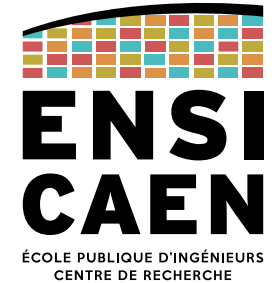


Développement mobile



Créé par ndiaga.faye@ensicaen.fr
Présenté par joan.reynaud@ensicaen.fr

2024-2025



L'École des INGÉNIEURS Scientifiques

A PROPOS DU COURS

■ Organisation

- ❑ **CM:** 5h.
- ❑ **TP:** 12h.



OBJECTIF DU COURS

- La philosophie de la programmation Android
- Connaissance de l'environnement de programmation
- Notions sur la programmation
- Manipulations



INTRODUCTION

- Android est un système d'exploitation et un framework pour développer des applications pour des terminaux de tous types
 - Actuellement : on parlera des Smartphones et Tablettes
 - Mais il existe : Smart Watch, Google Glass, Android TV, TPE, Android Auto...
 - Systèmes embarqués
 - Processeur ARM
- Les choix prépondérants de cette architecture sont basés sur la volonté de maîtriser un environnement contraint
 - Ressources potentiellement très limitées
 - Capacité de la batterie (optimisation du code)
 - Mémoire (qui peut être < 1 Go)
 - Stockage (... < 10 Go)



- A minima un téléphone Android doit avoir les ressources suivantes :
 - Fréquence de processeur > 250 Mhz
 - ≥ 64 Mb de RAM pour l'ensemble du système
 - Système d'exploitation basé sur Linux
 - Résolution minimale de 240 x 320
- Cadre applicatif très différent :
 - Rien ne doit empêcher de recevoir un appel téléphonique
 - Taille de l'écran réduite, pas de clavier, mais écran tactile
 - Session d'un application très courte → importance de l'ergonomie

AVANT ANDROID

Le mobile sans Java

- ❑ Développement principalement en C/C++ (sans parler de la préhistoire en assembleur)
- ❑ Symbian et autres API propriétaires

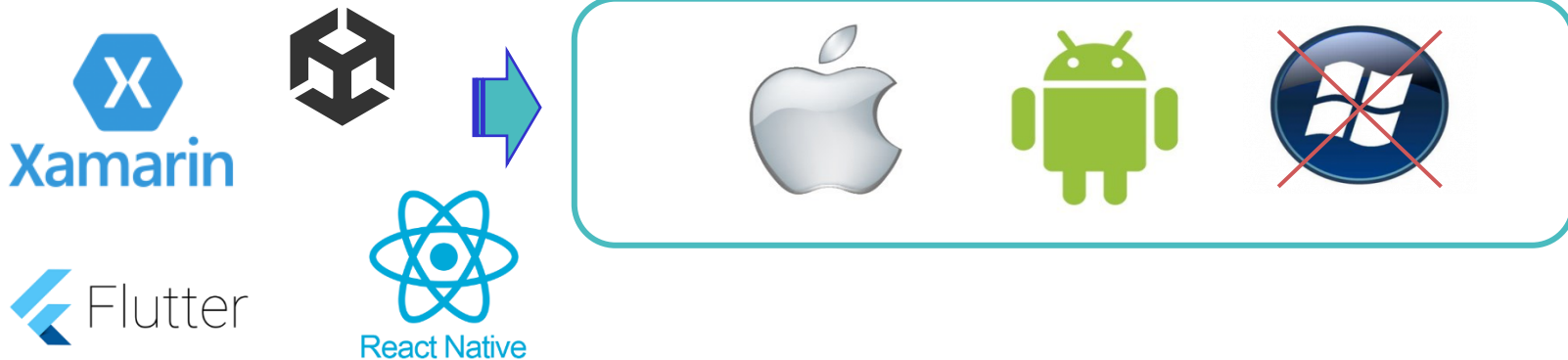
Avec Java

- ❑ Apparition des Midlets
- ❑ Mais un accès restreint au matériel

UN PETIT HISTORIQUE

AUJOURD'HUI

- ~~Windows Mobile – Palm Pre - iOS - Android – HarmonyOS~~
- Environnements de développement riches et simplifiés
- Hors Android et Cordova, ce sont des systèmes propriétaires
- Applications natives = Application tierces



