate(LayoutInflater, R.layout.list_item, viewGroup, false)
```



# TextView

Affiche du texte à l'utilisateur.

```
textView.setText(R.string.hello);  
textView.setTextSize(43);  
textView.setTextColor(0x112233);
```

OU

```
textView.text = getString(R.string.hello)  
textView.setTextSize = 43f  
textView.setTextColor(0x112233)
```



# EditText

Permet de saisir et modifier du texte.

Lorsque vous définissez ce widget, vous pouvez spécifier l'attribut inputType.

```
editText.hint = getString(R.string.hello)
editText.inputType = InputType.TYPE_TEXT_VARIATION_PASSWORD
editText.setLines(5)
val str : String = editText.text.toString()
Log.d( tag: "debugLog", msg: "voici le texte de l'EditText : $str")
```



# Checkbox

Type spécifique de bouton à deux états qui peut être coché ou décoché.

```
checkBox.setText(R.string.hello);  
checkBox.setChecked(true);  
  
if(checkBox.isChecked())  
    ; // do something
```



# Listener

- On a souvent besoin de gérer les interactions entre l'interface graphique et l'utilisateur (par exemple cliquer sur un bouton, entrer un texte, cocher une case, sélectionner un élément dans une liste,...).
- Pour pouvoir réagir à ces évènement, il faut utiliser un objet qui va détecter l'évènement de sorte à pouvoir le traiter.
- Ce type d'objet s'appelle un **listener**, une interface qui vous oblige à redéfinir des méthodes de *callback* et avec une méthode par évènement associé.
- Un listener est asynchrone



# Exemples de listeners

- sur un bouton (OnClickListener)
- sur un RadioGroup (RadioGroup.OnCheckedChangeListener)
- text watcher (Surveillance modification du texte des objets éditables)
- sur l'écran (mono-contact, OnTouchListener)
- sur l'accéléromètre et autres capteurs (SensorEventListener)
- Date() pour le widget et pour la dialogue
- Timer() pour le widget et pour la dialogue
- clique sur éléments (OnItemClickListener)



# Gestion du click

```
myButton.setOnClickListener { it: View!
    Log.d("myDebug", "on vient de clicker sur myButton")
}
```



# DatePicker

```
val calendar : Calendar! = Calendar.getInstance()
calendar.timeInMillis = System.currentTimeMillis()
dPicker.init(calendar.get(Calendar.YEAR), calendar.get(Calendar.MONTH), calendar.get(Calendar.DAY_OF_MONTH)) {
    : DatePicker!, year : Int, month : Int, dayOfMonth : Int) ->
    Log.d(tag: "Date", msg: "Year=" + year + " Month=" + (month + 1) + " day=" + dayOfMonth)
}
```



# AddTextChangeListener

```
editText.addTextChangedListener(object: TextWatcher {  
    override fun afterTextChanged(s: Editable?) {  
    }  
    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {  
    }  
    override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {  
    }  
})
```



# RadioGroup : OnCheckedChangeListener

```
radioGroup.setOnCheckedChangeListener { _, checkedId: Int ->
    var text = "You selected: "
    text += if (R.id.radioMale == checkedId) "male" else "female"
    Log.d("myDebug", text)
}
```

```
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/radioGroup"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent">

    <RadioButton
        android:id="@+id/radioMale"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:text="@string/male"/>

    <RadioButton
        android:id="@+id/radioFemale"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:text="@string/female"/>

</RadioGroup>
```



# MediaPlayer : setOnPreparedListener

```
mediaplayer = MediaPlayer.create( context: this, R.raw.beep_15sec)
mediaplayer?.setOnPreparedListener { it: MediaPlayer!
    println("READY TO GO")
}

pushButton.setOnTouchListener { _, event ->
    handleTouch(event)
    ^setOnTouchListener true
}
```



# TP – iPlant (3/10)



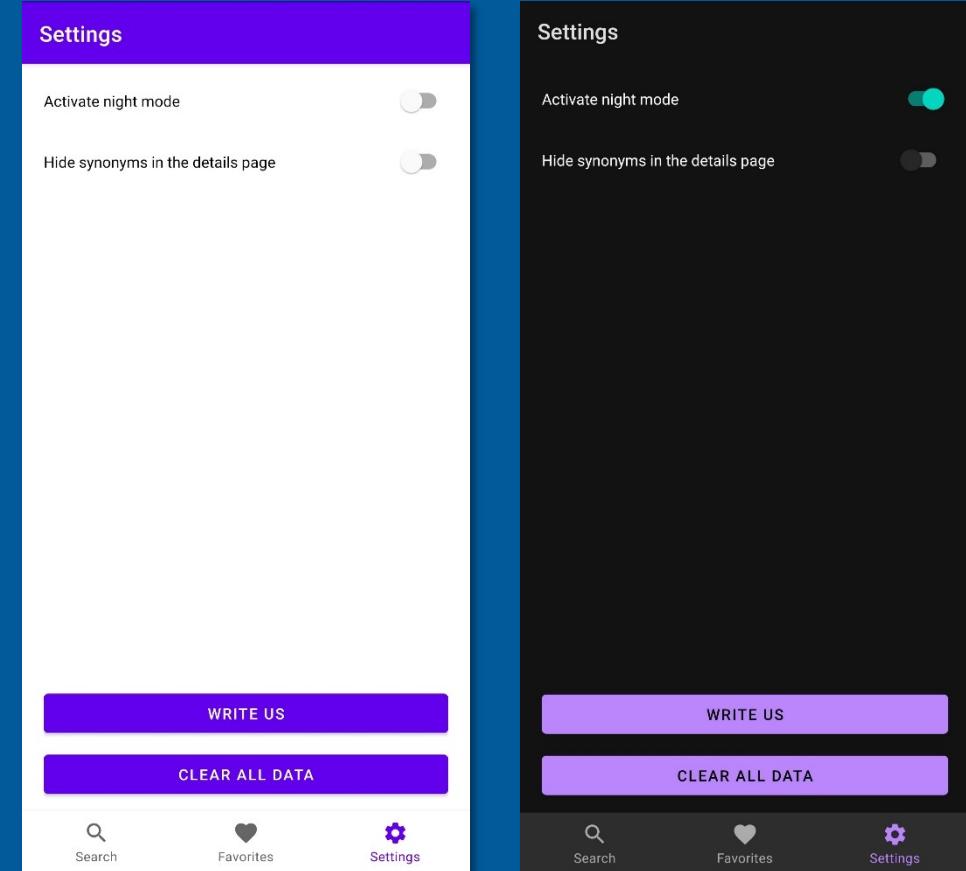
## Activation du mode nuit

Dans la partie Settings de l'application...

Faire en sorte que lorsque le switch lié au night mode change le thème de l'application.

Pour passer en mode nuit :

```
AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_YES)
```



# Les notifications utilisateur

---

Le rôle principal d'une notification est d'informer l'utilisateur lorsqu'un évènement se produit sur votre application.



# Toasts

```
Toast.makeText(this, "Androidly Short Toasts", Toast.LENGTH_SHORT).show()
Toast.makeText(this, "Androidly Long Toasts", Toast.LENGTH_LONG).show()
```

Fonction d'extension :

```
fun Context.toast(message: CharSequence) =
    Toast.makeText(this, message, Toast.LENGTH_SHORT).show()
```

Vous pouvez placer ceci n'importe où dans votre projet, là où vous le souhaitez. Par exemple, vous pouvez définir un fichier `mypackage.util.ContextExtensions.kt` et le placer là en tant que fonction de niveau supérieur. Chaque fois que vous avez accès à une instance de contexte, vous pouvez importer cette fonction et l'utiliser:

```
import mypackage.util.ContextExtensions.toast

fun myFun(context: Context) {
    context.toast("Hello world!")
}
```



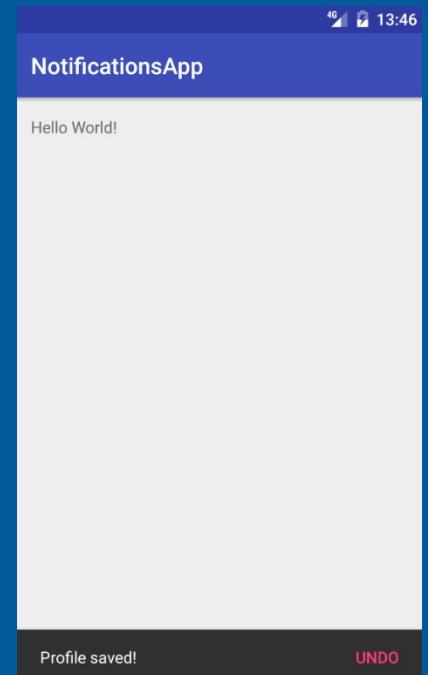
# Snackbar

Il nous faut implementer soit la lib de support design soit la lib androidX material :

com.android.support:design

com.google.android.material:material:1.0.0-rc01

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:id="@+id/mainLayout"
8     tools:context=".MainActivity">
9
10 </androidx.constraintlayout.widget.ConstraintLayout>
11
```



```
val snack : Snackbar = Snackbar.make(mainLayout, text: "This is a simple Snackbar", Snackbar.LENGTH_LONG)
snack.setAction( text: "DISMISS") { it: View!
    // executed when DISMISS is clicked
}
snack.show()
```



# Dialog

```
val dialog = Dialog(requireContext())
val dialogBinding: DialogConfirmationBinding = DataBindingUtil.inflate(
    LayoutInflater.from(requireContext()),
    R.layout.dialog_confirmation,
    parent: null,
    attachToParent: false
)

dialogBinding.btnExit.setOnClickListener { it: View!
    // TODO : when click on Yes
}

dialogBinding.btnExit.setOnClickListener { it: View!
    // TODO : when click on No
}

dialog.setContentView(dialogBinding.root)
dialog.show()
```

dialog.dismiss()

(pour fermer la Dialog)



# Notification

BasicNotifications • now ^

Notification Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pell..

```
var builder = NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle(textTitle)
    .setContentText(textContent)
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
```

Pour que la notification apparaisse, appelez `NotificationManagerCompat.notify()`, en lui transmettant un ID unique pour la notification et le résultat de `NotificationCompat.Builder.build()`



# TP – iPlant (4/10)



Dialog « Êtes-vous sûr ? »

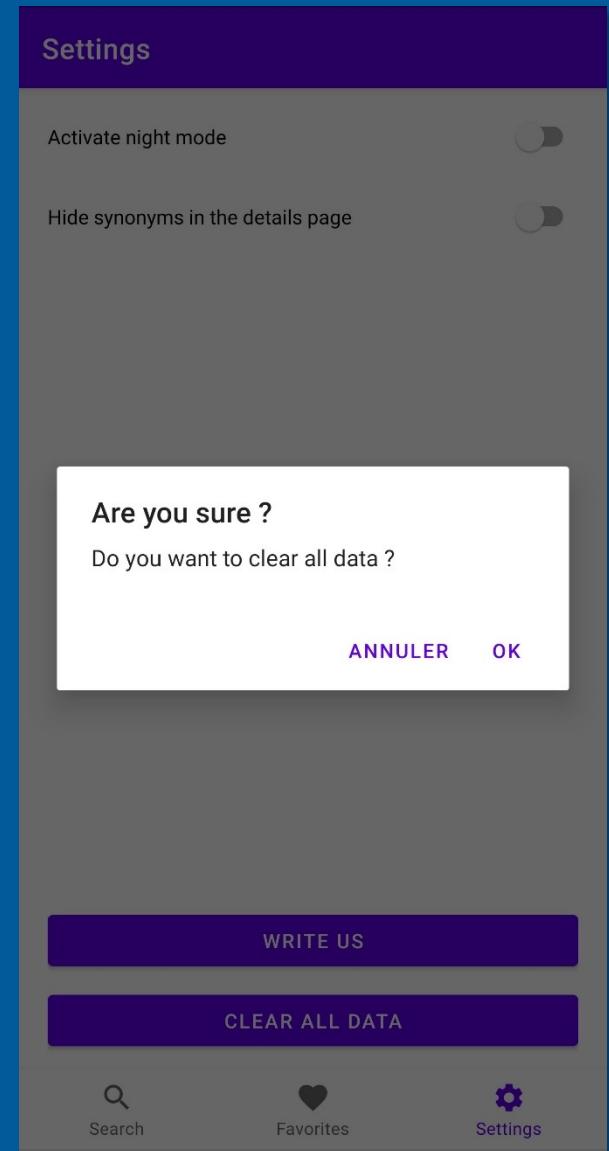
Dans la partie Settings de l'application...

Afficher une Dialog dans le cas où l'utilisateur clique sur « Clear all data ».

Cette fenêtre doit demander à l'utilisateur de confirmer le fait qu'il désire supprimer toutes ses données.

Si l'utilisateur click sur OK, afficher un toast avertissant l'utilisateur que les données ont bien été effacées.

Fermer la dialog si l'utilisateur click sur Annuler ou OK.



# Les activités

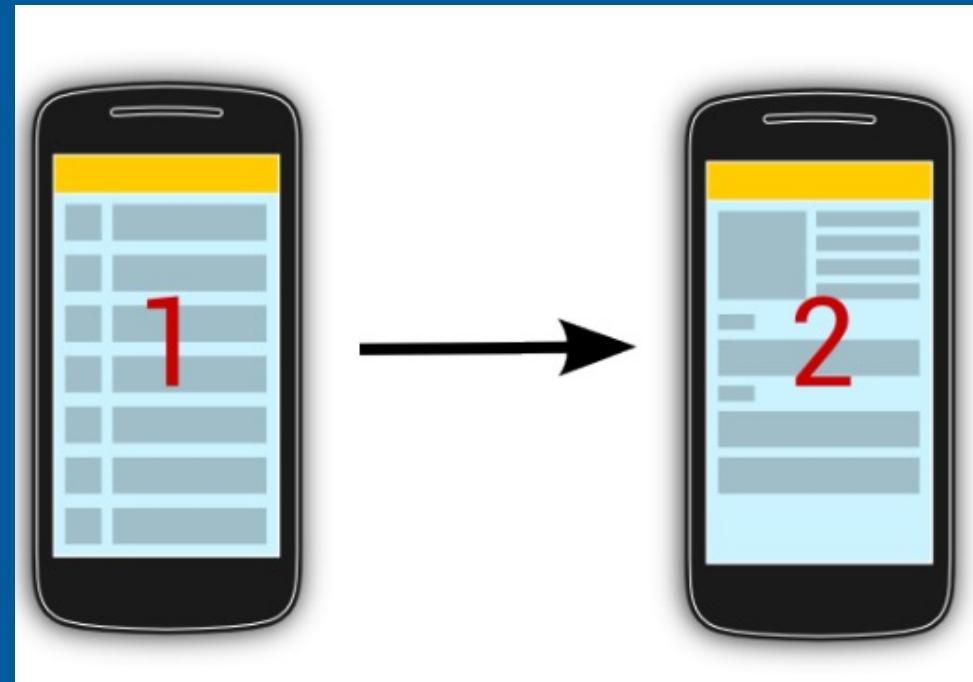
---

Une activité est une tâche unique et ciblée que l'utilisateur peut faire. Presque toutes les activités interagissant avec l'utilisateur, la classe d'activité se charge de créer une fenêtre dans laquelle vous pouvez placer votre interface utilisateur avec **setContentView(View)**.



# Qu'est-ce qu'une Activity ?

- La classe d'activité est un composant essentiel d'une application Android.
- La manière dont les activités sont lancées et regroupées est un élément fondamental du modèle d'application de la plate-forme.
- Contrairement aux paradigmes de programmation dans lesquels les applications sont lancées avec une méthode main(), le système Android initie du code dans une instance Activity en appelant des méthodes de rappel spécifiques correspondant à des étapes spécifiques de son cycle de vie.



# Context

- Interface avec des informations globales sur un environnement d'application.
- Il s'agit d'une classe abstraite dont l'implémentation est fournie par le système Android. Il permet d'accéder aux ressources et aux classes spécifiques à l'application, ainsi qu'aux appels supplémentaires pour les opérations au niveau de l'application, telles que les activités de lancement, les intentions de diffusion et de réception, etc.
- Pour faire simple, il s'agit du contexte de l'état actuel de l'application / de l'objet.
- Il permet aux objets nouvellement créés de comprendre ce qui se passe.
- En général, vous lappelez pour obtenir des informations sur une autre partie de votre programme (activité et package / application).



# Utilisation du Context

Le context est votre point d'accès pour les ressources liées à l'application

Context(s) illustrés :

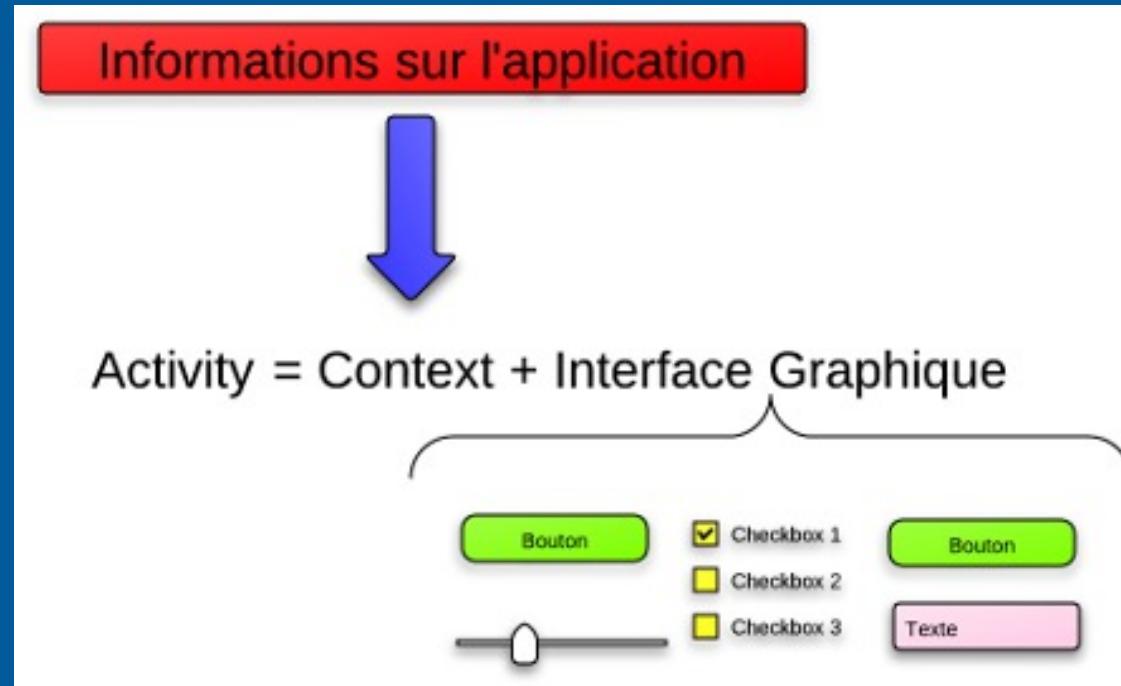


Utilisation d'un Context :

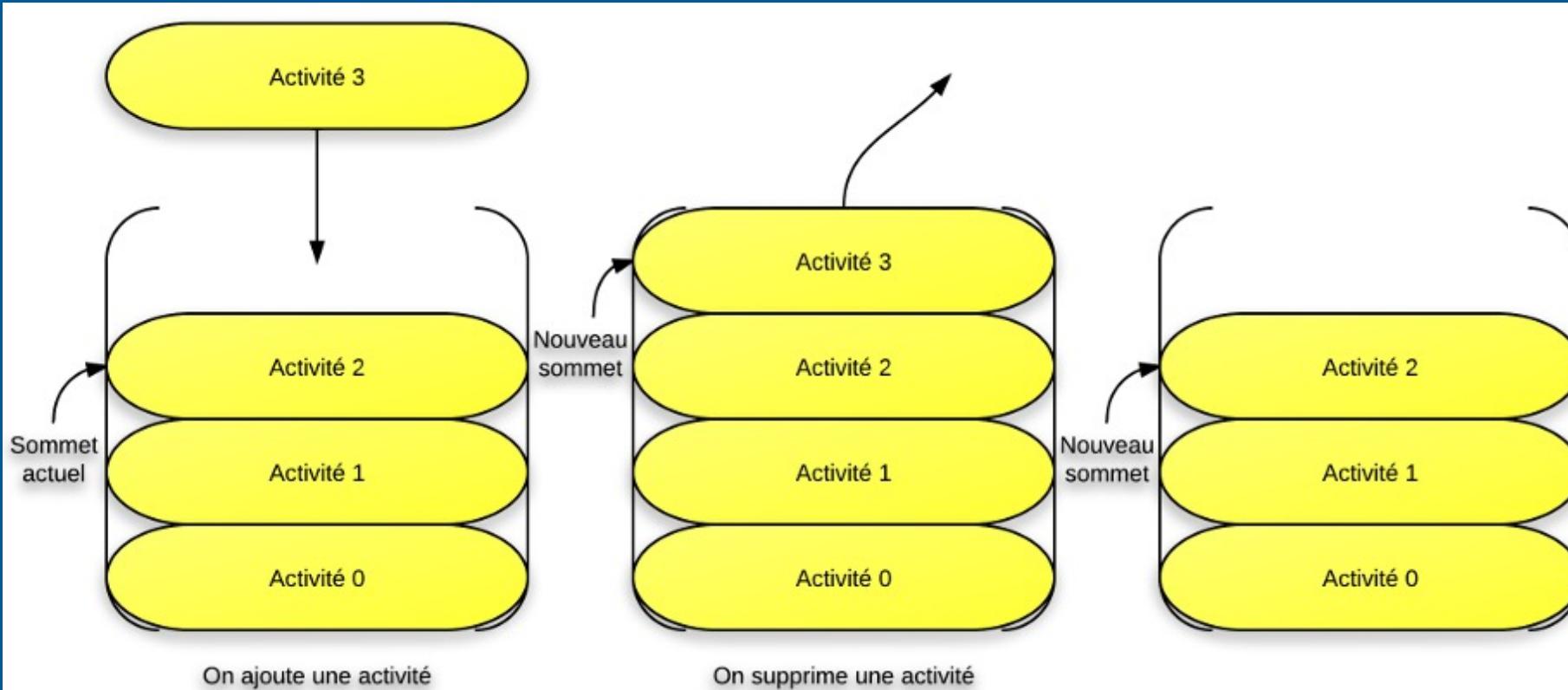
	<b>Application</b>	<b>Activity</b>	<b>Service</b>	<b>ContentProvider</b>	<b>BroadcastReceiver</b>
Show a Dialog	NO	YES	NO	NO	NO
Start an Activity	NO <sup>1</sup>	YES	NO <sup>1</sup>	NO <sup>1</sup>	NO <sup>1</sup>
Layout Inflation	NO <sup>2</sup>	YES	NO <sup>2</sup>	NO <sup>2</sup>	NO <sup>2</sup>
Start a Service	YES	YES	YES	YES	YES
Bind to a Service	YES	YES	YES	YES	NO
Send a Broadcast	YES	YES	YES	YES	YES
Register BroadcastReceiver	YES	YES	YES	YES	NO <sup>3</sup>
Load Resource Values	YES	YES	YES	YES	YES



# Activity, Context et interface graphique

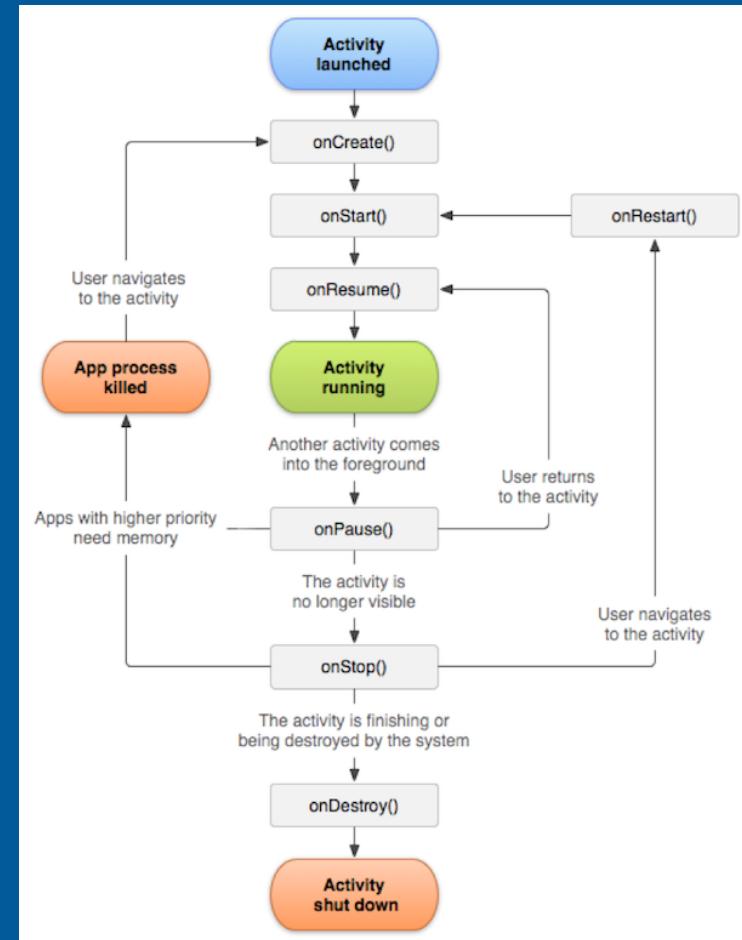


# Pile d'activités



# Cycle de vie d'une activité

- **onCreate**
  - méthode d'initialisation des vues, des paramètres et autres données.
- **onStart**
  - appelée quand l'activité est rendue visible à l'utilisateur.
- **onRestart**
  - appelée à un nouveau démarrage de la même activité (quand l'activité était arrêtée)
- **onResume**
  - appelée quand l'activité commence à interagir avec l'utilisateur.
- **onPause**
  - méthode qui sert à arrêter une activité temporairement.
- **onStop**
  - appelée quand l'activité n'est plus visible à l'utilisateur, soit à cause d'une nouvelle activité lancée, soit parce que l'activité en cours s'apprête à être détruite.
- **onDestroy**
  - est invoquée quand l'activité est détruite par la méthode `finish()` ou quand le système décide de tuer l'activité pour économiser de l'espace



# Pourquoi le cycle de vie est important ?

- L'utilisateur reçoit un appel téléphonique ou passe à une autre application tout en utilisant votre application.
- Perdre la progression de l'utilisateur s'il quitte votre application et y retourne plus tard
- Interruption ou perte de progression de l'utilisateur lorsque l'écran pivote entre l'orientation paysage et portrait...
- Données périmée si l'utilisateur revient sur l'app plusieurs jours après

```
public class Main extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.acceuil); }  
    protected void onDestroy() {  
        super.onDestroy(); }  
    protected void onPause() {  
        super.onPause(); }  
    protected void onResume() {  
        super.onResume(); }  
    protected void onStart() {  
        super.onStart(); }  
    protected void onStop() {  
        super.onStop(); } }
```



# Context

- Référence à des ressources gérées par l'application, des informations sur le système et accès à des services d'Android.
- Est employé pour lancer une nouvelle activité, réceptionner les intentions ou écouter les événements.
- Exemples :
  - création de nouveaux objets (TextView, Adapter...)
  - accès à des ressources (String, Array...)
  - accès implicite à des components, par exemple ***getContext().getContentResolver()***
- Comment récupérer le contexte ?
  - ***getApplicationContext()***
  - ***getContext()***
  - ***getBaseContext()***
  - ***this*** (*disponible uniquement depuis une activité*)

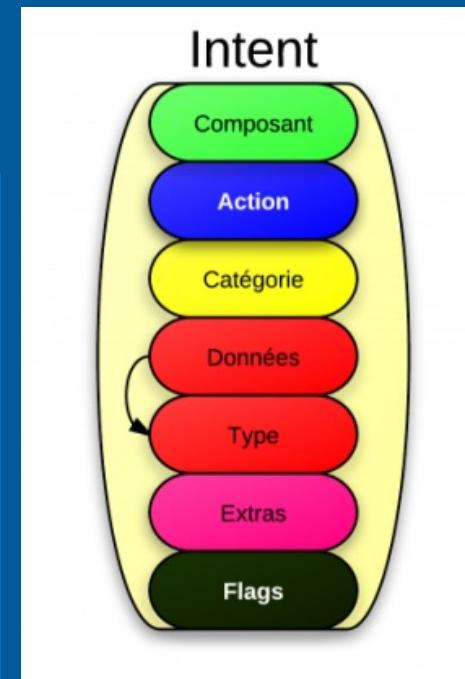


# Intent

- Ici nous permet de naviguer entre les écrans (*d'une Activity à une autre*)
- Pour les activités, on distingue deux type d'intentions :
  - **explicite** : on spécifie le nom de l'activité à invoquer.
  - **implicite** : on ne sait pas quelle activité doit être invoquée : Android se charge de trouver l'activité adéquate à lancer (exemple : pour un site web, il va lancer un des navigateurs disponibles).

Passage de `MainActivity()` à `NextActivity()` :

```
val myIntent = Intent(applicationContext, Activity2::class.java)
startActivity(myIntent)
```



# Bundle

- Le Bundle est une instance de classe passée en paramètre dans la méthode **onCreate()**.
- Cette classe est une sorte de conteneur pour les données transmissibles d'une activité à l'autre. Elle permet de récupérer tout type de données :
  - *String, long, char, ArrayList, etc...*

Envoyer un **extra** de type **int** à une autre **Activity** :

```
val myIntent = Intent(applicationContext, Activity2::class.java)
val extras = Bundle()
extras.putInt("age", 36)
myIntent.putExtras(extras)
startActivity(myIntent)
```

Recevoir un **int** dans l'autre **Activity** :

```
val extras: Bundle? = intent.extras
val age: Int? = extras?.getInt(key: "age")
```



# Exemple

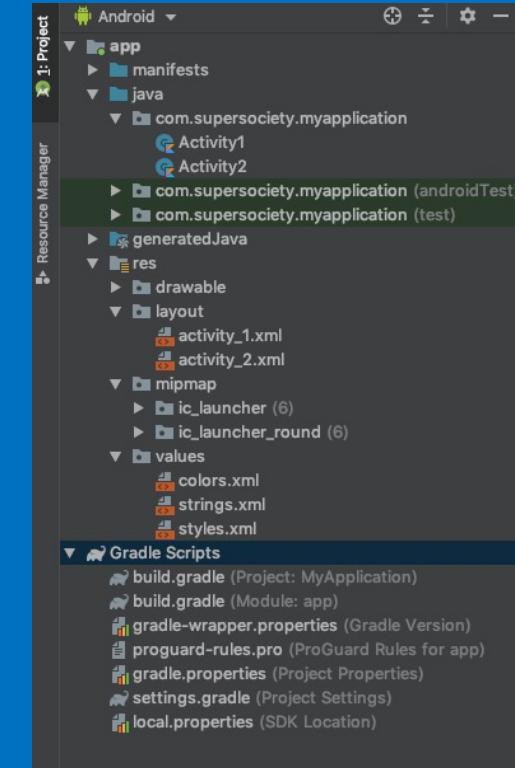
MainActivity.java :

```
6  class Activity1 : AppCompatActivity() {  
7      override fun onCreate(savedInstanceState: Bundle?) {  
8          super.onCreate(savedInstanceState)  
9          setContentView(R.layout.activity_1)  
10         val myIntent = Intent(applicationContext, Activity2::class.java)  
11         val extras = Bundle()  
12         extras.putInt("age", 36)  
13         myIntent.putExtras(extras)  
14         startActivity(myIntent)  
15     }  
16 }  
17  
18 }  
19  
20 }  
21 }
```

NextActivity.java :

```
8  override fun onCreate(savedInstanceState: Bundle?) {  
9      super.onCreate(savedInstanceState)  
10     setContentView(R.layout.activity_2)  
11  
12     val extras :Bundle? = intent.extras  
13     val age :Int? = extras?.getInt( key: "age")  
14     Log.d( tag: "myDebug", msg: "the age is : $age")  
15  
16 }  
17  
18 }  
19  
20 }
```

Structure du projet :



# INTENT IMPLICITES

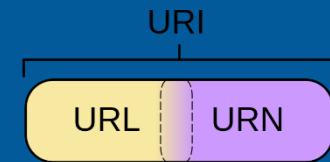
Un Intent est un ensemble de données qui peut être passé à un autre composant applicatif (de la même application ou non) de façon implicite (requête pour une action - lire de la musique ou scanner un code barre par exemple) ou explicite (lancement d'une classe précise)



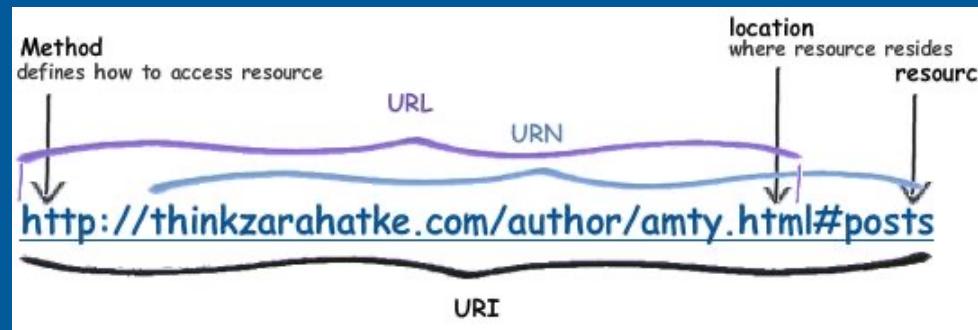
# C'est quoi ?

- Un Intent implicite permet de laisser le soin à Android de chercher l'app la plus adaptée pour notre action.
- Par exemple :
  - Ouvrir un URL
  - Envoyer un SMS
  - Partager quelque chose
  - Écouter une musique
  - Scanner un code-bar
  - Etc.

# Uniform Resource Identifier (URI)



Une URI est une chaîne de caractère respectant une norme et permettant d'identifier une ressource sur un réseau de manière permanente, même si cette ressource est déplacée ou supprimée. L'URL est une URI pour les ressources disponibles sur Internet, au même titre que l'ISBN est une URI pour les livres puisqu'il s'agit de l'identifiant unique d'un livre.



# Exemples URI

- <ftp://ftp.is.co.za/rfc/rfc1808.txt> [archive] : protocole *FTP* pour le service *File Transfer Protocol*
- <http://www.math.uio.no/faq/compression-faq/part1.html> [archive] : protocole *HTTP* pour le service *Hypertext Transfer Protocol*
- <mailto:mduerst@ifi.unizh.ch> [archive] : protocole *mailto* pour les adresses électroniques
- <news:comp.infosystems.www.servers.unix> [archive] : protocole *news* pour les newsgroups Usenet
- <telnet://melvyl.ucop.edu/> [archive] : protocole *telnet* pour les services interactifs via *telnet*
- <irc://irc.freenode.net/ubuntu-fr> [archive] : protocole *IRC* pour le service *Internet Relay Chat*
- <ssh://utilisateur@example.com> [archive] : protocole *SSH* pour le service *Secure shell*
- <sftp://utilisateur@example.com> [archive] : protocole *SFTP* pour le service *SSH file transfer protocol*
- <http://example.org/URI/absolu/avec/chemin/absolu/vers/une/ressource> [archive]
- /URI/relatif/avec/chemin/absolu/vers/une/ressource
- chemin/relatif/vers/une/ressource
- ../../ressource
- ./ressource#fragment
- ressource
- #fragment



# Quelques actions natives

Intitulé	Action	Entrée attendue	Sortie attendue
ACTION_MAIN	Pour indiquer qu'il s'agit du point d'entrée dans l'application	/	/
ACTION_DIAL	Pour ouvrir le composeur de numéros téléphoniques	Un numéro de téléphone semble une bonne idée :-p	/
ACTION_DELETE*	Supprimer des données	Un URI vers les données à supprimer	/
ACTION_EDIT*	Ouvrir un éditeur adapté pour modifier les données fournies	Un URI vers les données à éditer	/
ACTION_INSERT*	Insérer des données	L'URI du répertoire où insérer les données	L'URI des nouvelles données créées
ACTION_PICK*	Sélectionner un élément dans un ensemble de données	L'URI qui contient un répertoire de données à partir duquel l'élément sera sélectionné	L'URI de l'élément qui a été sélectionné
ACTION_SEARCH	Effectuer une recherche	Le texte à rechercher	/
ACTION_SENDTO	Envoyer un message à quelqu'un	La personne à qui envoyer le message	/
ACTION_VIEW	Permet de visionner une donnée	Un peu tout. Une adresse e-mail sera visionnée dans l'application pour les e-mails, un numéro de téléphone dans le composeur, une adresse internet dans le navigateur, etc.	/
ACTION_WEB_SEARCH	Effectuer une recherche sur internet	S'il s'agit d'un texte qui commence par « http », le site s'affichera directement, sinon c'est une recherche dans Google qui se fera	/



# Ouvrir le navigateur

```
val myIntent = Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.com"))
startActivity(myIntent)
```

OU

```
val myIntent = Intent(Intent.ACTION_VIEW)
myIntent.data = Uri.parse("http://www.google.com")
startActivity(myIntent)
```



# Envoyer un SMS

```
val myIntent = Intent(Intent.ACTION_VIEW, Uri.parse(uriString: "sms://0612345678"))
myIntent.putExtra(name: "sms_body", value: "ceci est un message")
startActivity(myIntent)
```

OU

```
val myIntent = Intent(Intent.ACTION_VIEW)
myIntent.data = Uri.parse(uriString: "sms://0612345678")
myIntent.putExtra(name: "sms_body", value: "ceci est un message")
startActivity(myIntent)
```



# Envoyer un email

```
val myIntent = Intent(Intent.ACTION_SENDTO)
myIntent.data = Uri.parse("mailto://abc@gmail.com")
myIntent.putExtra(Intent.EXTRA_SUBJECT, "ceci est le sujet du mail")
myIntent.putExtra(Intent.EXTRA_TEXT, "ceci est le message du mail")
startActivity(myIntent)
```



# D'autres exemples

```
val myIntentMail = Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.com"))
val myIntentDial = Intent(Intent.ACTION_DIAL, Uri.parse("tel:(+49)12345678"))
val myIntentGeo = Intent(Intent.ACTION_VIEW, Uri.parse("geo:50.123,7.1434?z=19"))
val myIntentContact = Intent(Intent.ACTION_VIEW, Uri.parse("content://contacts/people"))
```



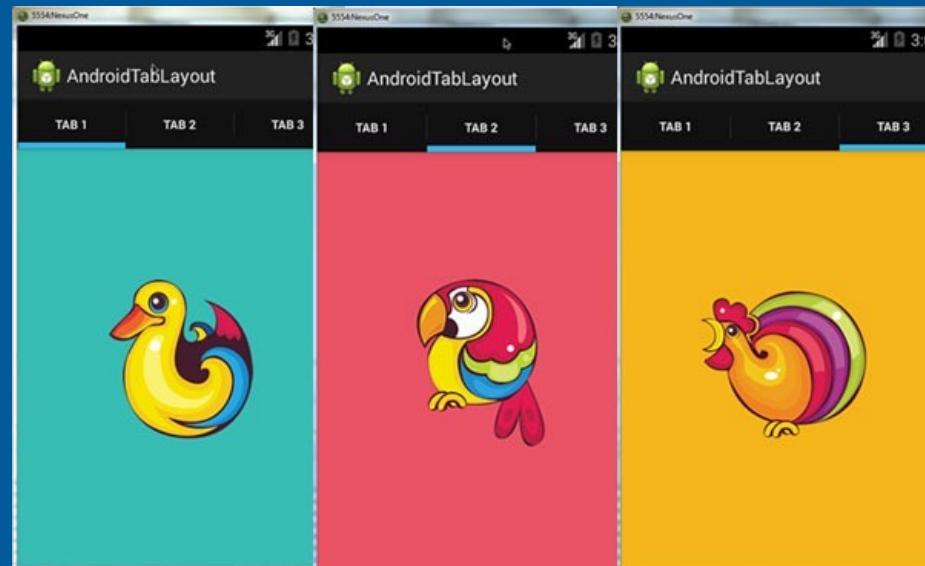
# Les fragments

Un fragment est un composant indépendant qui doit être utilisé dans une activité. Un fragment encapsule des fonctionnalités qui le rendent facile à réutiliser dans une activité ou dans une mise en page.



# Un fragment, c'est quoi ?

- Un **fragment** est un composant indépendant qui peut être utilisé dans une activité. Un fragment encapsule des fonctionnalités qui le rendent facile à réutiliser dans une activité ou dans une mise en page.
- Un fragment s'exécute dans le contexte d'une activité mais possède son propre cycle de vie et typiquement, il possède une interface utilisateur. Il est aussi possible de définir des fragments sans interface utilisateur, par exemple des fragments sans entête.
- Les fragments peuvent être ajoutés à une activité de manière statique ou dynamique.

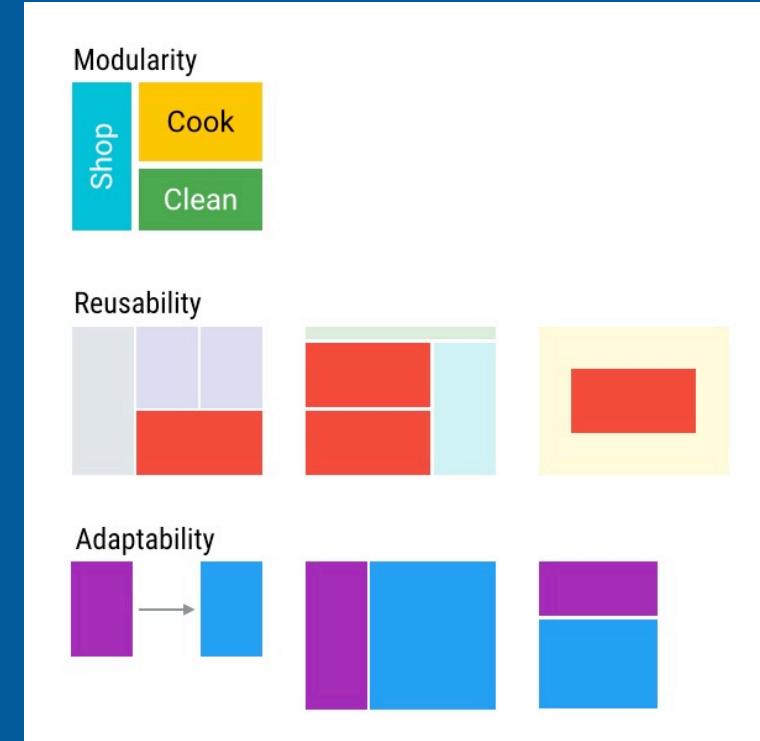
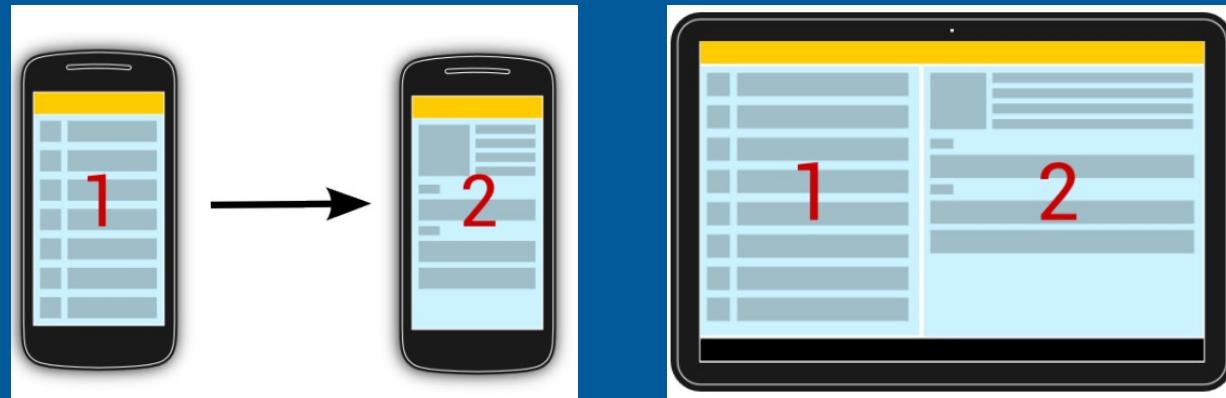


# Avantage de l'utilisation des fragments

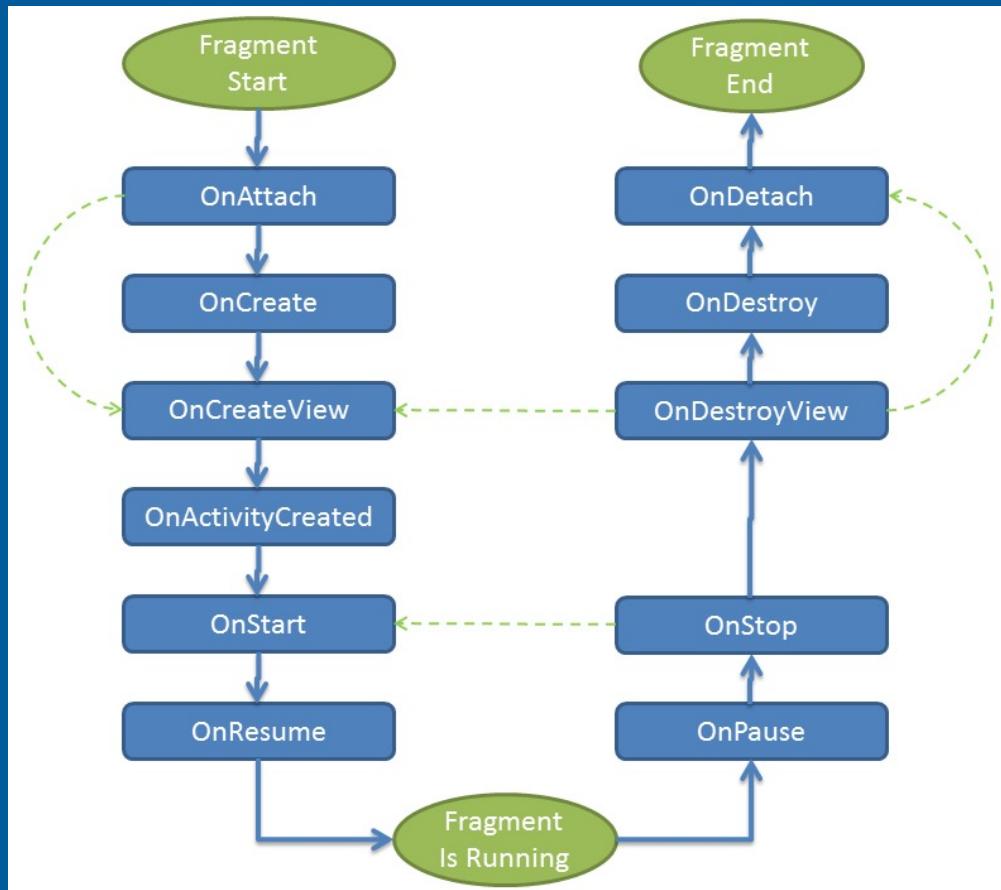
**Modularité:** division du code d'activité complexe en fragments pour une meilleure organisation et maintenance

**Réutilisabilité:** placer des éléments de comportement ou d'interface utilisateur en fragments pouvant être partagés par plusieurs activités

**Adaptabilité:** Représenter les sections d'une interface utilisateur sous forme de fragments et utiliser différentes dispositions en fonction de l'orientation et de la taille de l'écran.