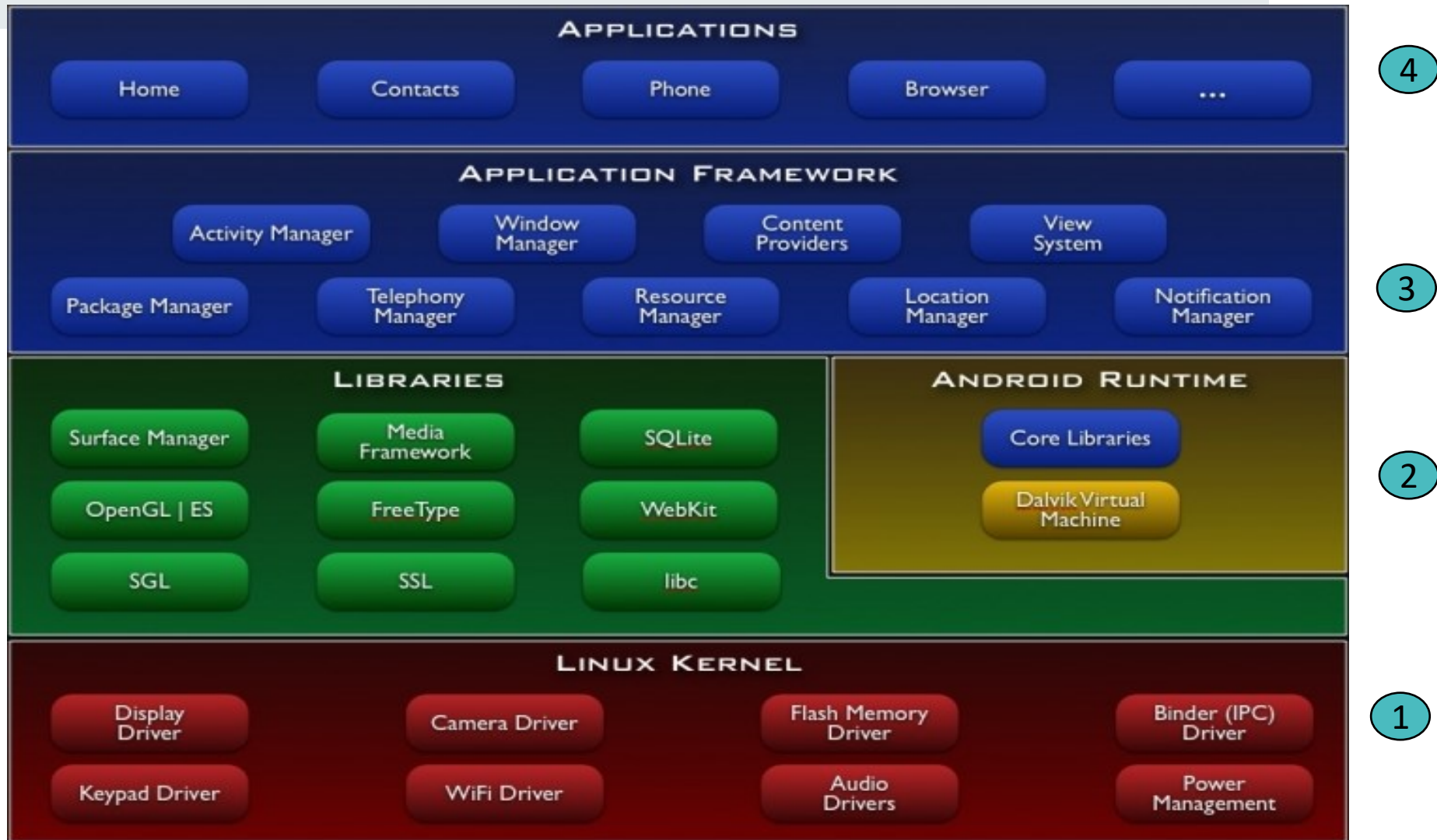
ANDROID N'EST PAS :

- Une implémentation de Java ME
- Un élément de Linux Phone Standards Forum (LiPS) ou d'Open Mobile Alliance
- Une simple couche applicative
- Un téléphone mobile
- La réponse de Google à l'iPhone