# Cycle de vie d'un Fragment



# Lien entre cycle de vie d'une Activity et d'un Fragment

La gestion du cycle de vie d'un fragment s'apparente beaucoup à la gestion du cycle de vie d'une activité. Comme une activité, un fragment peut exister dans trois états:

## 1. Resumed

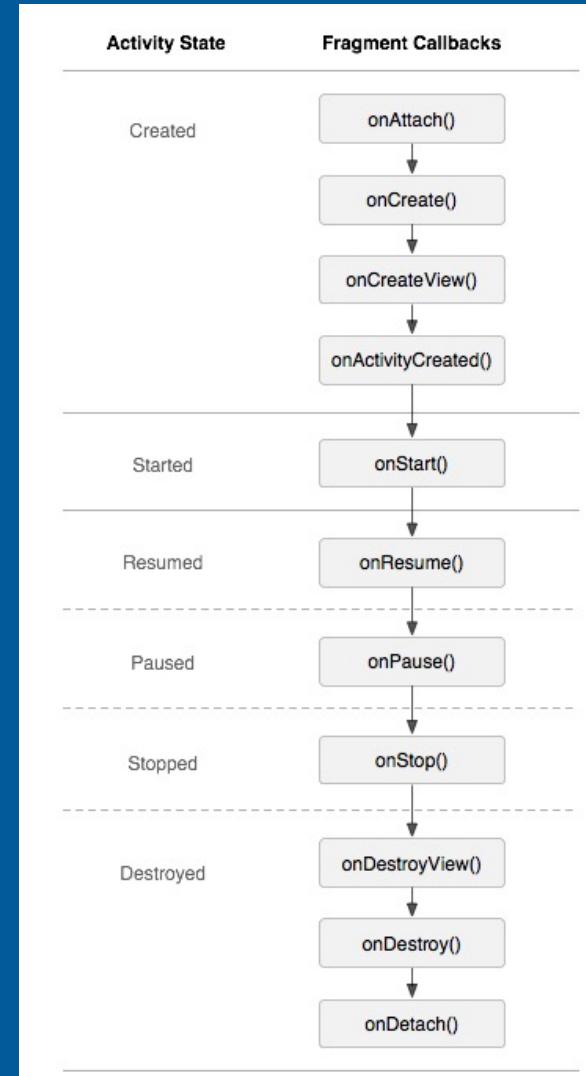
- Le fragment est visible dans l'activité en cours.

## 2. Paused

- Une autre activité est au premier plan et a le focus, mais l'activité dans laquelle ce fragment habite est toujours visible (l'activité au premier plan est partiellement transparente ou ne couvre pas tout l'écran).

## 3. Stopped

- Le fragment n'est pas visible. Soit l'activité de l'hôte a été arrêtée, soit le fragment a été retiré de l'activité mais ajouté à la pile arrière. Un fragment arrêté est toujours actif (toutes les informations sur l'état et les membres sont conservées par le système). Cependant, il n'est plus visible par l'utilisateur et est tué si l'activité est tuée.



# À quoi ressemble une classe Fragment() ?

```
class ExampleFragment : Fragment() {  
  
    override fun onCreateView(  
        inflater: LayoutInflater,  
        container: ViewGroup?,  
        savedInstanceState: Bundle?  
    ): View {  
        // Inflate the layout for this fragment  
        return inflater.inflate(R.layout.example_fragment, container, false)  
    }  
}
```

## La méthode inflate() prend trois arguments:

- 1) L'ID de ressource de la présentation que vous souhaitez gonfler.
- 2) Le ViewGroup doit être le parent de la disposition gonflée. Le passage du conteneur est important pour que le système applique les paramètres de présentation à la vue racine de la présentation gonflée, spécifiée par la vue parente dans laquelle il est inséré.
- 3) Un booléen indiquant si la présentation gonflée doit être attachée au ViewGroup (le deuxième paramètre) pendant l'inflation. (Dans ce cas, il s'agit de false, car le système insère déjà la présentation gonflée dans le conteneur. Si vous transmettez true, vous créez un groupe de vues redondant dans la présentation finale.)



# Ajouter un Fragment depuis un Layout

L'attribut **android:name** dans **<fragment>** spécifie la classe Fragment à instancier dans la présentation. Lorsque le système crée cette présentation d'activité, il instancie chaque fragment spécifié dans la présentation et appelle la méthode **onCreateView()** pour chacun d'eux, afin de récupérer la présentation de chaque fragment. Le système insère la vue renvoyée par le fragment directement à la place de l'élément **<fragment>**.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
        android:id="@+id/list"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
        android:id="@+id/viewer"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
</LinearLayout>
```



# Ajout d'un Fragment dans une ViewGroup en Kotlin

```
val fragmentManager = supportFragmentManager  
val fragmentTransaction = fragmentManager.beginTransaction()
```

```
val fragment = ExampleFragment()  
fragmentTransaction.add(R.id.fragment_container, fragment)  
fragmentTransaction.commit()
```

Remplacer le précédent Fragment de la ViewGroup, ne pas la mettre dans la BackStack :

```
val newFragment = ExampleFragment()  
val transaction = supportFragmentManager.beginTransaction()  
transaction.replace(R.id.fragment_container, newFragment)  
transaction.addToBackStack(null)  
transaction.commit()
```



# Appeler la méthode d'une Activity depuis un Fragment ?

```
(activity as YourActivityClassName).methodName()
```



# Navigation component

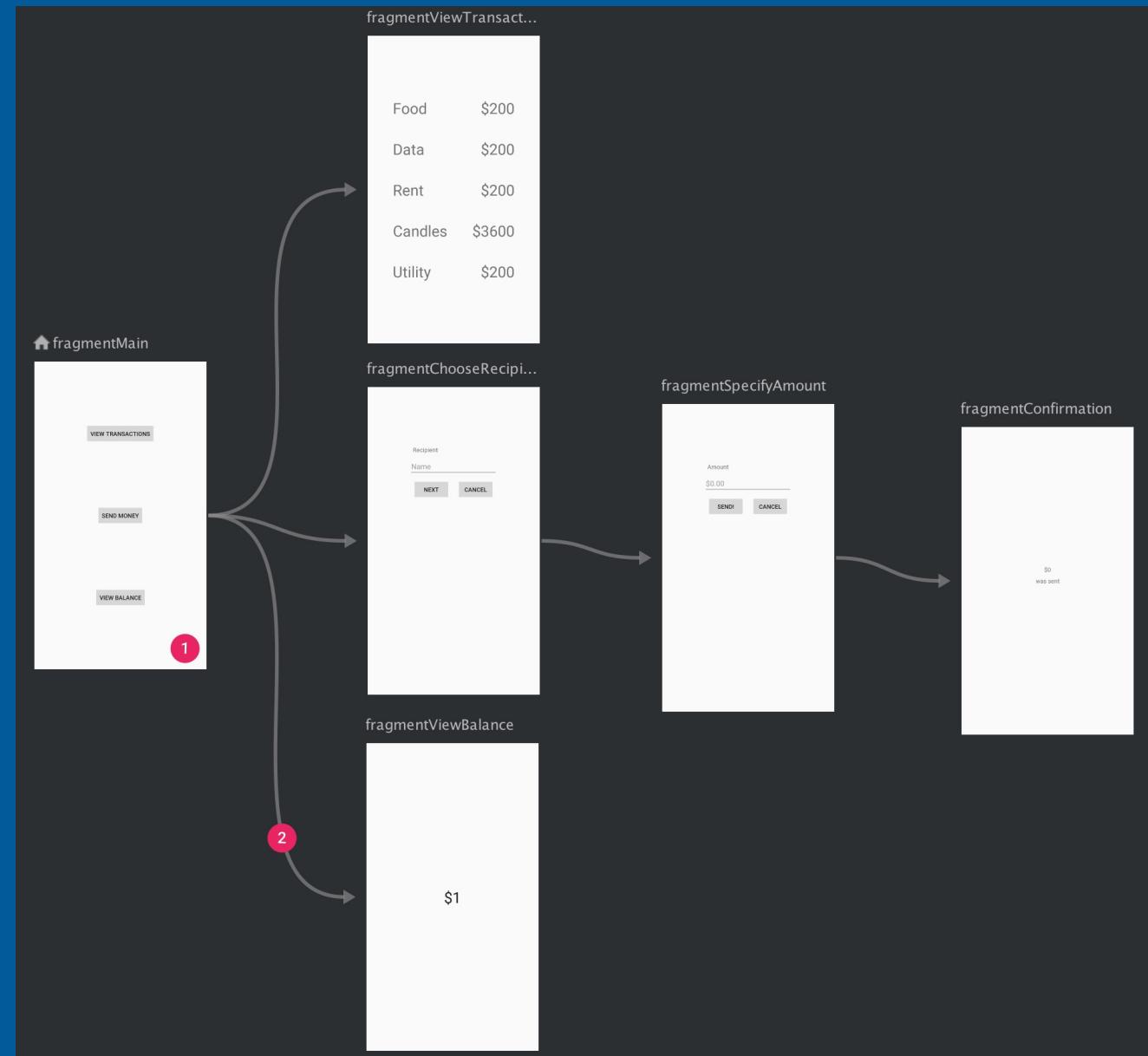
---

Naviguer dans son application



# Navigation ?

- La navigation fait référence aux interactions qui permettent aux utilisateurs de naviguer entre les différents éléments de contenu de votre application.
- Le composant de navigation d'Android Jetpack vous aide à implémenter la navigation, du simple clic de bouton aux modèles plus complexes, tels que les **app bars** ou **navigation drawer**.
- Le composant Navigation garantit également une expérience utilisateur cohérente et prévisible en adhérant à un ensemble de principes établis.



# Composition de Navigation component

- Le composant Navigation se compose de trois parties :
  - **Navigation graph**: ressource XML contenant toutes les informations relatives à la navigation dans un emplacement centralisé. Cela inclut toutes les zones de contenu individuelles de votre application, appelées **destinations**, ainsi que les chemins possibles qu'un utilisateur peut emprunter via votre application.
  - **NavHost**: Un conteneur vide qui affiche les destinations de votre graphique de navigation. Le composant Navigation contient une implementation de **NavHost** par défaut, **NavControllerFragment**, qui affiche les destinations des fragments.
  - **NavController**: Un objet qui gère la navigation des applications dans un **NavHost**. Le **NavController** orchestre l'échange de contenu de destination dans le **NavHost** au fur et à mesure que les utilisateurs se déplacent dans votre application.
- Lorsque vous parcourez votre application, vous indiquez **NavController** que vous souhaitez naviguer le long d'un chemin spécifique dans votre graphique de navigation ou directement vers une destination spécifique. Le **NavController** affiche alors la destination appropriée dans le **NavHost**.



# Avantages à utiliser Navigation component

- Le composant Navigation offre un certain nombre d'autres avantages, notamment les suivants:
  - Gestion des transactions de fragments.
  - Gérer correctement les actions Haut et Retour par défaut.
  - Fournir des ressources standardisées pour les animations et les transitions.
  - Implémentation et gestion des **deep link**.
  - Y compris les Navigation UI patterns, tels que les **navigation drawer** et **bottom navigation**, avec un minimum de travail supplémentaire.
  - **Safe Args** - un plugin Gradle qui offre une sécurité de type lors de la navigation et du transfert de données entre les destinations.
  - Prise en charge des **ViewModel** - vous pouvez étendre un **ViewModel** à un graphique de navigation pour partager des données liées à l'interface utilisateur entre les destinations du graphique.
- De plus, vous pouvez utiliser l'éditeur de navigation d'Android Studio pour afficher et modifier vos graphiques de navigation.



# Gradle de Projet

Classpath pour Safe Args :

```
def nav_version = "2.3.0-beta01"
classpath "androidx.navigation:navigation-safe-args-gradle-plugin:$nav_version"
```

# Gradle de Module

Java 8 :

```
compileOptions {  
    sourceCompatibility JavaVersion.VERSION_1_8  
    targetCompatibility JavaVersion.VERSION_1_8  
}  
  
kotlinOptions {  
    jvmTarget = JavaVersion.VERSION_1_8.toString()  
}
```

Libs :

```
def nav_version = "2.3.0-beta01"  
implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"  
implementation "androidx.navigation:navigation-ui-ktx:$nav_version"
```

Application du plugin pour Safe Args :

```
apply plugin: 'androidx.navigation.safeargs.kotlin'
```



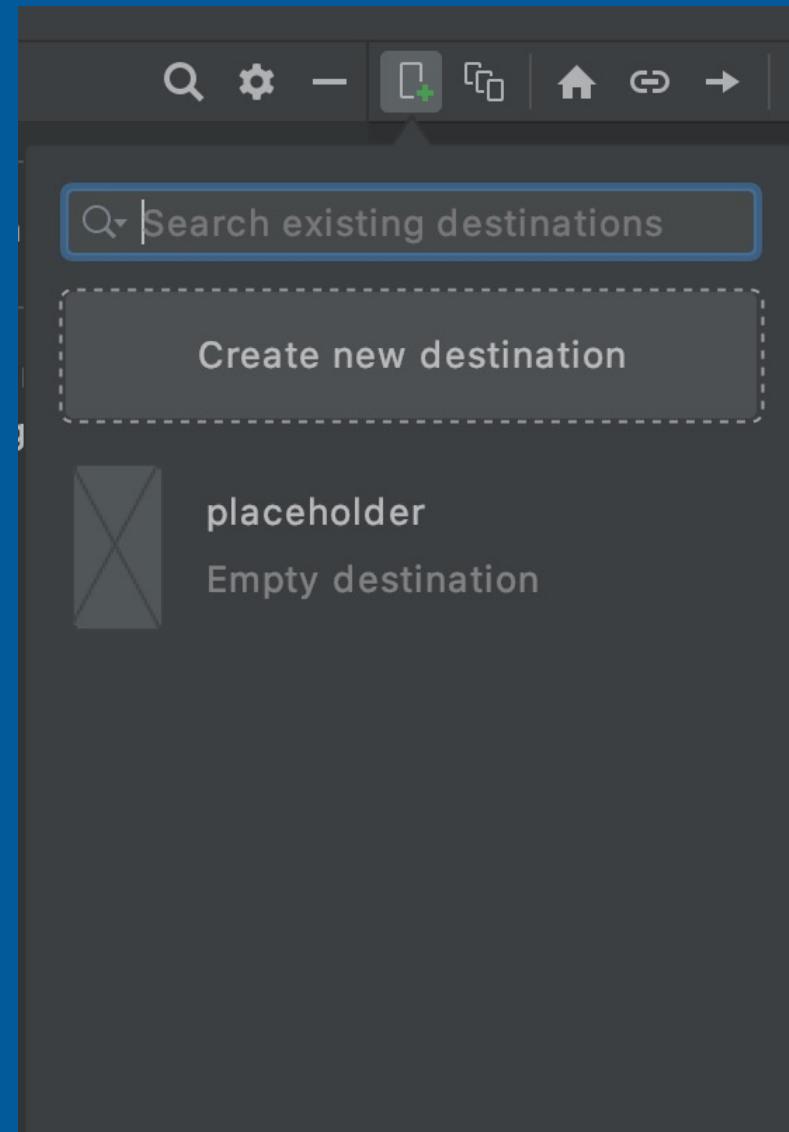
# Ajouter une ressource XML pour la navigation

- Click droit sur le dossier **res** puis **New > Android Resource File**
- Dans la fenêtre **New Resource File** :
  - Mettez dans **File name** un nom comme "nav\_graph".
  - Selectionnez **Navigation** dans **Resource type** puis cliquez sur **OK**.

# Nouvelle destination

- Cliquez sur l'icône **New Destination**
- Selectionnez **Create new destination**
- Créez un fragment vide (FirstFragment)

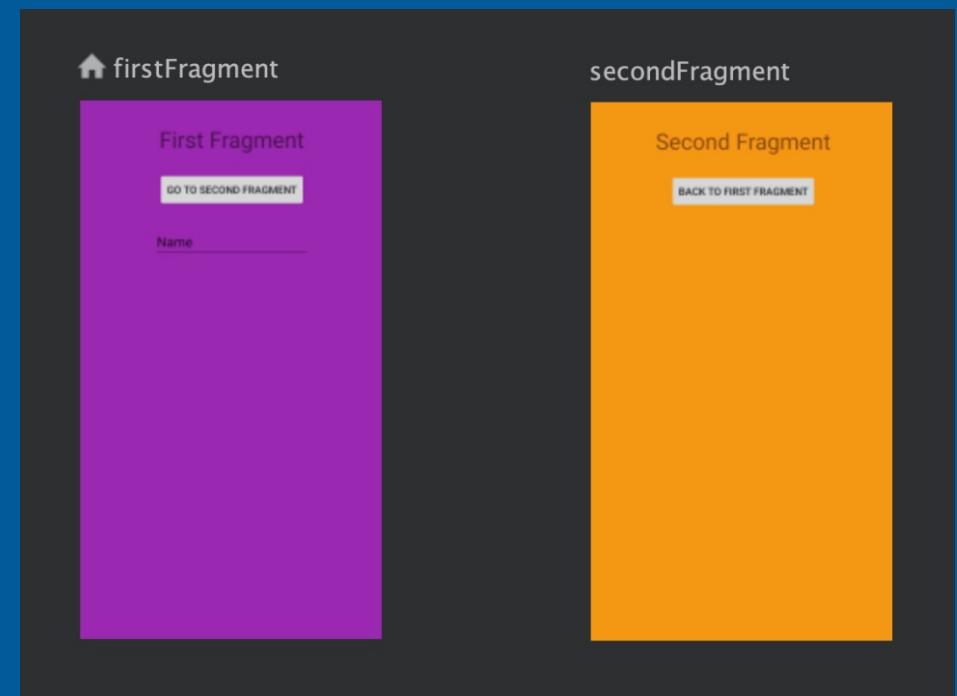
Réitérez l'opération une autre fois avec SecondFragment.



# Layout des fragments

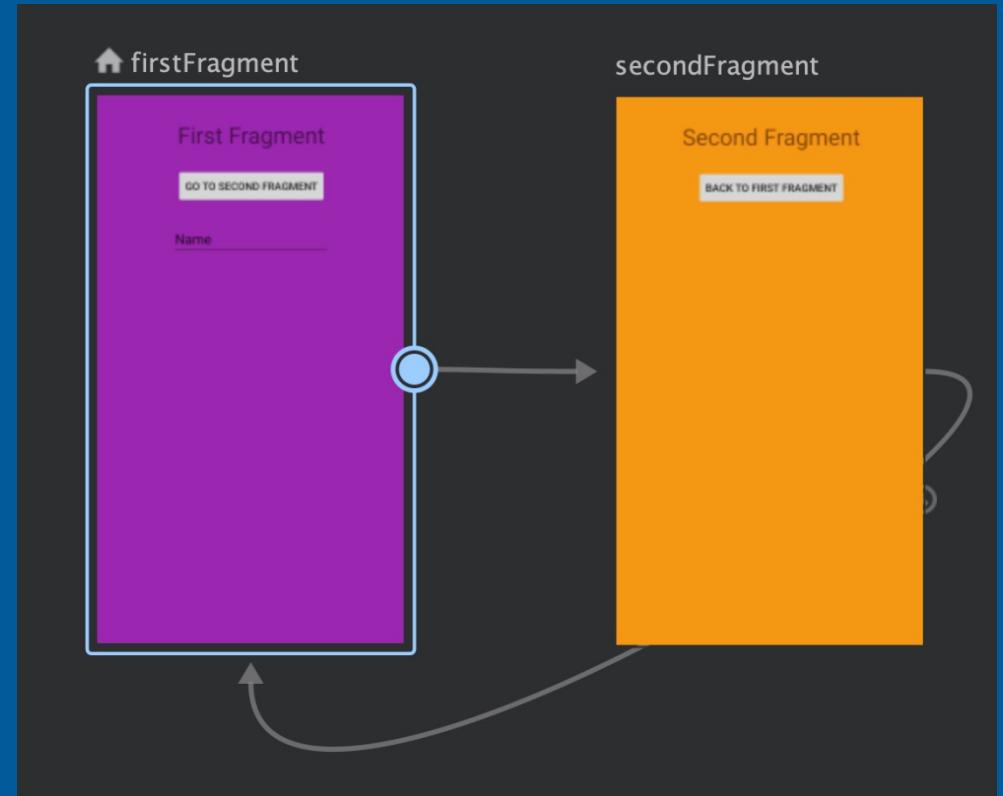
Vous devez avoir deux fragments :

- FirstFragment – placez dedans :
  - un **TextView** qui affiche « First Fragment »
  - un **Button** (*btnGoSecond*) qui affiche « Go to Second Fragment»
  - un **EditText** (*editName*) avec comme texte de fond « Name »
- SecondFragment – placez dedans :
  - un **TextView** qui affiche « Second Fragment »
  - un **Button** (*btnBackFirst*) qui affiche « Go to First Fragment»



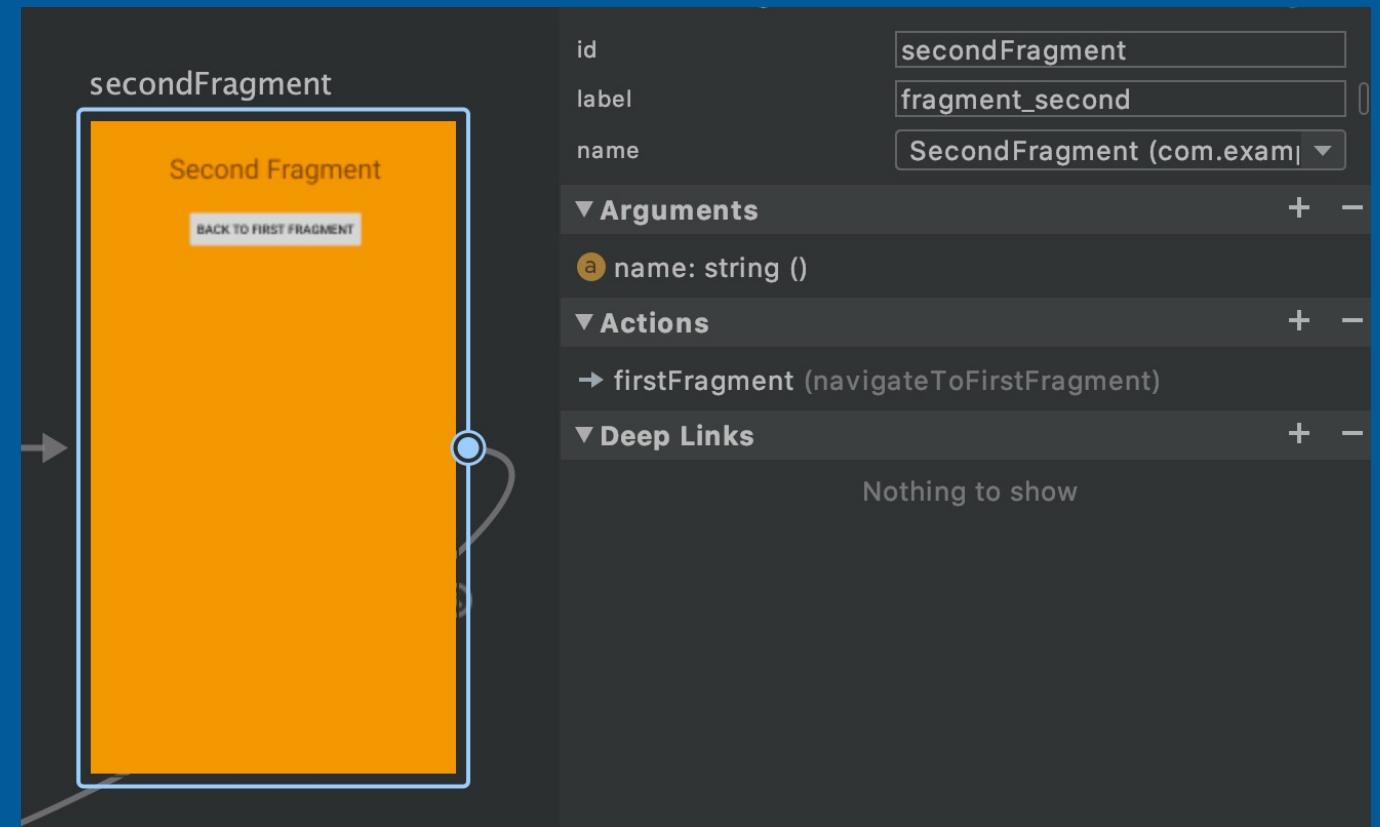
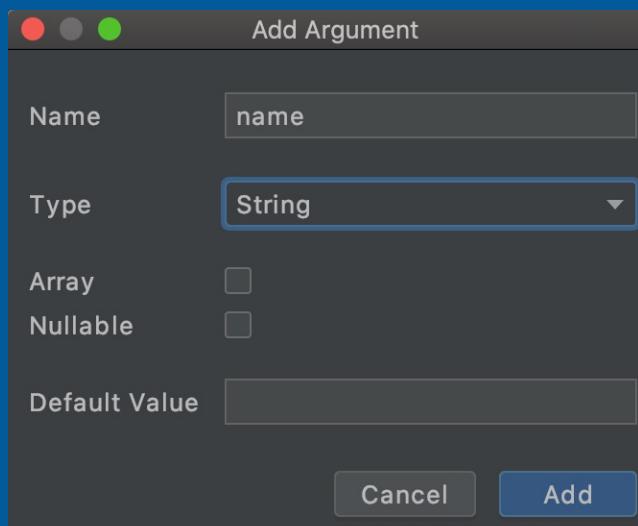
# Ajouter des actions

- Cliquez sur le point à droite de chaque Fragment de sorte à créer une flèche qui pointe vers l'autre Fragment.
- Ces flèches sont des actions. Si vous cliquez dessus, vous pouvez changer les ids. Mettez donc :
  - **navigateToSecondFragment** pour la flèche qui part de FirstFragment jusqu'à SecondFragment
  - **navigateToFirstFragment** pour la flèche qui part de SecondFragment jusqu'à FirstFragment



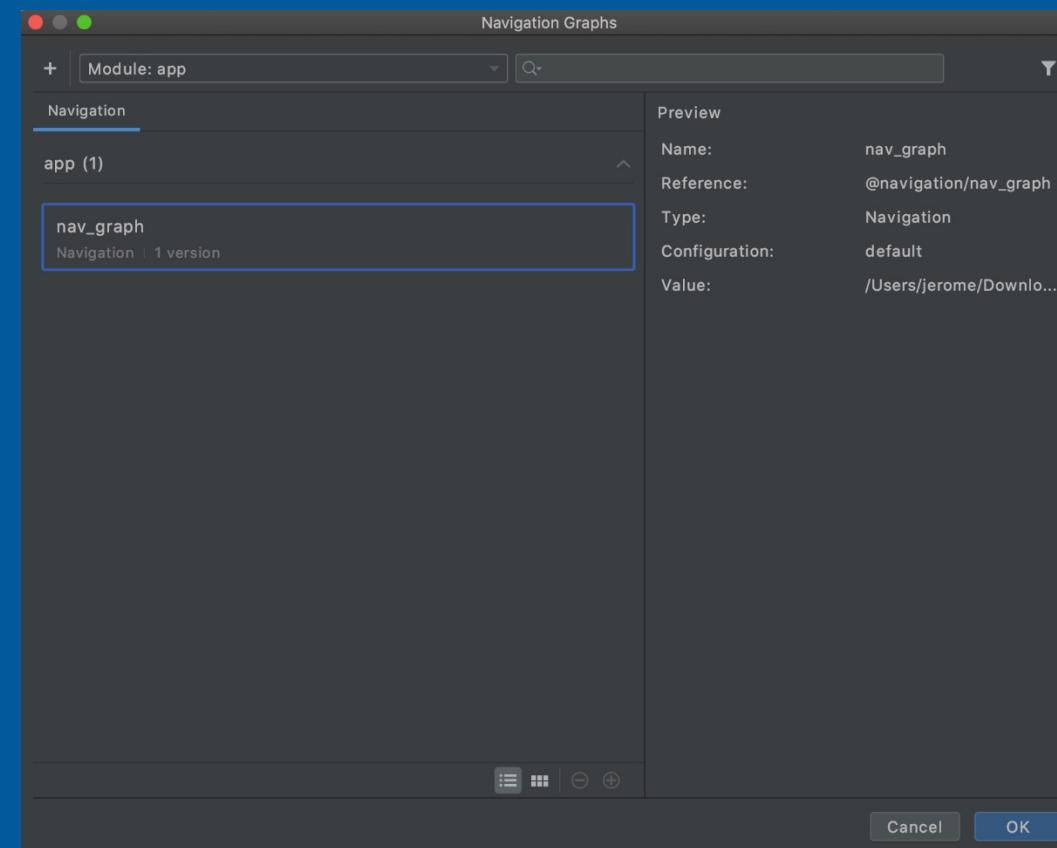
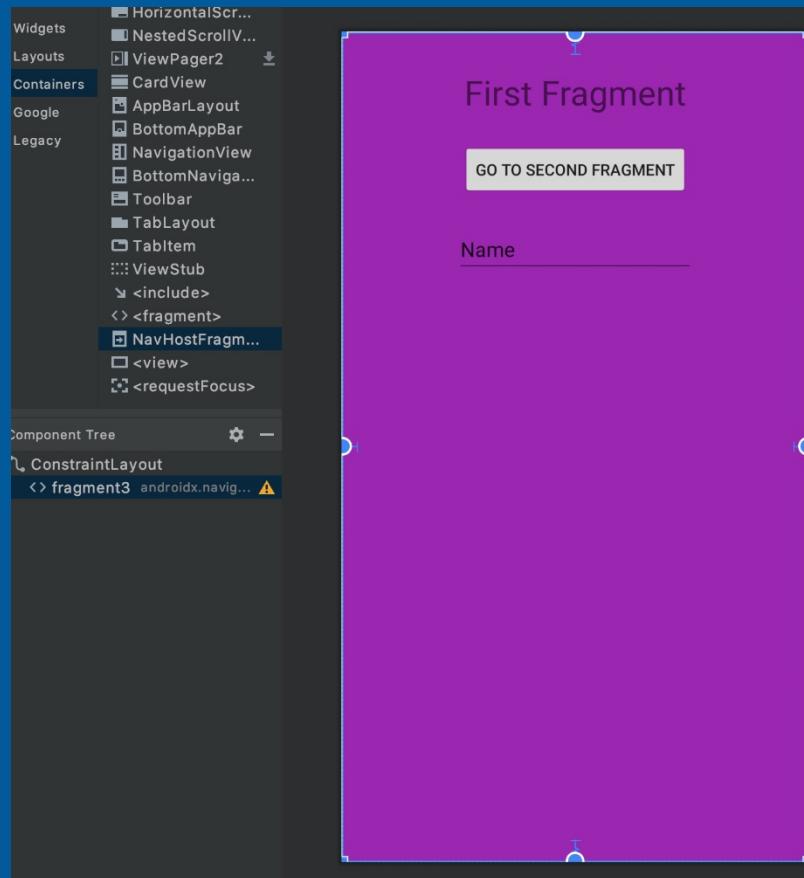
# Ajouter un argument

- Sur SecondFragment, ajoutez un argument name de type String, avec comme paramètre par défaut une chaîne de caractères vide.



# NavHostFragment

Ajoutez le widget NavHostFragment dans le layout de votre Activity :



# FirstFragment code

Dans la méthode **onViewCreated** : ajoutez ce code qui va permettre de lancer SecondFragment (avec l'argument name qui prendra comme valeur le contenu de l'EditText) quand on cliquera sur le boutton :

```
btnGoSecond.setOnClickListener { it: View!  
    val action : NavDirections = FirstFragmentDirections.navigateToSecondFragment(name = editText.text.toString())  
    view.findNavController().navigate(action)  
}
```

Il se peut que FirstFragmentDirections n'existe pas. C'est qu'il n'a pas été généré. Dans ce cas, un clean et build de votre projet devra faire l'affaire 😊



# SecondFragment code

On va récupérer tous les arguments dans notre classe SecondFragment de cette manière :

```
private val args: SecondFragmentArgs by navArgs()
```

Puis, dans la méthode onViewCreated, on affchera le name lié aux arguments et on gèrera le click pour un retour au FirstFragment :

```
Toast.makeText(context, text: "the value for argument name is ${args.name}", Toast.LENGTH_SHORT).show()

btnBackFirst.setOnClickListener { it: View!
    val action : NavDirections = SecondFragmentDirections.navigateToFirstFragment()
    view.findNavController().navigate(action)
}
```



# Architecture MVVM

---

Comment avoir une belle architecture dans son code ?



# Expérience utilisateur et applications mobiles

- Dans la majorité des cas, les applications de bureau ont un point d'entrée unique à partir d'un lanceur de bureau ou de programme, puis s'exécutent comme un processus monolithique unique.
- Les applications Android, en revanche, ont une structure beaucoup plus complexe. Une application Android typique contient plusieurs composants d'application, notamment des activités, fragments, services, fournisseurs de contenu et récepteurs de diffusion.
- Étant donné qu'une application Android contient plusieurs composants et que les utilisateurs interagissent souvent avec plusieurs applications dans un court laps de temps, les applications doivent s'adapter à différents types de workflows et de tâches pilotés par l'utilisateur.
- Exemple avec un partage de photo dans votre application :
  - L'application déclenche une intention de caméra -> le système d'exploitation Android lance une application d'appareil photo. À ce stade, l'utilisateur a quitté votre application, mais son expérience est toujours transparente.
  - L'application appareil photo peut déclencher d'autres intentions, comme le lancement du sélecteur de fichiers, qui peut lancer une autre application. Finalement, l'utilisateur retourne à l'application de réseautage social et partage la photo.
  - À tout moment du processus, l'utilisateur peut être interrompu par un appel téléphonique ou une notification. Après avoir agi sur cette interruption, l'utilisateur s'attend à pouvoir reprendre et reprendre ce processus de partage de photos. Ce comportement de saut d'application est courant sur les appareils mobiles, votre application doit donc gérer correctement ces flux.
- Gardez à l'esprit que les appareils mobiles sont également limités en ressources, donc à tout moment, le système d'exploitation peut tuer certains processus d'application pour faire de la place pour de nouveaux.
- Compte tenu des conditions de cet environnement, il est possible que les composants de votre application soient lancés individuellement et hors service, et le système d'exploitation ou l'utilisateur peut les détruire à tout moment. Étant donné que ces événements ne sont pas sous votre contrôle, vous ne devez pas stocker de données ni d'état d'application dans vos composants d'application, et vos composants d'application ne doivent pas dépendre les uns des autres.



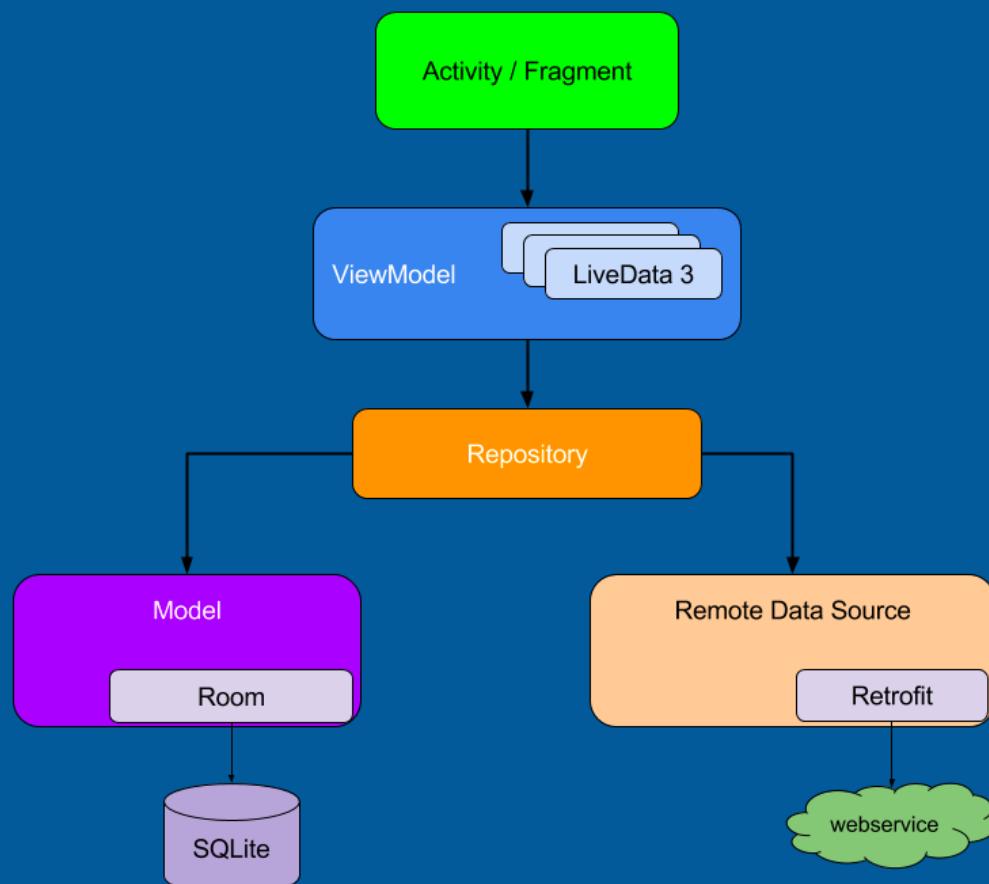
# Principes architecturaux communs

- Si vous ne devez pas utiliser de composants d'application pour stocker les données et l'état de l'application, comment devez-vous concevoir votre application?
- Le principe le plus important à suivre est la séparation des préoccupations. C'est une erreur courante d'écrire tout votre code dans une activité ou un fragment. Ces classes basées sur l'interface utilisateur ne doivent contenir que la logique qui gère les interactions de l'interface utilisateur et du système d'exploitation. En gardant ces classes aussi simples que possible, vous pouvez éviter de nombreux problèmes liés au cycle de vie.
- Gardez à l'esprit que vous ne possédez pas d'implémentations d'Activity and Fragment. Le système d'exploitation peut les détruire à tout moment en fonction des interactions des utilisateurs ou en raison de conditions système telles que la mémoire insuffisante. Pour fournir une expérience utilisateur satisfaisante et une expérience de maintenance d'application plus gérable, il est préférable de minimiser votre dépendance à leur égard.
- Un autre principe important est que vous devez piloter votre interface utilisateur à partir d'un modèle, de préférence un modèle persistant. Les modèles sont des composants chargés de gérer les données d'une application. Ils sont indépendants des objets View et des composants d'application de votre application, ils ne sont donc pas affectés par le cycle de vie de l'application et les problèmes associés.
- La persistance est idéale pour les raisons suivantes:
  - Vos utilisateurs ne perdent pas de données si le système d'exploitation Android détruit votre application pour libérer des ressources.
  - Votre application continue de fonctionner dans les cas où une connexion réseau est instable ou indisponible.
- En basant votre application sur des classes de modèle avec la responsabilité bien définie de gérer les données, votre application est plus testable et cohérente.



# Architecture d'application recommandée

- Imaginez que nous construisons une interface utilisateur qui affiche un profil utilisateur. Nous utilisons un backend privé et une API REST pour récupérer les données d'un profil donné.
- Pour commencer, considérez le diagramme suivant, qui montre comment tous les modules doivent interagir les uns avec les autres après la conception de l'application:

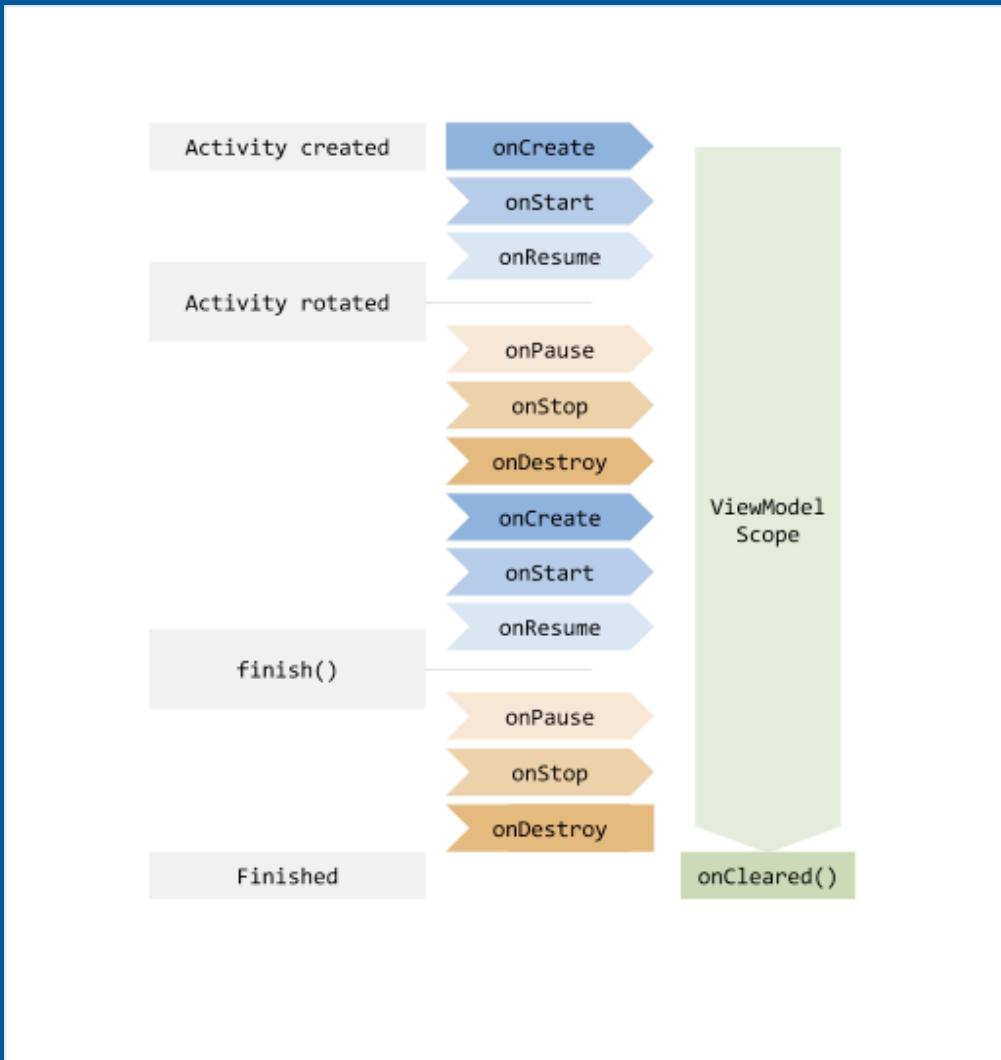


Notez que chaque composant dépend uniquement du composant un niveau en dessous. Par exemple, les activités et les fragments dépendent uniquement d'un modèle de vue. Le référentiel est la seule classe qui dépend de plusieurs autres classes; dans cet exemple, le référentiel dépend d'un modèle de données persistant et d'une source de données backend distante.

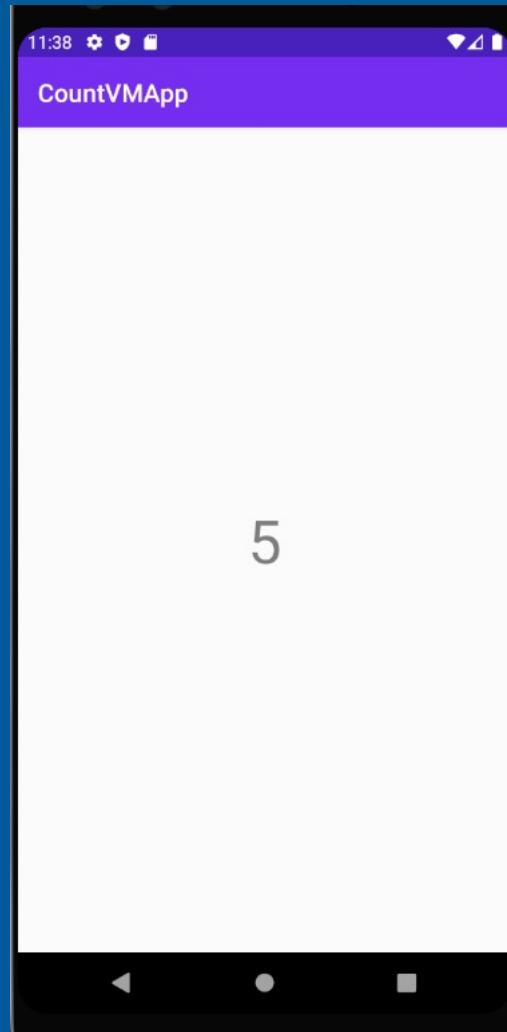
Cette conception crée une expérience utilisateur cohérente et agréable. Que l'utilisateur revienne à l'application plusieurs minutes après sa dernière fermeture ou plusieurs jours plus tard, il voit instantanément les informations d'un utilisateur indiquant que l'application persiste localement. Si ces données sont périmées, le module de référentiel de l'application commence à mettre à jour les données en arrière-plan.



# ViewModel



# Exemple



# Lifecycle extension

```
implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
```



# Layout

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

    <TextView
        android:id="@+id/my_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="48sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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



# ViewModel

```
class MyViewModel : ViewModel(), LifecycleObserver {

    private var count: Int = 0
    val changeNotifier = MutableLiveData<Int>()

    fun increment() { changeNotifier.value = ++count }

    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME) fun onResume() { increment() }
}
```



# MainActivity