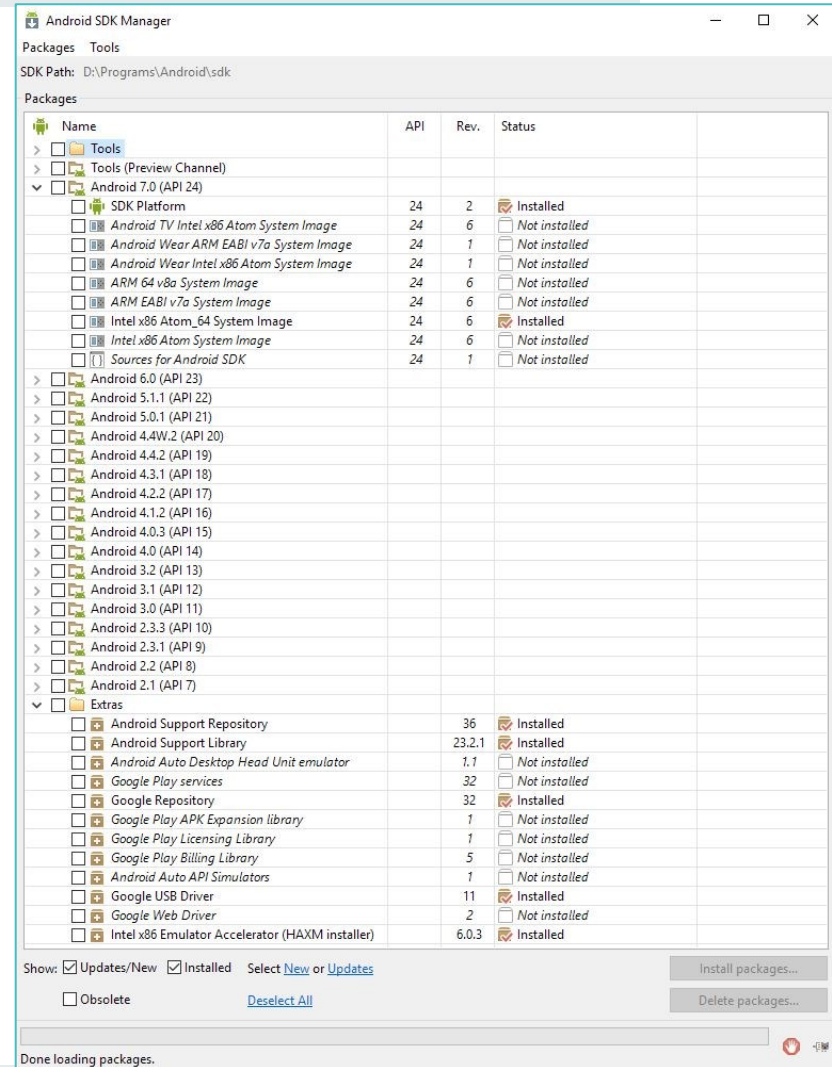
ANDROID EST :

- Un système d'exploitation open-source
- Des équipements et applications développés pour ce système dont un kit de développement
- Des applications ... très nombreuses.

ARCHITECTURE ANDROID



- Les API Android
- Des outils de développement
- Le Virtual Device Manager et l'Emulator
- Une documentation complète
- Des exemples de code
- Un support en ligne



- Noyau Linux
- Bibliothèques (en C/C++)
 - Multimédia
 - Gestionnaire de surface
 - Graphique 2D et 3D
 - SQLite pour avoir une base de données en natif
 - Un navigateur web et la sécurité inhérente

- Le moteur d'exécution Android
 - Bibliothèques de base
 - Le framework applicatif
 - La couche applicative
 - Machine virtuelle Dalvik ou ART

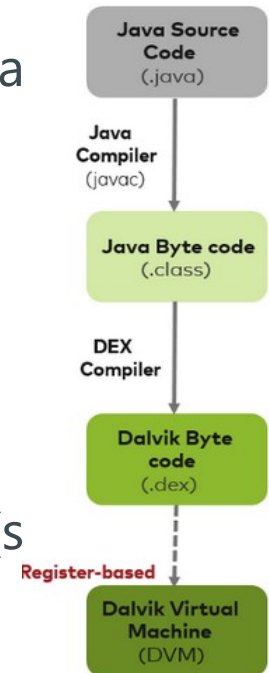
DALVIK (ANCIENNE VM POUR ANDROID)

■ Dalvik est 2 fois plus rapide qu'une VM Java classique temps !)