```
class MainActivity : AppCompatActivity() {

    private val viewModel: MyViewModel by lazy {
        ViewModelProvider(owner: this).get(MyViewModel::class.java)
    }

    private val changeObserver = Observer<Int> { value -> value?.let { incrementCount(value) } }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        lifecycle.addObserver(viewModel)
        viewModel.changeNotifier.observe(owner: this, changeObserver)
        my_container.setOnClickListener { viewModel.increment() }
    }

    private fun incrementCount(value: Int) {
        my_text.text = (value).toString()
    }
}
```



# Les permissions

Le but d'une autorisation est de protéger la vie privée d'un utilisateur Android. Les applications Android doivent demander l'autorisation d'accéder aux données utilisateur sensibles (telles que les contacts et les SMS), ainsi qu'à certaines fonctionnalités du système (telles que l'appareil photo et Internet). Selon la fonctionnalité, le système peut accorder l'autorisation automatiquement ou inviter l'utilisateur à approuver la demande.



# Qu'est-ce qu'un permission

- Une application mobile, quelque soit le système d'exploitation sur lequel elle tourne, peut potentiellement faire des milliers de choses. A titre d'exemple, une application peut :
  - envoyer / afficher des SMS
  - appeler
  - surfer sur internet
  - prendre des photos
  - visionner des photos
  - programmer un réveil
  - etc.



# Android 6 et les permissions

- À partir d'Android 6 (API 23) l'utilisateur donne les permissions au moment où l'application utilise la fonctionnalité cible.
  - Par exemple, un utilisateur peut donner accès au GPS à une application mais refuser de donner l'accès à la liste des contacts.
  - L'utilisateur peut aussi autoriser / supprimer des permissions dans l'écran dédié à cet effet dans les paramètres.



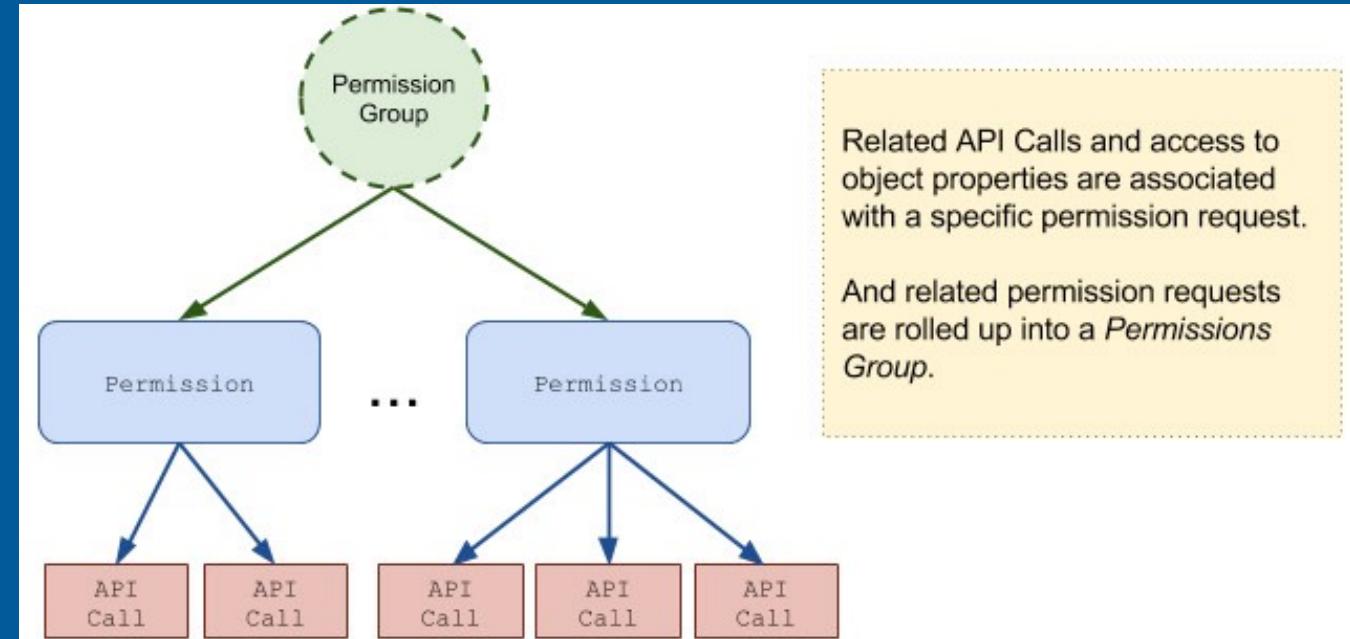
# Quelques conseils

- Ne demander une permission que si elle est nécessaire
- Expliquer à l'utilisateur pourquoi votre application nécessite cette permission
- Ne pas harceler l'utilisateur pour qu'il vous donne la permission
- Accepter la décision de l'utilisateur



# Groupes de permissions

Permission Group	Permissions
CALENDAR	<ul style="list-style-type: none"><li>• READ_CALENDAR</li><li>• WRITE_CALENDAR</li></ul>
CALL_LOG	<ul style="list-style-type: none"><li>• READ_CALL_LOG</li><li>• WRITE_CALL_LOG</li><li>• PROCESS_OUTGOING_CALLS</li></ul>
CAMERA	<ul style="list-style-type: none"><li>• CAMERA</li></ul>
CONTACTS	<ul style="list-style-type: none"><li>• READ_CONTACTS</li><li>• WRITE_CONTACTS</li><li>• GET_ACCOUNTS</li></ul>
LOCATION	<ul style="list-style-type: none"><li>• ACCESS_FINE_LOCATION</li><li>• ACCESS_COARSE_LOCATION</li></ul>
MICROPHONE	<ul style="list-style-type: none"><li>• RECORD_AUDIO</li></ul>
PHONE	<ul style="list-style-type: none"><li>• READ_PHONE_STATE</li><li>• READ_PHONE_NUMBERS</li><li>• CALL_PHONE</li><li>• ANSWER_PHONE_CALLS</li><li>• ADD_VOICEMAIL</li><li>• USE_SIP</li></ul>
SENSORS	<ul style="list-style-type: none"><li>• BODY_SENSORS</li></ul>
SMS	<ul style="list-style-type: none"><li>• SEND_SMS</li><li>• RECEIVE_SMS</li><li>• READ_SMS</li><li>• RECEIVE_WAP_PUSH</li><li>• RECEIVE_MMS</li></ul>
STORAGE	<ul style="list-style-type: none"><li>• READ_EXTERNAL_STORAGE</li><li>• WRITE_EXTERNAL_STORAGE</li></ul>



Related API Calls and access to object properties are associated with a specific permission request.

And related permission requests are rolled up into a *Permissions Group*.



# Déclarer une permission dans le Manifest

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.example.snazzyapp">  
  
    <uses-permission android:name="android.permission.SEND_SMS"/>  
  
    <application ...>  
        ...  
    </application>  
</manifest>
```



# Vérifier une permission

- À chaque utilisation d'une fonctionnalité nécessitant une permission, il faut vérifier si la permission est donnée ou non
- Pour cela, on utilise la méthode **checkSelfPermission** qui nous renvoie :
  - **PackageManager.PERMISSION\_GRANTED** lorsqu'on a la permission
  - **PackageManager.PERMISSION\_DENIED** lorsqu'on n'a pas la permission

```
val permissionCheck : Int = ContextCompat.checkSelfPermission(  
    applicationContext,  
    android.Manifest.permission.WRITE_CALENDAR)  
  
if (permissionCheck == PackageManager.PERMISSION_GRANTED)  
    Log.d( tag: "debugApp", msg: "j'ai le droit d'écrire sur le calendrier")  
else  
    Log.d( tag: "debugApp", msg: "je n'ai pas le droit d'écrire sur le calendrier")
```



# Vérifier si la permission a déjà été demandée

La méthode **shouldShowRequestPermissionRationale** permet de savoir si la permission à déjà été demandée à l'utilisateur.

Si c'est le cas et que la permission a été refusée, mieux vaut ne pas redemander la permission à l'utilisateur.

```
if (ActivityCompat.shouldShowRequestPermissionRationale( activity: this, android.Manifest.permission.READ_CONTACTS))  
    Log.d( tag: "myDebug", msg: "La permission a déjà été demandée et refusée")
```



# Demander une permission à l'utilisateur

Vous pouvez demander une permission à l'aide de la méthode **requestPermissions()**

```
ActivityCompat.requestPermissions(thisActivity,  
    arrayOf(Manifest.permission.READ_CONTACTS),  
    MY_PERMISSIONS_REQUEST_READ_CONTACTS)
```



# Exemple

```
// Here, thisActivity is the current activity
if (ContextCompat.checkSelfPermission(thisActivity,
    Manifest.permission.READ_CONTACTS)
!= PackageManager.PERMISSION_GRANTED) {

    // Permission is not granted
    // Should we show an explanation?
    if (ActivityCompat.shouldShowRequestPermissionRationale(thisActivity,
        Manifest.permission.READ_CONTACTS)) {
        // Show an explanation to the user *asynchronously* -- don't block
        // this thread waiting for the user's response! After the user
        // sees the explanation, try again to request the permission.
    } else {
        // No explanation needed, we can request the permission.
        ActivityCompat.requestPermissions(thisActivity,
            arrayOf(Manifest.permission.READ_CONTACTS),
            MY_PERMISSIONS_REQUEST_READ_CONTACTS)

        // MY_PERMISSIONS_REQUEST_READ_CONTACTS is an
        // app-defined int constant. The callback method gets the
        // result of the request.
    }
} else {
    // Permission has already been granted
}
```



# Connaitre la réponse de l'utilisateur

Une fois la permission demandée, le résultat de la demande sera retourné dans la méthode `onRequestPermissionsResult` (à surcharger).

```
override fun onRequestPermissionsResult(requestCode: Int,
    permissions: Array<String>, grantResults: IntArray) {
    when (requestCode) {
        MY_PERMISSIONS_REQUEST_READ_CONTACTS -> {
            // If request is cancelled, the result arrays are empty.
            if ((grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED))
                // permission was granted, yay! Do the
                // contacts-related task you need to do.
            } else {
                // permission denied, boo! Disable the
                // functionality that depends on this permission.
            }
            return
        }

        // Add other 'when' lines to check for other
        // permissions this app might request.
    else -> {
        // Ignore all other requests.
    }
    }
}
```



# Les librairies

Le but d'une librairie est de ne pas réinventer la roue.



# Les librairies sur Android

- En dev on a tendance à perdre du temps à implémenter des fonctionnalités qui, avec un peu de recherche sur la toile, nous aurait permis d'aller plus vite dans notre développement tout en ayant du code propre et maintenable.
- Le développement avec Android ne déroge pas à la règle. On peut trouver des nombreuses libs qui permettent de nous faciliter la vie en :
  - nous permettant d'aller plus vite dans notre dev
  - étant plus focus sur le cœur même de l'application



- AndroidX est le projet open-source utilisé par l'équipe Android pour développer, tester, créer des packages, créer des versions et créer des bibliothèques dans Jetpack.
- AndroidX constitue une amélioration majeure de la bibliothèque de support Android d'origine. À l'instar de la bibliothèque de support, AndroidX est livré séparément du système d'exploitation Android et offre une compatibilité ascendante entre les versions Android. AndroidX remplace totalement la bibliothèque de support en fournissant une parité des fonctionnalités et de nouvelles bibliothèques.
- De plus, AndroidX inclut les fonctionnalités suivantes:
  - Tous les packages sous AndroidX résident dans un espace de noms cohérent commençant par la chaîne `androidx`. Les packages de la bibliothèque de support ont été mappés dans les packages `androidx.*` Correspondants. Pour un mappage complet de toutes les anciennes classes et créer des artefacts pour les nouveaux, voir la page Refactoring de paquet
  - Contrairement à la bibliothèque de support, les packages AndroidX sont gérés et mis à jour séparément. Les packages `androidx` utilisent la version sémantique stricte à partir de la version 1.0.0. Vous pouvez mettre à jour les bibliothèques AndroidX de votre projet indépendamment.
  - Tous les nouveaux développements de la bibliothèque de support auront lieu dans la bibliothèque AndroidX. Cela inclut la maintenance des artefacts de la bibliothèque de support d'origine et l'introduction de nouveaux composants Jetpack.



<https://www.android-arsenal.com/>

Un répertoire catégorisé des bibliothèques et des outils pour Android.

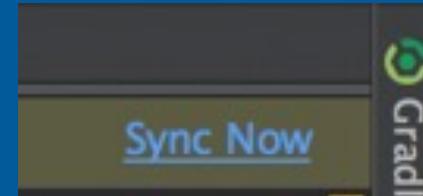
The screenshot displays the Android Arsenal website, which is a curated list of open-source libraries and tools for Android development. The page features a header with navigation links like 'Home', 'Tools', 'Libraries', 'Articles', 'About', and 'Contact'. Below the header, there's a search bar and a 'Recent' section. The main content area is organized into several columns, each containing a library or tool card. Each card includes the library name, developer, license status (e.g., 'Free', 'Paid'), and a brief description. Some cards also feature small illustrations or screenshots. The cards are arranged in a grid-like fashion, with some cards having larger descriptions or images than others. The overall layout is clean and professional, designed to help developers quickly find the right tools for their projects.



# Installation d'une librairie sur Android

```
dependencies {  
    implementation fileTree(dir: 'libs', include: ['*.jar'])  
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"  
    implementation 'androidx.appcompat:appcompat:1.0.2'  
    implementation 'androidx.core:core-ktx:1.0.2'  
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'  
    implementation 'com.google.android.material:material:1.1.0-alpha07'  
    testImplementation 'junit:junit:4.12'  
    androidTestImplementation 'androidx.test:runner:1.2.0'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'  
}
```

Ne pas oublier de synchroniser le projet après :



# Utilisation d'une librairie sur Android

## Usage

```
<de.hdodenhof.circleimageview.CircleImageView  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    android:id="@+id/profile_image"  
    android:layout_width="96dp"  
    android:layout_height="96dp"  
    android:src="@drawable/profile"  
    app:civ_border_width="2dp"  
    app:civ_border_color="#FF000000"/>
```

```
1 <?xml version="1.0" encoding="utf-8"?>  
2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
3     xmlns:tools="http://schemas.android.com/tools"  
4     android:id="@+id/activity_main"  
5     android:layout_width="match_parent"  
6     android:layout_height="match_parent"  
7     android:gravity="center"  
8     tools:context="teamgeny.imagewithlibrary.MainActivity">  
9  
10    <de.hdodenhof.circleimageview.CircleImageView  
11        xmlns:app="http://schemas.android.com/apk/res-auto"  
12        android:id="@+id/profile_image"  
13        android:layout_width="300dp"  
14        android:layout_height="300dp"  
15        android:src="@drawable/trololo"  
16        app:civ_border_width="2dp"  
17        app:civ_border_color="#FF000000"/>  
18  
19    </RelativeLayout>  
20
```

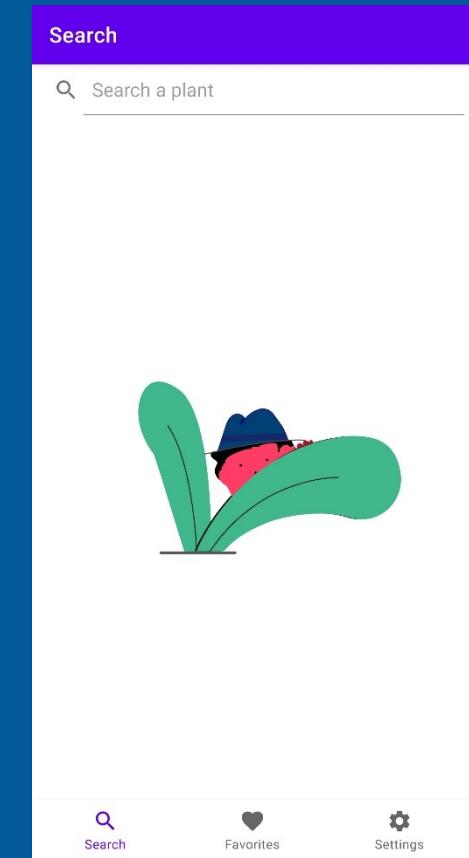


# TP – iPlant (5/10)

## Utilisation de la librairie Lottie



- Vous allez implémenter la librairie Lottie. Elle permet d'afficher des animation vectorielles sous format Json.
  - Vous pouvez trouver la librairie ici : [github.com/airbnb/lottie-android](https://github.com/airbnb/lottie-android)
- Ajouter une animation au milieu de l'écran de SearchFragment.
  - Vous pouvez trouver des animations ici : [lottiefiles.com](https://lottiefiles.com)
- *Dans un prochain TP, vous devrez la faire disparaître lorsque l'utilisateur fera une recherche.*



# SharedPreferences

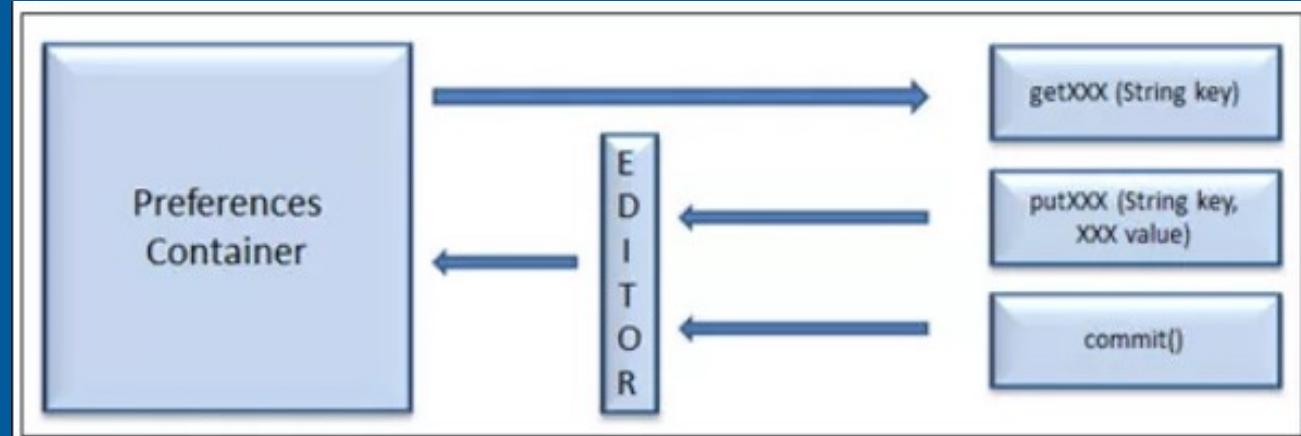
---

Stocker les préférences de son app facilement.



# Les SharedPreferences c'est quoi ?

Nested classes	
interface	<code>SharedPreferences.Editor</code>
	Interface used for modifying values in a <code>SharedPreferences</code> object.
Public methods	
abstract boolean	<code>contains(String key)</code> Checks whether the preferences contains a preference.
abstract <code>SharedPreferences.Editor</code>	<code>edit()</code> Create a new Editor for these preferences, through which you can make modifications to the data in the preferences and atomically commit those changes back to the <code>SharedPreferences</code> object.
abstract <code>Map&lt;String, ?&gt;</code>	<code>getAll()</code> Retrieve all values from the preferences.
abstract boolean	<code>getBoolean(String key, boolean defValue)</code> Retrieve a boolean value from the preferences.
abstract float	<code>getFloat(String key, float defValue)</code> Retrieve a float value from the preferences.
abstract int	<code>getInt(String key, int defValue)</code> Retrieve an int value from the preferences.
abstract long	<code>getLong(String key, long defValue)</code> Retrieve a long value from the preferences.
abstract String	<code>getString(String key, String defValue)</code> Retrieve a String value from the preferences.
abstract <code>Set&lt;String&gt;</code>	<code>getStringSet(String key, Set&lt;String&gt; defValues)</code> Retrieve a set of String values from the preferences.
abstract void	<code>registerOnSharedPreferenceChangeListener(SharedPreferences.OnSharedPreferenceChangeListener listener)</code> Registers a callback to be invoked when a change happens to a preference.
abstract void	<code>unregisterOnSharedPreferenceChangeListener(SharedPreferences.OnSharedPreferenceChangeListener listener)</code> Unregisters a previous callback.



# Exemple

```
val mesPrefs : SharedPreferences!
    = getSharedPreferences( name: "PREFS", Context.MODE_PRIVATE)

// Récupération de prefs
val age : Int = mesPrefs.getInt("age", 0)
val name : String? = mesPrefs.getString("name", "")
val nightMode : Boolean = mesPrefs.getBoolean("night_mode", false)
Log.d( tag: "mesLogs", msg: "age $age / name $name / night_mode $nightMode")

// Stockage de prefs
mesPrefs.edit().apply { this: SharedPreferences.Editor!
    putInt("age", 42)
    putString("name", "Jean")
    putBoolean("night_mode", true)
    apply()
}
```



# Différents modes d'accessibilité

- **Context.MODE\_PRIVATE**
  - pour que le fichier créé ne soit accessible que par l'application qui l'a créé.
- **Context.MODE\_WORLD\_READABLE**
  - pour que le fichier créé puisse être lu par n'importe quelle application.
- **Context.MODE\_WORLD\_WRITEABLE**
  - pour que le fichier créé puisse être lu *et* modifié par n'importe quelle application.



# Types valides

- Les préférences partagées ne fonctionnent qu'avec les objets de type :
  - Boolean
  - Float
  - Int
  - Long
  - String

# Un lib sympa : KotPref

```
object UserInfo : KotprefModel() {
    var gameLevel by enumValuePref(GameLevel.NORMAL)
    var name by stringPref()
    var code by nullableStringPref()
    var age by intPref(default = 14)
    var highScore by longPref()
    var rate by floatPref()
    val prizes by stringSetPref {
        val set = TreeSet<String>()
        set.add("Beginner")
        return@stringSetPref set
    }
}

enum class GameLevel {
    EASY,
    NORMAL,
    HARD
}
```

```
UserInfo.gameLevel = GameLevel.HARD
UserInfo.name = "chibatching"
UserInfo.code = "DAEF2599-7FC9-49C5-9A11-3C12B14A6898"
UserInfo.age = 30
UserInfo.highScore = 49219902
UserInfo.rate = 49.21F
UserInfo.prizes.add("Bronze")
UserInfo.prizes.add("Silver")
UserInfo.prizes.add("Gold")

Log.d(TAG, "Game level: ${UserInfo.gameLevel}")
Log.d(TAG, "User name: ${UserInfo.name}")
Log.d(TAG, "User code: ${UserInfo.code}")
Log.d(TAG, "User age: ${UserInfo.age}")
Log.d(TAG, "User high score: ${UserInfo.highScore}")
Log.d(TAG, "User rate: ${UserInfo.rate}")
UserInfo.prizes.forEachIndexed { i, s -> Log.d(TAG, "prize[$i]: ${s}") }
```



# TP – iPlant (6/10)



## Stocker des préférences

- En utilisant les SharedPreferences, faites en sorte que vos choix dans la partie des préférences soient sauvegardés :
  - Activate night mode
  - Hide synonyms in the details page
- Vous pouvez utiliser une librairie si vous le souhaitez.



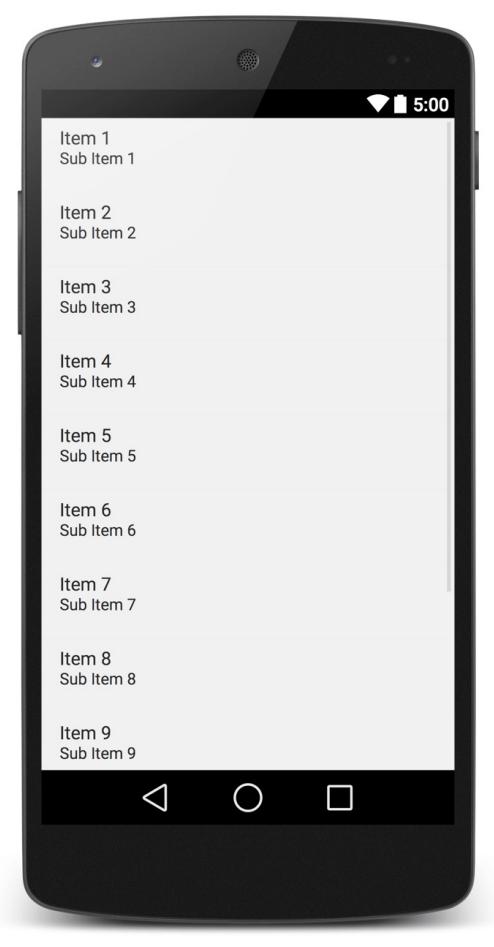
# Les listes avec RecyclerView

---

Afficher des listes propres et scroolables à l'inifini et de manière fluide!

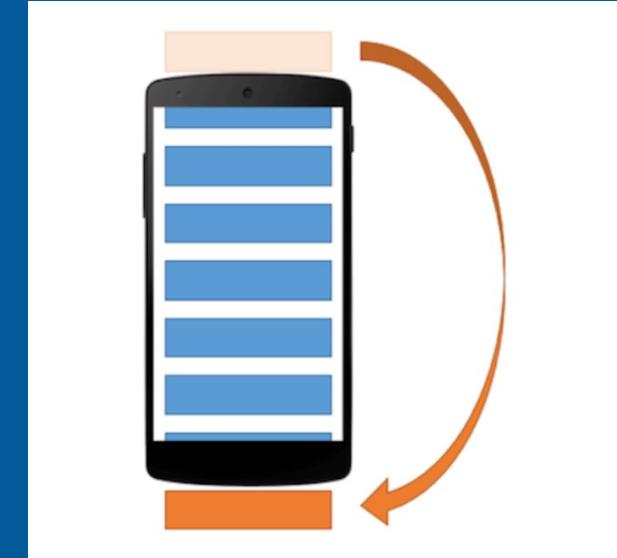


# Afficher une liste



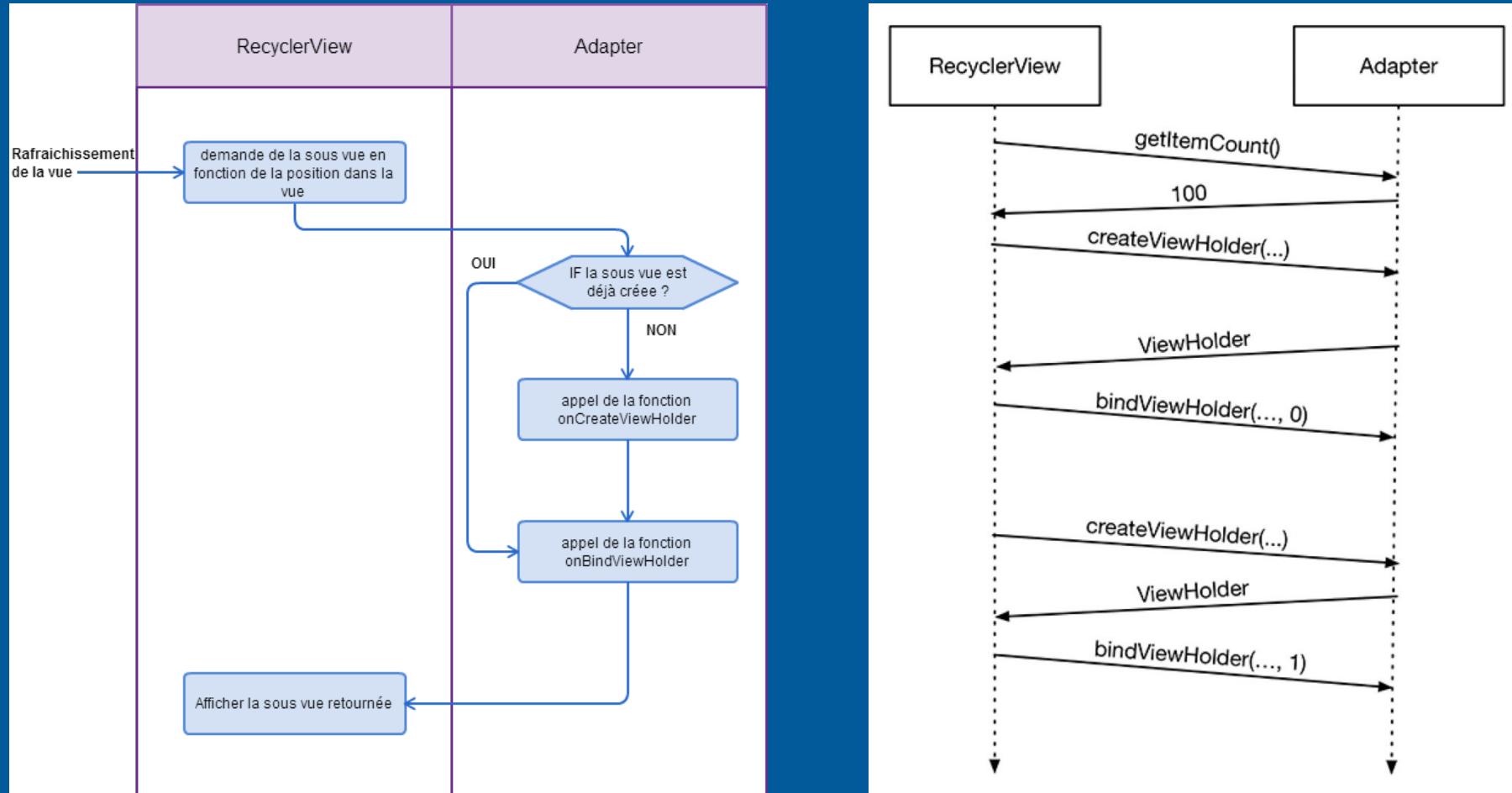
# RecyclerView > ListView ?

- Dans une ListView, il y aura autant de vues créées que d'items... C'est lourd !
- Dans une RecyclerView, il y aura autant de vues créées que d'items visibles à l'écran. Lorsque que l'on fait défiler les vues et que l'une d'entre elles disparaît, elle est "recyclée" pour être réutilisée.
- Il est possible de choisir son **LayoutManager** avec les **RecyclerView** :
  - **LinearLayoutManager**
  - **GridLayoutManager**
  - **StaggeredGridLayoutManager**

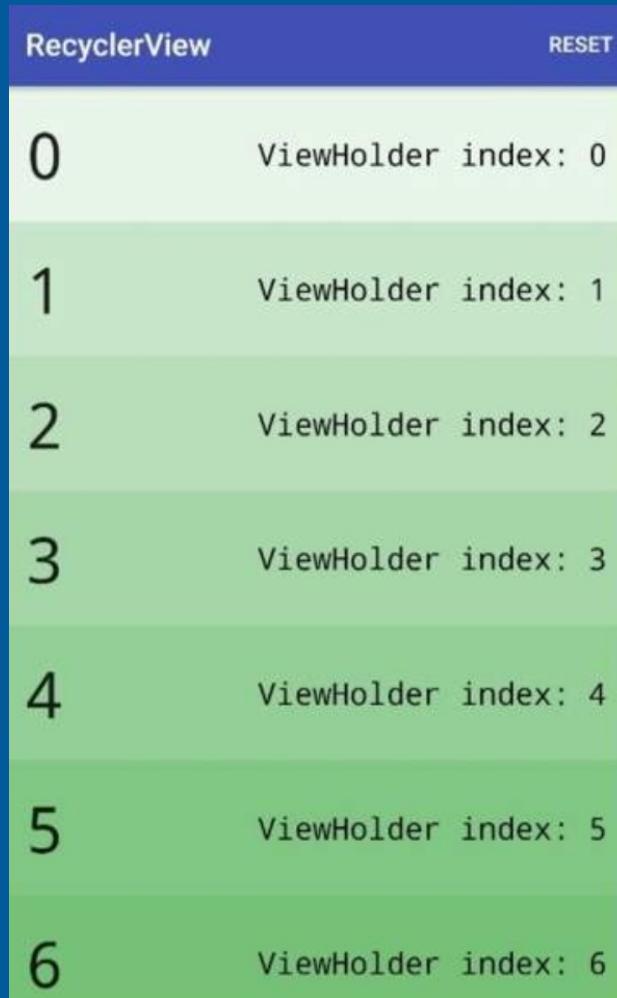


```
<androidx.recyclerview.widget.RecyclerView  
    android:layout_width="0dp"  
    android:layout_height="0dp"  
    app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```

# Comment ça marche ?



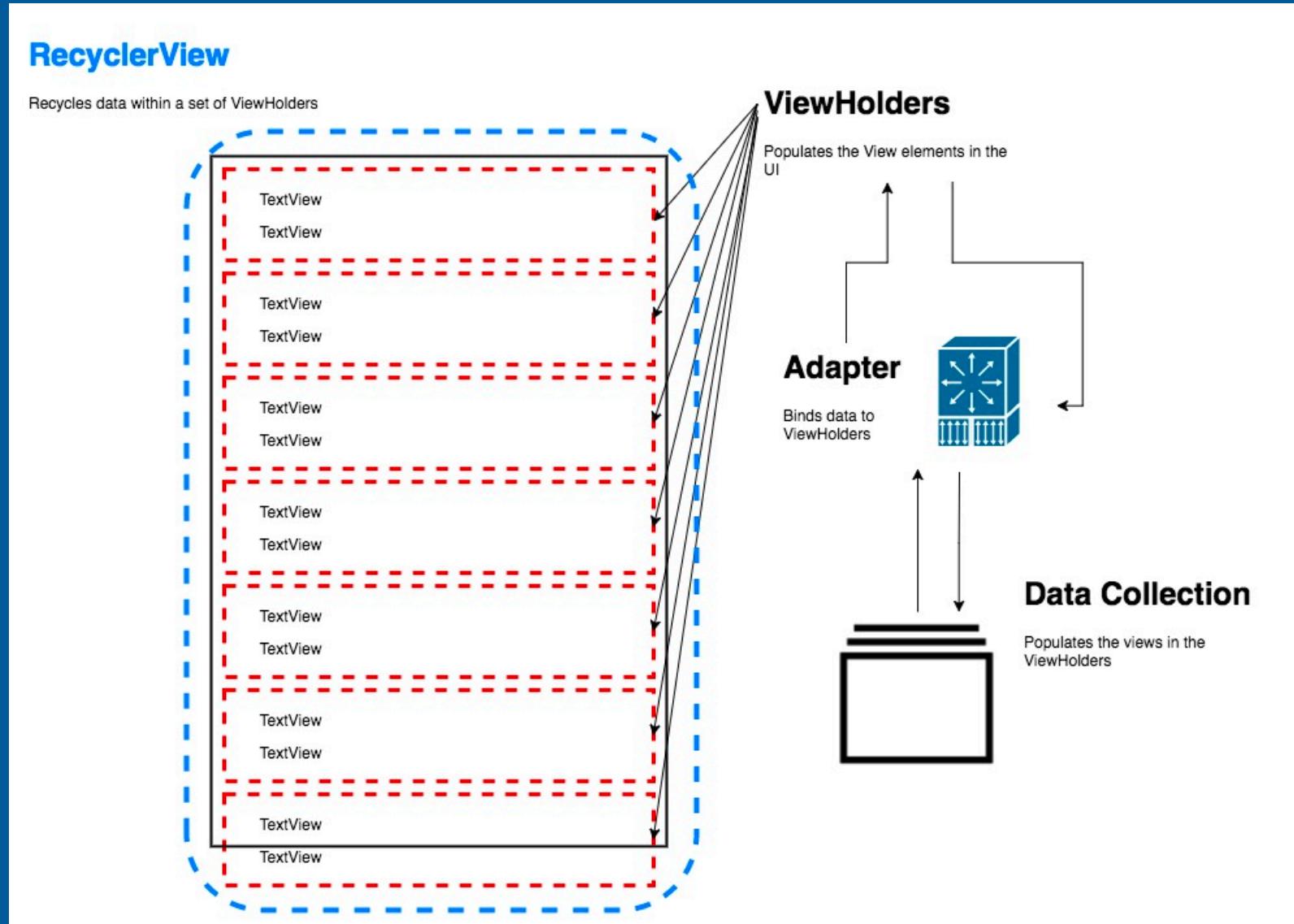
# ViewHolder



Les implémentations `RecyclerView.Adapter` doivent sous-classer `ViewHolder` et ajouter des champs pour la mise en cache des résultats `findViewByIId(int)` potentiellement gourmand.

ViewHolders appartiennent à l'adaptateur.  
Les adaptateurs ne doivent pas hésiter à utiliser leurs propres implémentations `ViewHolder` personnalisées pour stocker des données qui facilitent la liaison du contenu de la vue.

# Fonctionnement de la RecyclerView



# Exemple

On souhaite afficher une liste de users :

John  
42

Marie  
12

Jean  
35

Eve  
88

Antoine  
74

Amelie  
56

Alan  
9

Christophe  
11

Mathieu  
15

Pour cela, on aura besoin :

- 1) d'une classe User pour faire une liste d'objets
- 2) Une vue de type RecyclerView
- 3) Un layout pour les items
- 4) Un Adapter pour créer des ViewHolder
- 5) Un ViewHolder pour binder chaque item.



# Classe User

Notre liste se basera sur cette simple classe User :

```
data class User(  
    val name: String,  
    val age: Int,  
)
```

# Layout de l'item

Nous devons créer un layout pour les items de notre liste.

Nous mettons du databinding dans cet item en encapsulant notre layout avec la balise **<layout>**.

Dans notre balise **<data>** nous avons ajouté une variable de type de la classe **User** précédemment créée.

*Résultat attendu :*



```
<layout 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">

    <data>
        <variable
            name="user"
            type="com.iplant.models.User" />
    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp">

        <TextView
            android:id="@+id/textName"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textStyle="bold"
            tools:text="John"
            android:text="@{user.name}"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
            android:id="@+id/textAge"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            tools:text="42"
            android:text="@{String.valueOf(user.age)}"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/textName" />

    </androidx.constraintlayout.widget.ConstraintLayout>

</layout>
```

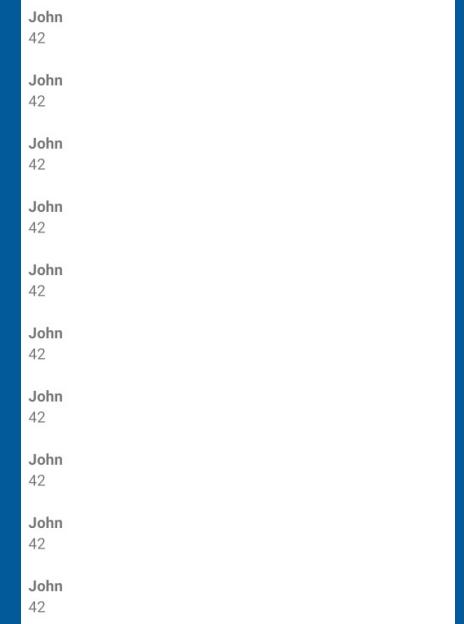
*item\_user.xml*



# Ajout de la vue RecyclerView sur un fragment :

View :

```
<androidx.recyclerview.widget.RecyclerView  
    android:layout_width="0dp"  
    android:layout_height="0dp"  
    android:id="@+id/recyclerUsers"  
    app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"  
    tools:listitem="@layout/item_user"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```



***tools:listitem*** nous permet d'avoir une preview de la liste à partir d'un layout d'item



# UserViewHolder

```
class UsersListAdapter {  
  
    inner class UserViewHolder(private val binding: ItemUserBinding) : RecyclerView.ViewHolder(binding.root){  
  
        fun bind(user: User){  
            binding.user = user  
        }  
    }  
}
```



# UsersListAdapter

```
class UsersListAdapter(private val users: List<User>) : RecyclerView.Adapter<UsersListAdapter.UserViewHolder>(){

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): UserViewHolder =
        UserViewHolder(ItemUserBinding.inflate(LayoutInflater.from(parent.context), parent, attachToRoot: false))

    override fun onBindViewHolder(holder: UserViewHolder, position: Int) {
        holder.bind(users[position])
    }

    inner class UserViewHolder(private val binding: ItemUserBinding) : RecyclerView.ViewHolder(binding.root){

        fun bind(user: User){
            binding.user = user
        }
    }

    override fun getItemCount(): Int = users.size
}
```



# Fragment

Dans notre fragment, nous créons un jeu de données factices sous forme de **MutableList<User>** puis nous créons une instance de notre **UsersListAdapter** (avec notre liste dans son constructeur). Enfin, nous désignons l'adaptateur de la **RecyclerView** avec cette instance :

```
val users = mutableListOf<User>()
users.add(User( name: "John", age: 42))
users.add(User( name: "Marie", age: 12))
users.add(User( name: "Jean", age: 35))
users.add(User( name: "Eve", age: 88))
users.add(User( name: "Antoine", age: 74))
users.add(User( name: "Amelie", age: 56))
users.add(User( name: "Alan", age: 9))
users.add(User( name: "Christophe", age: 11))
users.add(User( name: "Mathieu", age: 15))
users.add(User( name: "Mathilde", age: 73))
users.add(User( name: "Maxime", age: 43))

val adapter = UsersListAdapter(users)
binding.recyclerUsers.adapter = adapter
```



# Gestion du click via les lambdas

```
UsersListAdapter(private val users: List<User>, val onClick : (user: User) -> Unit)
```

```
fun bind(user: User){  
    binding.user = user  
    binding.root.setOnClickListener { onClick(user) }  
}
```

```
val adapter = UsersListAdapter(users) { it: User  
    Toast.makeText(requireContext(), it.toString(), Toast.LENGTH_SHORT).show()  
}  
binding.recyclerUsers.adapter = adapter
```

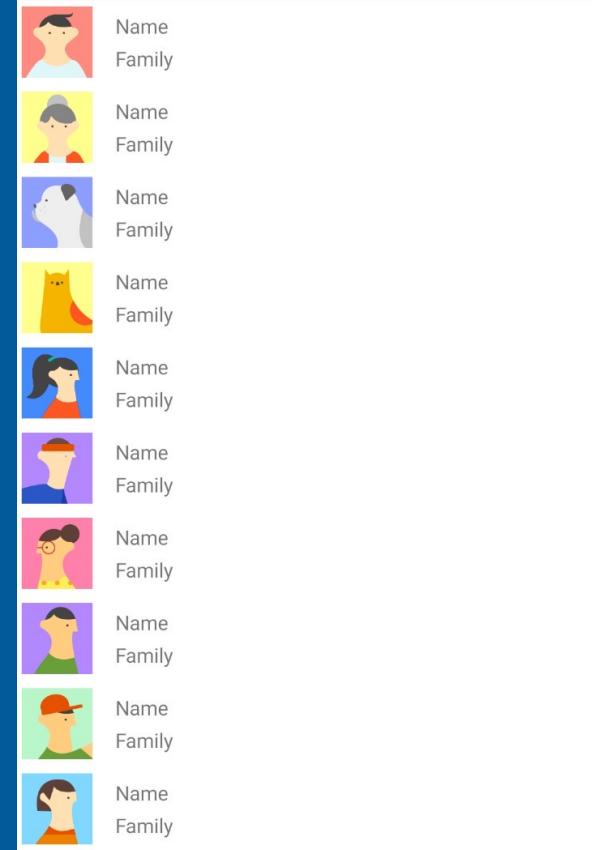


# TP – iPlant (7/10)



## Adapter et liste

- Dans SearchFragment, créer une liste de type RecyclerView qui nous servira à afficher les plantes.
- Pour commencer, partez d'une liste avec un item simple qui contient :
  - Nom de la plante
  - Famille de la plante
  - Image de la plante
- Créer une classe Plante qui contient nom, famille et url pour l'image.
- Créer des fausses données afin de tester votre liste.
- *Le screenshot de droite est une preview visible au niveau de l'éditeur de layout d'Android Studio. Afin d'avoir de fausses images générées en preview, vous pouvez utiliser pour tools:srcCompat :* `@tools:sample/avatars`



# Communiquer avec une API grâce à Retrofit2

---

Quand on aborde la gestion des APIs, le format standard de communication est JSON, appelé via des RESTful. Il existe bien d'autres standards, comme SOAP, OData et XML mais JSON est devenu le standard par défaut. Une API JSON peut être facilement consommée par des projets d'applications Web ou mobiles.



# Retrofit2

- Framework qui permet de mettre en place une abstraction simple pour vos appels réseaux
- Fournit un objet Call qui encapsule l'interaction avec une requête et sa réponse
- Permet de paramétriser l'objet Response
- Offre de multiples et efficaces convertisseurs (XML, JSON)
- Offre de multiples mécanismes pluggables d'exécution

```
@GET("posts/1")
Call<Post> getPostOne();

@POST("posts")
Call<Post> addNewPost(@Body Post post);

@PUT("users/1")
Call<User> updateUserOne(@Body User user);

@DELETE("user/{id}")
Call<User> deleteUserById(@Path("id")int id);
```

*Exemple d'interface*



# OkHttp

- OkHttp est un client HTTP
- Son objectif est de prendre en charge la communication avec le serveur
- Il est basé sur Okio (servant pour la lecture/écriture du flux de données)
- Il est bien plus performant que HttpClient/HttpURLConnection
- Il est désormais intégré dans la version 2 de Retrofit



- Bibliothèque open-source développée par Retrofit
- Elle permet de convertir un objet Java/Kotlin dans sa représentation JSON et vice versa (sérialisation et déserialisation)

```
import com.squareup.moshi.Json

data class Links(
    @field:Json(name = "genus")
    val genus: String?,
    @field:Json(name = "plant")
    val plant: String?,
    @field:Json(name = "self")
    val self: String?
)
```



# Exemple avec l'API de StackExchange

- Nous allons utiliser l'API de StackExchange pour afficher les dernières questions de Stackoverflow
- L'adresse de la requête que nous allons utiliser est :  
<https://api.stackexchange.com/2.2/questions?order=desc&sort=activity&site=stack overflow>

Ajouter Java 8 à son projet :

```
compileOptions {  
    sourceCompatibility JavaVersion.VERSION_1_8  
    targetCompatibility JavaVersion.VERSION_1_8  
}
```

Nous aurons besoin de ces librairies:

```
implementation 'com.squareup.retrofit2:retrofit:2.7.2'  
implementation 'com.squareup.retrofit2:converter-moshi:2.7.2'
```

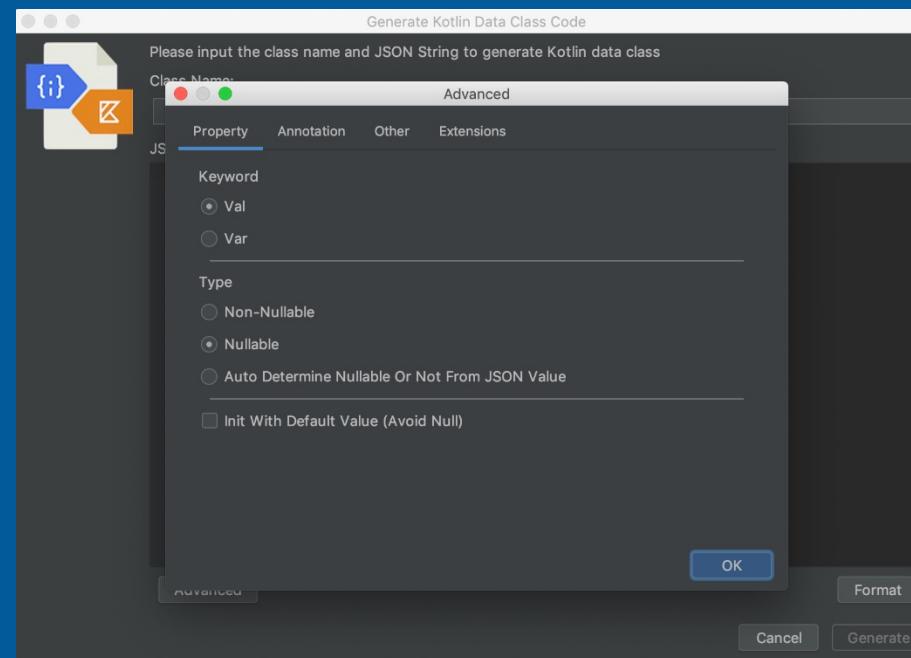
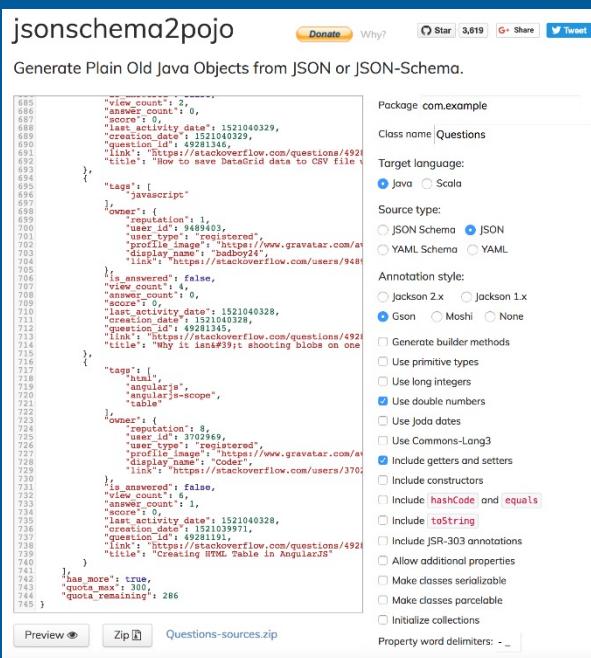
Et n'oubliez pas d'accorder la permission d'accéder à internet!

```
<uses-permission android:name="android.permission.INTERNET"/>
```



# Convertir un Json type en objet (POJO)

- Soit à la main (ça peut être très long)
  - Soit le dev back est sympa et il vous fournit ces objets
  - Avec un Json type et un convertisseur en ligne (exemple JsonSchema2Pojo)
  - Avec un Json type et un plugin Android Studio (exemple JsonToKotlinClass)

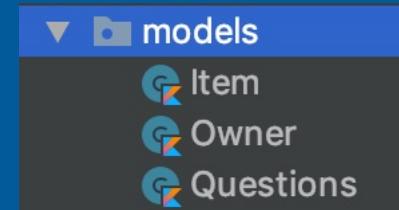


# Objets avec annotations Moshi

```
data class Questions(  
    @Json(name = "has_more")  
    val hasMore: Boolean,  
    @Json(name = "items")  
    val items: List<Item>,  
    @Json(name = "quota_max")  
    val quotaMax: Int,  
    @Json(name = "quota_remaining")  
    val quotaRemaining: Int  
)
```

```
data class Owner(  
    @Json(name = "display_name")  
    val displayName: String,  
    @Json(name = "link")  
    val link: String,  
    @Json(name = "profile_image")  
    val profileImage: String,  
    @Json(name = "reputation")  
    val reputation: Int,  
    @Json(name = "user_id")  
    val userId: Int,  
    @Json(name = "user_type")  
    val userType: String  
)
```

```
data class Item(  
    @Json(name = "accepted_answer_id")  
    val acceptedAnswerId: Int,  
    @Json(name = "answer_count")  
    val answerCount: Int,  
    @Json(name = "bounty_amount")  
    val bountyAmount: Int,  
    @Json(name = "bounty_closes_date")  
    val bountyClosesDate: Int,  
    @Json(name = "creation_date")  
    val creationDate: Int,  
    @Json(name = "is_answered")  
    val isAnswered: Boolean,  
    @Json(name = "last_activity_date")  
    val lastActivityDate: Int,  
    @Json(name = "last_edit_date")  
    val lastEditDate: Int,  
    @Json(name = "link")  
    val link: String,  
    @Json(name = "owner")  
    val owner: Owner,  
    @Json(name = "protected_date")  
    val protectedDate: Int,  
    @Json(name = "question_id")  
    val questionId: Int,  
    @Json(name = "score")  
    val score: Int,  
    @Json(name = "tags")  
    val tags: List<String>,  
    @Json(name = "title")  
    val title: String,  
    @Json(name = "view_count")  
    val viewCount: Int  
)
```



# Interface

- C'est dans cette interface que sera centralisé les différentes requêtes de notre API
- Le type de requête (GET, POST, PUT, DELETE) est défini grâce à une annotation
- La classe Call encapsule la classe Moshi qui sert à convertir notre Json en classe Kotlin : quand nous aurons une réponse du serveur, celui-ci nous retournera une instance de Call qui encapsulera une instance de Questions.

```
interface StackService {  
    @GET("questions?order=desc&sort=activity&site=stackoverflow")  
    fun getQuestions(): Call<Questions>  
}
```



# Client

```
const val BASE_URL = "https://api.stackexchange.com/2.2/"

val retrofitClient: StackService by lazy {
    val retrofit = Retrofit.Builder()
        .baseUrl(BASE_URL) ←
        .client(OkHttpClient()) ←
        .addConverterFactory(MoshiConverterFactory.create())
        .build()
    return@lazy retrofit.create(StackService::class.java)
}
```

URL de base pour nos appels webservice.

Choix du client HTTP à utiliser pour les requêtes. OkHttpClient() est un client intégré par défaut dans Retrofit2.

Choix du convertisseur Json/Kotlin. Ici nous utilisons MoshiConverterFactory qui fonctionnera avec nos modèles annotés avec Moshi.



# ApiResult

```
sealed class ApiResult<out T: Any> {
    data class Success<out T : Any>(val data: T) : ApiResult<T>()
    data class Error(val exception: String) : ApiResult<Nothing>()
}

suspend fun <T : Any> safeApiCall(call: suspend () -> Response<T>): ApiResult<T> {
    return try {
        val myResp = call.invoke()
        when {
            myResp.isSuccessful -> ApiResult.Success(myResp.body()!!)
            else -> ApiResult.Error( exception: myResp.errorBody()?.toString() ?: "Error")
        }
    } catch (e: Exception) {
        ApiResult.Error( exception: e.message ?: "Internet error runs")
    }
}
```



# Retours possibles

- **onResponse** : vous avez eu une réponse (erreur 404 ou 500 inclus). Pour vérifier que votre réponse a un statut 200-300, il faut vérifier si elle est **isSuccessful()**.
- **onFailure** : network exception ou si une exception non prévue est arrivée pendant la requête ou le traitement de votre réponse.

# TP – iPlant (8/10)

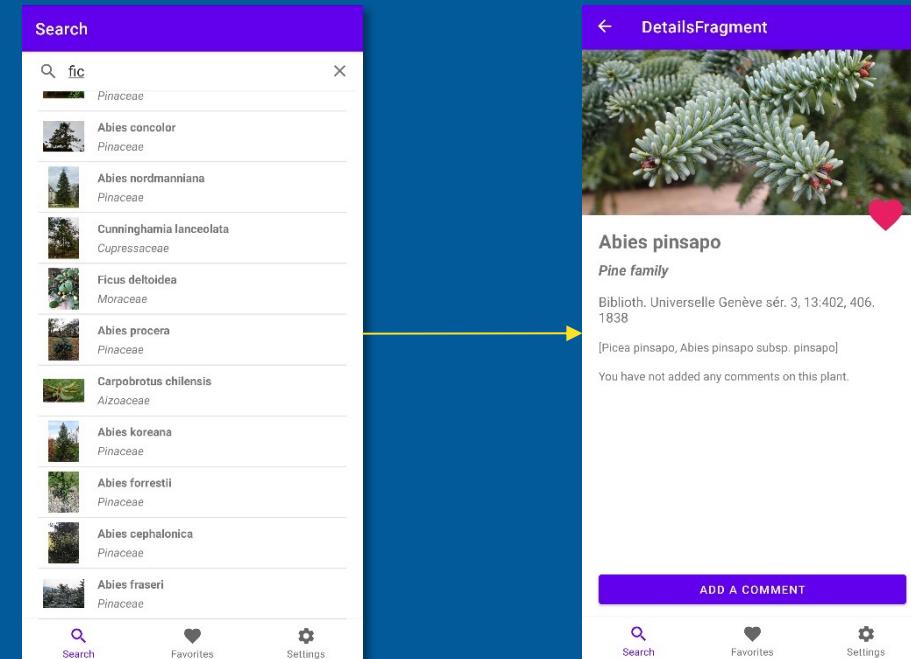


## Communication avec l'API Trefle

- Vous allez maintenant communiquer avec l'API [trefle.io](#)
  - Il vous faudra au préalable créer un compte afin d'avoir un token d'accès à l'API
- Vous utiliserez ces deux requêtes :
  - [\*plants/search\*](#) : pour rechercher des plantes dans **SearchFragment**
  - [\*plants/{id}\*](#) : pour afficher les détails d'une plante dans **DetailsFragment**

Réutilisez l'adapter que vous avez créé afin d'afficher le résultat de la recherche de l'utilisateur. Lorsque l'utilisateur sélectionnera une plante, afficher les détails de celle-ci dans DetailsFragment.

Inspirez-vous de l'exemple à droite.



# Base de données avec Room

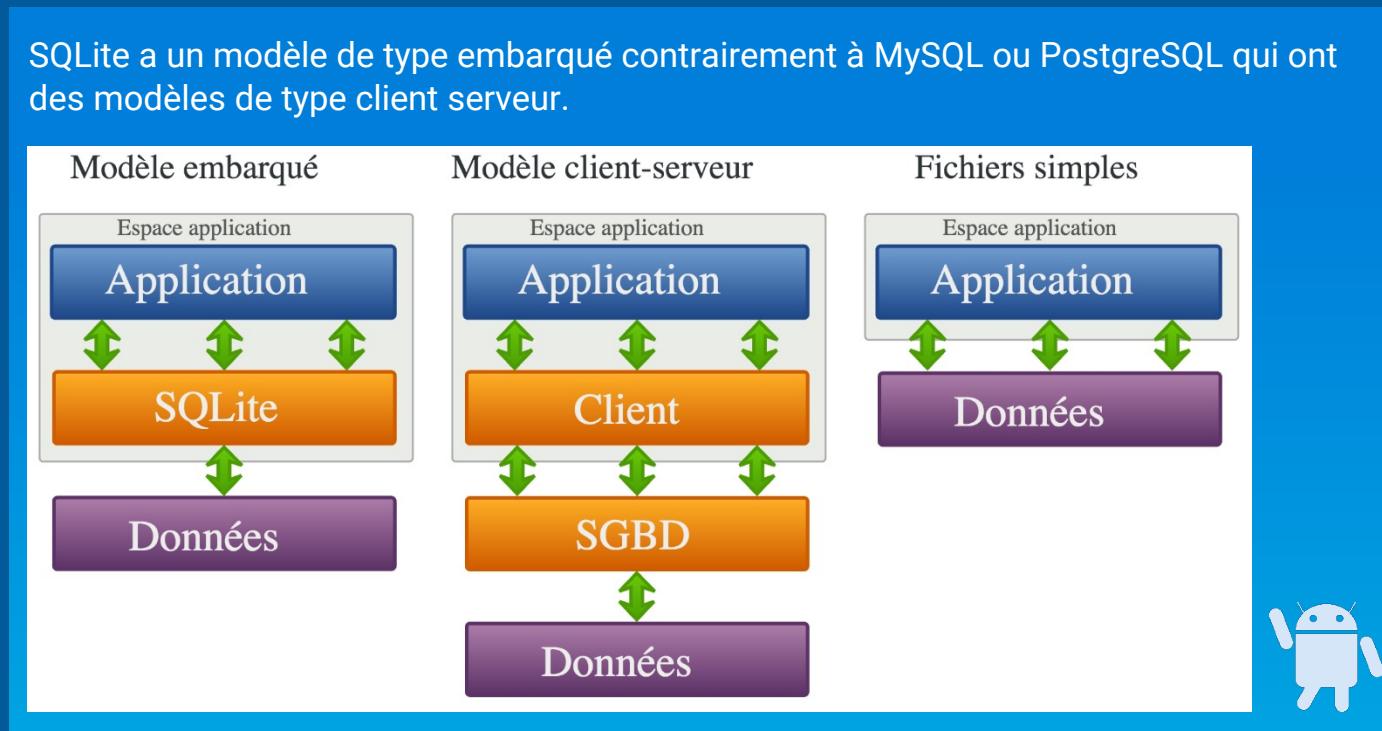
---

Gérer la persistance de données dans son app



# SQLite c'est quoi ?

- Bibliothèque écrite en C qui propose un moteur de base de données accessible par le langage SQL (standard SQL-92).
- SQLite est directement intégrée aux programmes
- Première version de SQLite en août 2000



# Types de données dans SQLite

- **INTEGER** : pour les entiers (*sans virgule*)
  - INT, INTEGER, TINYINT, SMALLINT, MEDIUMINT, BIGINT, UNSIGNED BIG INT, INT2, INT8
- **REAL** : pour les nombres réels (*avec virgule*)
  - REAL, DOUBLE, DOUBLE PRECISION, FLOAT
- **TEXT** : pour les chaînes de caractères
  - CHARACTER(20), VARCHAR(255), VARYING CHARACTER(255), NCHAR(55), NATIVE CHARACTER(70), NVARCHAR(100), TEXT, CLOB
- **NUMERIC** : pour les données numériques
  - NUMERIC, DECIMAL(10,5), BOOLEAN, DATE, DATETIME
- **BLOB** : pour les données brutes (*par exemple : images, etc*)
  - BLOB (*pas de datatype*)
- **NULL** : pour les données NULL



# Syntaxe SQLite

```
CREATE TABLE nom_de_la_table (
    nom_du_champ_1 type {contraintes},
    nom_du_champ_2 type {contraintes},
    ...);
```

Pour chaque attribut, on doit déclarer au moins 2 informations :

- son nom
- son type de donnée

Il est possible de déclarer des contraintes pour chaque attribut à l'emplacement de {contraintes}. On trouve comme contraintes :

- **PRIMARY KEY** pour désigner la clé primaire sur un attribut
- **NOT NULL** pour indiquer que cet attribut ne peut valoir NULL
- **CHECK** afin de vérifier que la valeur de cet attribut est cohérente
- **DEFAULT** sert à préciser une valeur par défaut

```
1 CREATE TABLE nom_de_la_table (
2     nom_du_champ_1 INTEGER PRIMARY KEY,
3     nom_du_champ_2 TEXT NOT NULL,
4     nom_du_champ_3 REAL NOT NULL CHECK (nom_du_champ_3 > 0),
5     nom_du_champ_4 INTEGER DEFAULT 10);
6
```



# Paramètres de la méthode query()

Paramètre	Commentaire
<b>String dbName</b>	Nom de la table
<b>String[] columnNames</b>	Liste des colonnes à retourner. Toutes les colonnes seront retournées avec en passant <b>null</b> .
<b>String whereClause</b>	Clause <b>where</b> : filtre pour la sélection des données. Toutes les données seront sélectionnées en passant <b>null</b> .
<b>String[] selectionArgs</b>	Si vous incluez des <b>?</b> dans la <b>whereClause</b> , ils seront remplacés par les valeurs du tableau <b>selectionArgs</b> .
<b>String[] groupBy</b>	Filtre qui déclare comment regrouper les lignes. Les lignes ne seront pas groupées en passant <b>null</b> .
<b>String[] having</b>	Filtre pour les groupes. Pas de filtrage en passant <b>null</b> .
<b>String[] orderBy</b>	Colonnes de la table utilisées pour ordonner les données. Les données ne seront pas ordonnées en passant <b>null</b> .



# Android Jetpack

- Android Jetpack est un ensemble de librairies, d'outils et de conseils architecturaux qui vous aideront à créer rapidement et facilement de superbes applications Android.
- Les composants sont individuellement adoptables mais construits pour fonctionner ensemble tout en profitant des fonctionnalités de Kotlin qui vous rendent plus productif.
- Android Jetpack gère des activités fastidieuses telles que les tâches d'arrière-plan, la navigation et la gestion du cycle de vie.
- Conçus autour de pratiques de conception modernes, les composants Android Jetpack permettent moins de plantage et moins de fuite de mémoire grâce à la rétrocompatibilité.



# Foundation, Architecture, Behavior et UI



## Foundation

Foundation components provide core system capabilities, Kotlin extensions and support for multidex and automated testing.

### [AppCompat](#)

Degrade gracefully on older versions of Android

### [Android KTX](#)

Write more concise, idiomatic Kotlin code

### [Multidex](#)

Provide support for apps with multiple DEX files

### [Test](#)

An Android testing framework for unit and runtime UI tests



## Architecture

Architecture components have classes that help manage your UI component lifecycle, handle data persistence, and more.

### [Data Binding](#)

Declaratively bind observable data to UI elements

### [Lifecycles](#)

Manage your activity and fragment lifecycles

### [LiveData](#)

Notify views when underlying database changes

### [Navigation](#)

Handle everything needed for in-app navigation

### [Paging](#)

Gradually load information on demand from your data source

### [Room](#)

Fluent SQLite database access

### [ViewModel](#)

Manage UI-related data in a lifecycle-conscious way

### [WorkManager](#)

Manage your Android background jobs



## Behavior

Behavior Components help you design robust, testable, and maintainable apps.

### [Download manager](#)

Schedule and manage large downloads

### [Media & playback](#)

Backwards compatible APIs for media playback and routing (including Google Cast)

### [Notifications](#)

Provides a backwards-compatible notification API with support for Wear and Auto

### [Permissions](#)

Compatibility APIs for checking and requesting app permissions

### [Sharing](#)

Provides a share action suitable for an app's action bar

### [Slices](#)

Create flexible UI elements that can display app data outside the app



## UI

UI components make it easy for you to make your app not only easy, but delightful to use.

### [Animation & transitions](#)

Move widgets and transition between screens

### [Auto](#)

Components to help develop apps for Android Auto.

### [Emoji](#)

Enable an up-to-date emoji font on older platforms

### [Fragment](#)

A basic unit of composable UI

### [Layout](#)

Lay out widgets using different algorithms

### [Palette](#)

Pull useful information out of color palettes

### [TV](#)

Components to help develop apps for Android TV.

### [Wear OS by Google](#)

Components to help develop apps for Wear.

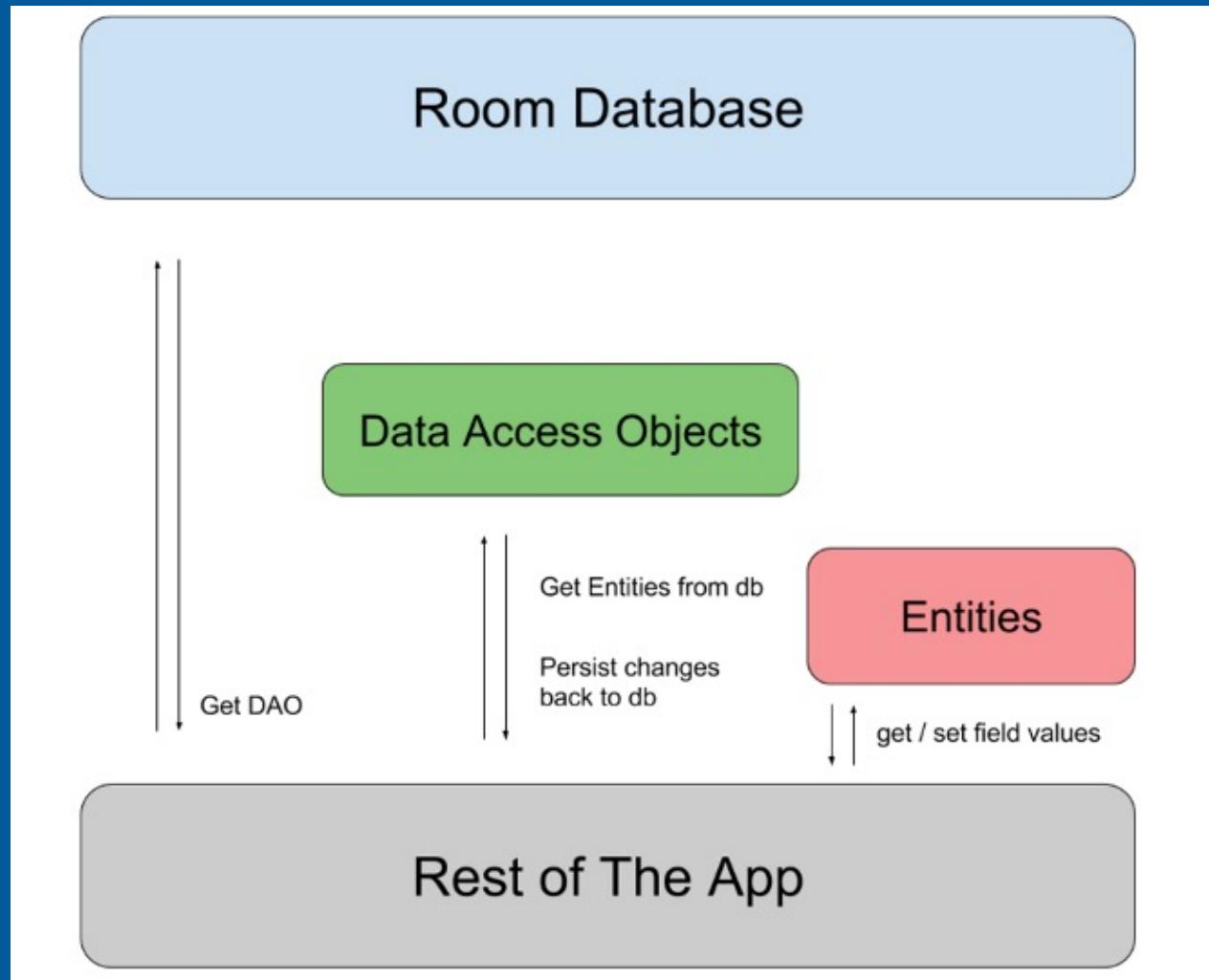


# Room Persistence Library

- Room est une couche abstraite qui enveloppe la base de données SQLite standard
- Room simplifie toutes les opérations liées à la base de données et les rend beaucoup plus puissantes car **elle permet de renvoyer des observables et des requêtes SQL vérifiées à la compilation.**
- Room est composé de trois composants principaux :
  - la base de données
  - le DAO (Data Access Objects)
  - l'entité
- La mise en œuvre des composants est assez simple, grâce aux annotations et aux classes abstraites fournies
  - Une table de base de données exclusive est créée pour chaque classe annotée avec @Entity.
  - Le DAO est l'interface annotée avec @Dao qui gère l'accès aux objets de la base de données et de ses tables. Il existe quatre annotations spécifiques pour les opérations DAO de base: @Insert, @Update, @Delete et @Query.
  - Le composant Database est une classe abstraite annotée avec @Database, qui étend RoomDatabase. La classe définit la liste des Entités et de leurs DAO.



# Comment ça marche ?



# Librairie Room Persistence Library

```
apply plugin: 'kotlin-kapt'
```

```
def room_version = "2.1.0"  
implementation "androidx.room:room-runtime:$room_version"  
kapt "androidx.room:room-compiler:$room_version"
```



# Entity

```
6  @Entity
7  data class User(
8      @PrimaryKey (autoGenerate = true) val id: Int,
9      val name: String,
10     val age: Int
11    ){
12        constructor(name: String, age: Int) : this(id: 0, name, age)
13    }
14 }
```



```
9  @Dao
10 interface UserDao {
11
12     @Query( value: "SELECT * FROM User")
13     fun getAll() : List<User>
14
15     @Query( value: "SELECT * FROM User WHERE id = :id")
16     fun getUserById(id: Int) : User
17
18     @Insert
19     fun insert(userEntity: User)
20
21     @Delete
22     fun deleteUser(user: User)
23
24     @Query( value: "DELETE FROM User")
25     fun deleteAll()
26
27 }
28 }
```



# Database

```
@Database(entities = [User::class], version = 1)
abstract class UserDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao
}

private var INSTANCE: UserDatabase? = null

fun getDatabase(context: Context): UserDatabase? {
    if (INSTANCE == null) {
        INSTANCE = Room.databaseBuilder(
            context,
            UserDatabase::class.java,
            name: "user_db")
            .allowMainThreadQueries()
            .build()
    }
    return INSTANCE
}
```



# Utilisation dans une Activity

```
8
9  class MainActivity : AppCompatActivity() {
10
11    override fun onCreate(savedInstanceState: Bundle?) {
12        super.onCreate(savedInstanceState)
13        setContentView(R.layout.activity_main)
14
15        val userDao : UserDao? = getDatabase(applicationContext)?.userDao()
16        val john = User( name: "John", age: 42)
17        userDao?.insert(john)
18
19        userDao?.getAll()?.get(0).let { it: User?
20            Log.d( tag: "myDebug", msg: "id : " + (it?.id ?: ""))
21            Log.d( tag: "myDebug", msg: "name : " + (it?.name ?: ""))
22            Log.d( tag: "myDebug", msg: "age : " + (it?.age ?: ""))
23            ""^let
24        }
25    }
26
27 }
28 }
```

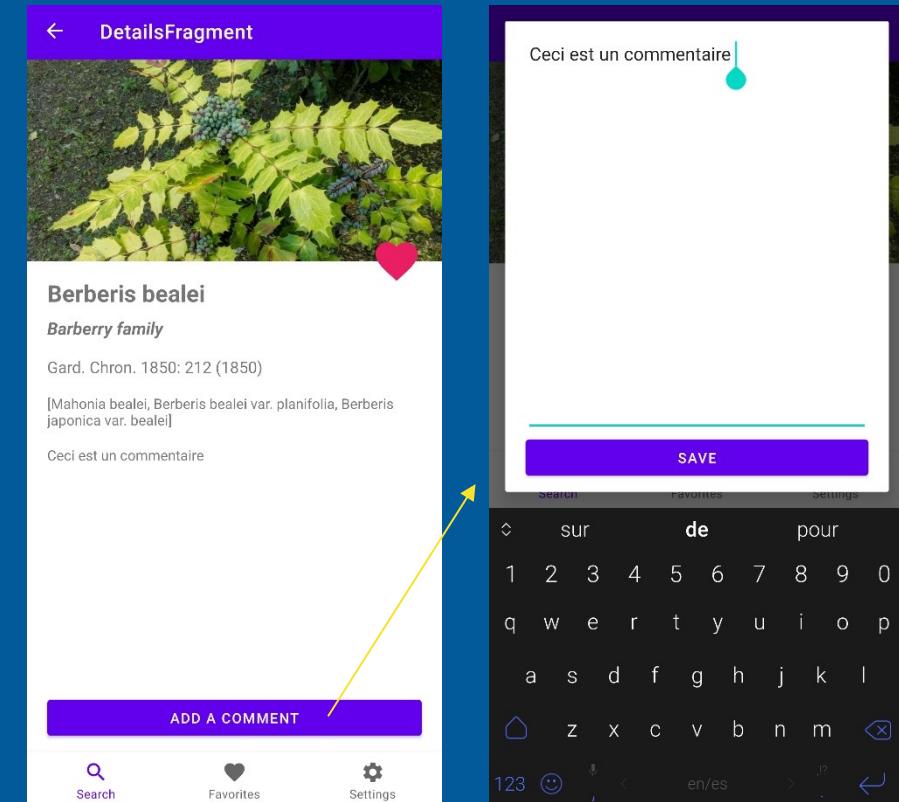


# TP – iPlant (9/10)



## Gestion d'une base données avec Room

- Utilisez la Room Persistence Library afin de stocker les commentaires liés aux plantes ainsi que les plantes favorites.
- Concernant les commentaires :
  - Si un commentaire existe en local dans la fiche dans laquelle l'utilisateur se trouve, afficher le commentaire
  - Affichez une **Dialog** lorsque l'utilisateur clique sur le bouton permettant d'ajouter un commentaire.
  - Demandez le commentaire de l'utilisateur dans celle-ci.
  - Si un commentaire existe déjà pour cette plante, pré-remplir le champs de texte par le commentaire déjà existant. Le bouton devra aussi avoir pour texte « change your comment ».
- Concernant les favoris :
  - Changer la teinte du cœur visible sur la fiche de la plante en fonction du fait qu'elle soit en favoris ou non.
  - Pour cela on utilisera l'attribut **tint**





- Arrivé à cette étape, vous devriez avoir une app fonctionnelle.
- Si vous avez du temps devant vous ou que vous souhaitez aller plus loin :
  - Affichez la liste des plantes favorites dans FavoritesFragment. Au clique, ouvrir DetailsFragment avec la bonne plante.
  - Dans SettingsFragment, lancez un intent implicite au clique sur le bouton « Write us » afin d'écrire un mail à l'équipe de dev
  - Toujours dans SettingsFragment, effacez toutes les données (SharedPreferences et Room Persistance Library) lorsque l'on clique sur « Clear all data »
  - Cachez les synonyms dans DetailsFragment lorsque l'option est activée dans SettingsFragment
  - Générez un APK afin de partager votre application à d'autres personnes

# Tests unitaires

---

Tester le bon fonctionnement d'une partie précise  
d'un programme



# Avantages

- Avoir un retour rapide sur votre développement.
  - Si le test échoue, vous pouvez rapidement corriger votre code pendant que tout est frais dans votre esprit;
- Il vous permet d'avoir un filet de sécurité.
  - Si vous cassez quelque chose, vous vous en apercevez tout de suite ;
- Il vous permet d'éviter d'accumuler de la dette technique, en ayant une application qui soit facilement maintenable et évolutive.



# Approches possibles

- Ne jamais en écrire (majorité des cas)
- Écrire les tests quand on a le temps et la motivation (c'est à dire très rarement) ;
- Écrire les tests avant toute ligne de code métier (TTD ou Test-Driven-Development) :
  - Écrire les tests qui permettent de valider les différentes exigences de votre application puis écrire le code de l'application qui permet de valider ces tests.
  - Cette méthode permet d'éviter d'écrire trop de code. La base de code est plus saine et moins sujette aux erreurs. D'un autre côté, il n'est pas toujours évident d'avoir cette approche avec des tests d'interface ou lorsqu'une connectivité réseau est présente. Car les paramètres qui rentrent en compte sont multiples.



# Les différentes catégories de test

- **Les tests unitaires**
  - doivent être petits, rapides et très ciblés. Ils permettent de détecter les erreurs au plus tôt. La majorité des composants sont simulés, ce qui implique qu'ils ne représentent pas forcément la réalité. Ils ont l'avantage de pouvoir s'exécuter sur l'ordinateur du développeur, ce qui permet de les passer très rapidement ;
- **Les tests d'intégration**
  - permettent de tester plusieurs composants. Certains sont simulés, d'autres sont réels. Ils peuvent s'exécuter sur l'émulateur ou sur un équipement réel. Ils sont plus longs à écrire, mais sont beaucoup plus fidèles que les tests unitaires ;
- **Les tests d'interface**
  - permettent de tester une application comme un vrai utilisateur. Ces tests ont l'avantage d'être extrêmement fidèles à la réalité, mais ce sont les plus longs et les plus compliqués à écrire.



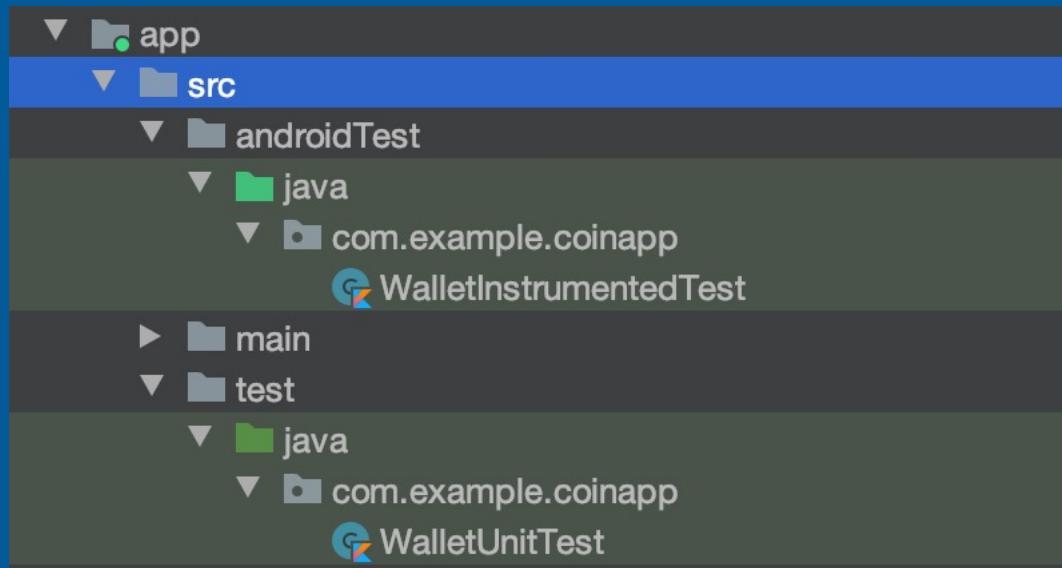
# Les tests sur Android

- **Les tests unitaires *locaux*** (ou *Local Unit Tests*).
  - exécutés sur la JVM de votre ordinateur
    - Sachant qu'il n'existe pas de dépendance avec Android, il n'est pas nécessaire de démarrer l'émulateur ou d'accéder à un équipement réel. Ces tests sont donc déroulés très rapidement
    - Parfois, un test unitaire local a besoin d'utiliser une dépendance Android pour s'exécuter (par exemple accéder à l'objet **Context**). Dans ce cas, au lieu d'écrire un test unitaire instrumentalisé pour accéder au système Android, il est possible d'utiliser un "bouchon", ou *mock* en anglais. Ce mécanisme permet de continuer de bénéficier des avantages de rapidité des tests unitaires locaux.
- **Les tests unitaires *instrumentalisés*** (ou *Instrumented Unit Tests*).
  - ont besoin du système Android pour fonctionner (car ils utilisent ses API)
  - Le lancement de l'émulateur ou le branchement d'un équipement réel est nécessaire pour pouvoir exécuter ces tests. Ils sont donc plus longs à dérouler.

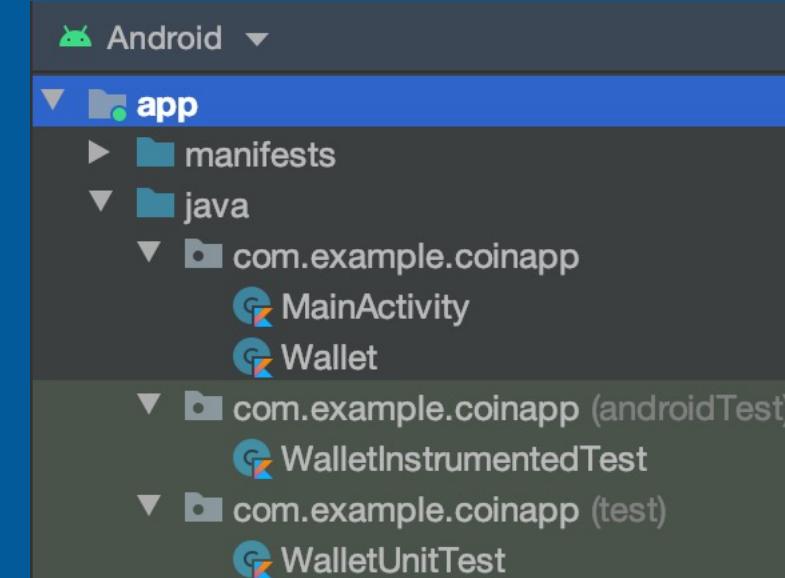


# Où sont les tests ?

Arborescence réelle :



Arborescence simplifiée :



# Junit et Espresso

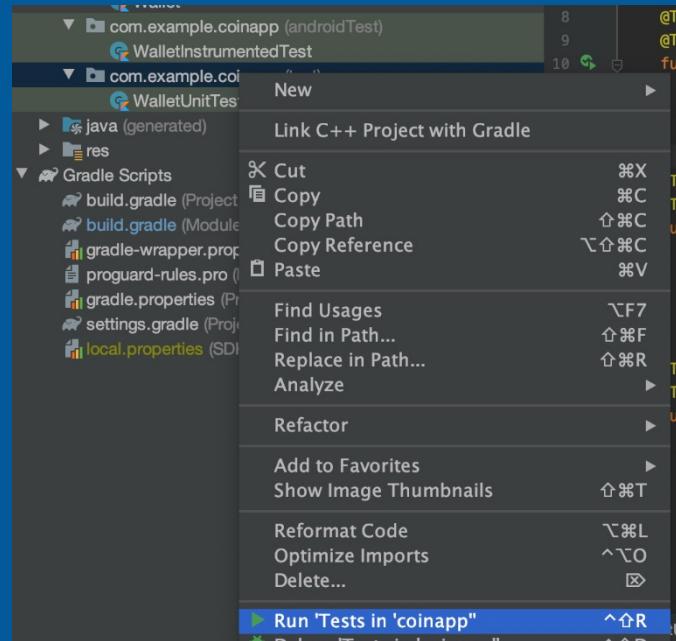
- Avec Android Studio, les tests unitaires sont exécutés en utilisant le framework JUnit 4. Ce framework est couramment utilisé par la communauté des développeurs Java.
- La librairie JUnit est chargée grâce à Gradle.

```
testImplementation 'junit:junit:4.12'  
androidTestImplementation 'androidx.test.ext:junit:1.1.1'  
androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
```

```
testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
```

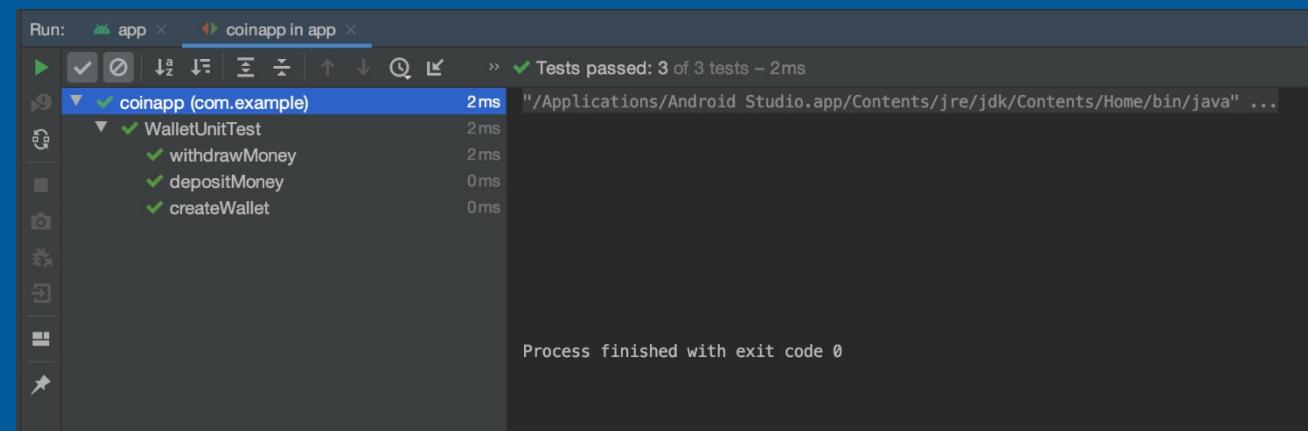


# Lancement d'un test unitaire



A screenshot of the Java code editor showing a context menu over a piece of code. The menu includes options like Run 'createWallet()', Debug 'createWallet()', and Run 'createWallet()' with Coverage.

```
class WalletUnitTest {  
    @Test  
    @Throws(Exception::class)  
    fun createWallet() {}  
    @Test  
    @Throws(java.lang.Exception::class)  
    fun depositMoney() {}  
}
```



# Création de son premier test unitaire

```
data class Wallet(var balance: Double)
```

```
@Test  
@Throws(Exception::class)  
fun createWallet() {  
    val wallet = Wallet( balance: 42.00)  
    assertEquals( expected: 42.00, wallet.balance, delta: 0.001)  
}
```



# Exercice

## Premiers tests unitaires



- Créer deux méthodes dans une classe Wallet(balance: Double) :
  - **deposit()** -> ajoute de l'argent à la balance
  - **withdraw()** -> retire de l'argent à la balance
- Créer deux tests unitaires qui testent les méthodes créées précédemment :
  - **depositMoney()**
  - **withdrawMoney()**

# Automatiser les tests

---

- Ouvrez la fenêtre listant les différentes configurations disponibles (**Run > Edit Configurations...** ou sur la barre d'outils)
- Dans le menu de gauche, déroulez **Android App** puis sélectionnez **app**
- Tout en bas à droite, dans la section *Before launch*, cliquez sur le petit bouton +
- Sélectionnez **Run Another Configuration**
- Vous devriez voir apparaître la configuration **WalletUnitTest**. Sélectionnez-la puis cliquez sur OK
- Cliquez sur **Apply** puis sur **OK**



# Mockito

---

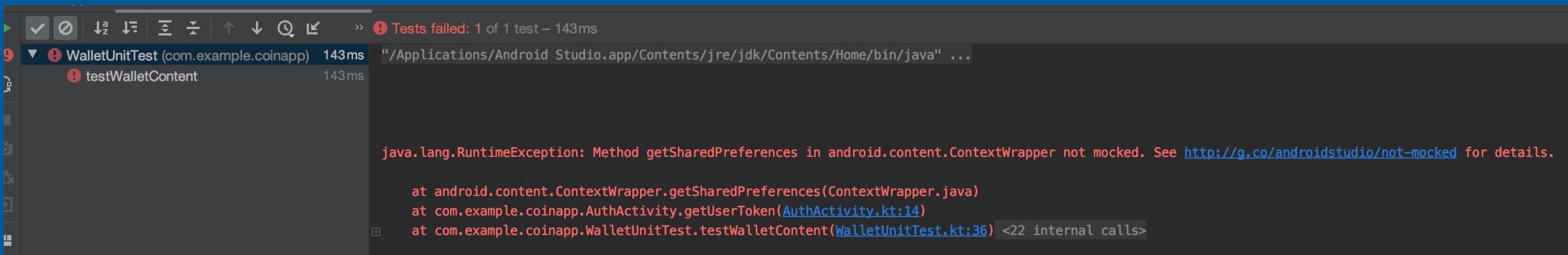
- Mockito est un framework qui vous permet de bouchonner certaines parties de votre code. Il vous permet de simuler l'appel à certaines méthodes, et de décider à l'avance de la valeur retournée.
- Exemple :
  - Nous souhaitons faire évoluer notre app en ajoutant une nouvelle fonctionnalité. Cette fonctionnalité permet de protéger l'accès au contenu du porte-monnaie virtuel à l'aide d'une empreinte digitale et d'un mot de passe (eh oui, l'application est ultra-sécurisée). Afin de pouvoir récupérer le solde du porte-monnaie, un écran est donc présenté à l'utilisateur. Cet écran lui demande de saisir son mot de passe et de valider son empreinte digitale. Cette opération permet de récupérer un token qui sera utilisé pour récupérer le solde du porte-monnaie.
  - Le problème, c'est qu'il est désormais nécessaire de lancer l'application et d'intervenir manuellement pour récupérer ce jeton. Et c'est là que Mockito intervient : il va nous permettre de simuler cette partie, et nous renvoyer directement un token. Et le tout localement, savoir avoir à lancer l'émulateur.



# Context sans Mockito

```
class AuthActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_auth)  
    }  
  
    fun getUserToken(): String {  
        val prefs = getSharedPreferences("UserToken", MODE_PRIVATE)  
        // Fetches the token, does a lot of things, then...  
        return "theToken";  
    }  
}
```

```
@Test  
@Throws(java.lang.Exception::class)  
fun testWalletContent() {  
    val authActivity = AuthActivity()  
    val token: String = authActivity.getUserToken()  
    // Test token  
}
```



L'API **SharedPreferences** repose sur la classe Context. Or, cette classe n'est disponible qu'au sein du système Android. Sachant que le test est exécuté localement sur la JVM, nous n'y avons pas accès. D'où l'erreur constatée, qui stipule que la méthode **getSharedPreferences()** n'est pas bouchonnée.



# Création du bouchon avec Mockito

```
testImplementation "org.mockito:mockito-core:2.28.2"  
testImplementation 'org.mockito:mockito-inline:2.13.0'
```

```
import org.mockito.Mockito.mock
```

```
@Test  
@Throws(java.lang.Exception::class)  
fun testWalletContent() {  
    val authActivity = mock(AuthActivity::class.java)  
    `when`(authActivity.getUserToken()).thenReturn("FakeToken")  
    val token = authActivity.getUserToken()  
    // Test token  
}
```

Lorsque la méthode **getUserToken()** est appelée, alors renvoyer directement la chaîne de caractères "**FakeToken**".

Vous pouvez ainsi bouchonner toutes les méthodes qui vous intéressent, et de décider vous-même de la valeur à renvoyer.

Il existe toutefois une limitation à l'utilisation de Mockito : il n'est pas possible de bouchonner des méthodes statiques ou privées.



# D'autres frameworks de test

- Robolectric
  - Exécuter un test sur l'émulateur Android est lent. Ce framework bouchonne les appels au SDK d'Android, ce qui vous permet d'exécuter vos tests localement, sur la JVM. Ainsi, vos tests sont beaucoup plus proches de la réalité, tout en s'exécutant le plus rapidement possible.
- Espresso
  - Cet utilitaire permet d'exécuter des tests d'interface. Dit autrement, il simule les clics sur les différents éléments graphiques de votre application, en reproduisant ce que vous lui avez appris, puis en comparant les résultats. Cet outil est nativement intégré à Android Studio. Pour découvrir son fonctionnement, cliquez sur **Run** puis **Record Espresso Test**.



# Industrialiser les tests avec l'intégration continue

- Le principe d'intégration continue (ou *Continuous Integration* en anglais) consiste à dérouler le plan de tests après chaque modification du code source.
  - Par exemple, les tests sont déroulés lorsque vous poussez vos modifications vers le serveur de gestion de versions (par exemple **git**). Ou alors lorsque vous tentez de fusionner vos modifications dans la branche principale du projet. Si un seul test échoue, la fusion n'est pas possible. Cela permet d'éviter toute régression, et force le développeur à corriger son application.
- Quelques outils d'intégration continue : **Jenkins**, **CircleCI**, **Bamboo** et **Travis CI**.