- Nécessite une conversion du bytecode en pseudo code machine
- L'ensemble des classes compilées dans un seul fichier cla qui peut être directement mappé en mémoire

■ Basée sur des registres (non sur une pile)

- Grand nombre de registres
- Couple de registres utilisés pour le (s) double(s) et long(s)
- Compilateur exploitant au maximum les registres



ART (LA NOUVELLE VM)

Les améliorations à l'utilisation de ART sont :

- Ahead-of-Time (AOT) compilation à l'installation d'une application
- Amélioration de la « Garbage Collection »
- Tracer l'exécution d'une appli en évitant de trop ralentir l'appli
- Plus d'aide au debug (notamment pour le GC)
- Diagnostics des exceptions plus précis

Problème de ART :

- L'utilisation d'outils de modification des fichiers « .dex » passe en général avec Dalvik mais pas avec ART
- ~~ART reste donc encore expérimental~~

ART remplace Dalvik depuis Android 5.0

- Application précompilée entièrement lors de sa première installation
- Exécution 2 fois plus rapide qu'avec Dalvik
- Une meilleure autonomie

Problème

- Temps d'installation rallonge
- Taille des APK augmentée

VERSIONS

- Alpha : 2007, Beta : novembre 2007
- 1.0 : septembre 2008 (le market, les GoogleApps, ...)
- 2.0 : octobre 2009 (optimisations, tailles d'écrans, HTML5, ...)
- 2.2 Froyo : mai 2010 (dpi plus élevé, Flash, Google Play, ...)
- 2.3 Gingerbread : décembre 2010 (NFC, Google Wallet, ...)
- 3.0 Honeycomb : février 2011 (les tablettes)
- 4.0 Ice Cream Sandwich : octobre 2011 (smartphones + tablettes)
- 4.1 à 4.3 Jelly Bean : juillet 2012 (AndroidBeam, multi-utilisateurs, multi-écrans, ...)
- 4.4 KitKat : septembre 2013 (HCE, rigueur sur les accès cartes SD, Android Wear / TV, ...)
- 5.0 Lollipop : novembre 2014 (refonte de l'UI, ART, ...)

VERSIONS

- 6 Marshmallow : octobre 2015 (améliorations : écran d'accueil, barre d'état, notifs, paramètres systèmes, ...)
- 7 Nougat : août 2016
- 8 Oreo : août 2017 (amélioration multitâche, batterie et sécurité)
- 9 Pie : été 2018 (I.A. intégrée au système, gestion batterie améliorée ...)
- 10 Q : été 2019 (5G, mode sombre, améliorations : notif, vie privée, sécurité, rapidité ...)
- 11 Red Velvet Cake : Sept. 2020 (autorisation à usage unique, permissions étendues, écrans pliables, bulles de conversation ...)
- 12 Snow Cone : Oct. 2021 (vie privée et sécurité, UI Material You, coins arrondis !)
- 13 Tiramisu : Sept. 2022 (langue par appli, meilleures notifs., vie privée ...)
- 14 Upside Down Cake : Octobre 2023 (langues par appli, accessibilité, LLM ...)



VERSIONS DE JAVA

- À partir de Android 14 (API 34) : Java 17
- A partir de Android 12 (API 32) : Java 11 obligatoire
- Avant : Java 11 ou Java 8
- Ancien, pas de chance si c'est le cas : Java 7 (attention, pas de Lambda)
- Très ancien, surtout de les déterrés pas : Java 6

Android Studio basé sur IntelliJ IDEA

- <http://developer.android.com/sdk/index.html>

Détails d'installation :

- <http://developer.android.com/sdk/installing.html>

LE SDK MANAGER



Settings

Settings

Appearance & Behavior

Appearance

Menus and Toolbars

System Settings

HTTP Proxy

Data Sharing

Date Formats

Updates

Process Elevation

Passwords

Android SDK

Memory Settings

Notifications

Quick Lists

Path Variables

Keymap

Editor

Build, Execution, Deployment

Kotlin

Tools

Advanced Settings

Appearance & Behavior > System Settings > Android SDK

Manager for the Android SDK and Tools used by the IDE

Android SDK Location: C:\Users\joan\AppData\Local\Android\Sdk [Edit](#) [Optimize disk space](#)

SDK Platforms SDK Tools SDK Update Sites

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, the IDE will automatically check for updates. Check "show package details" to display individual SDK components.

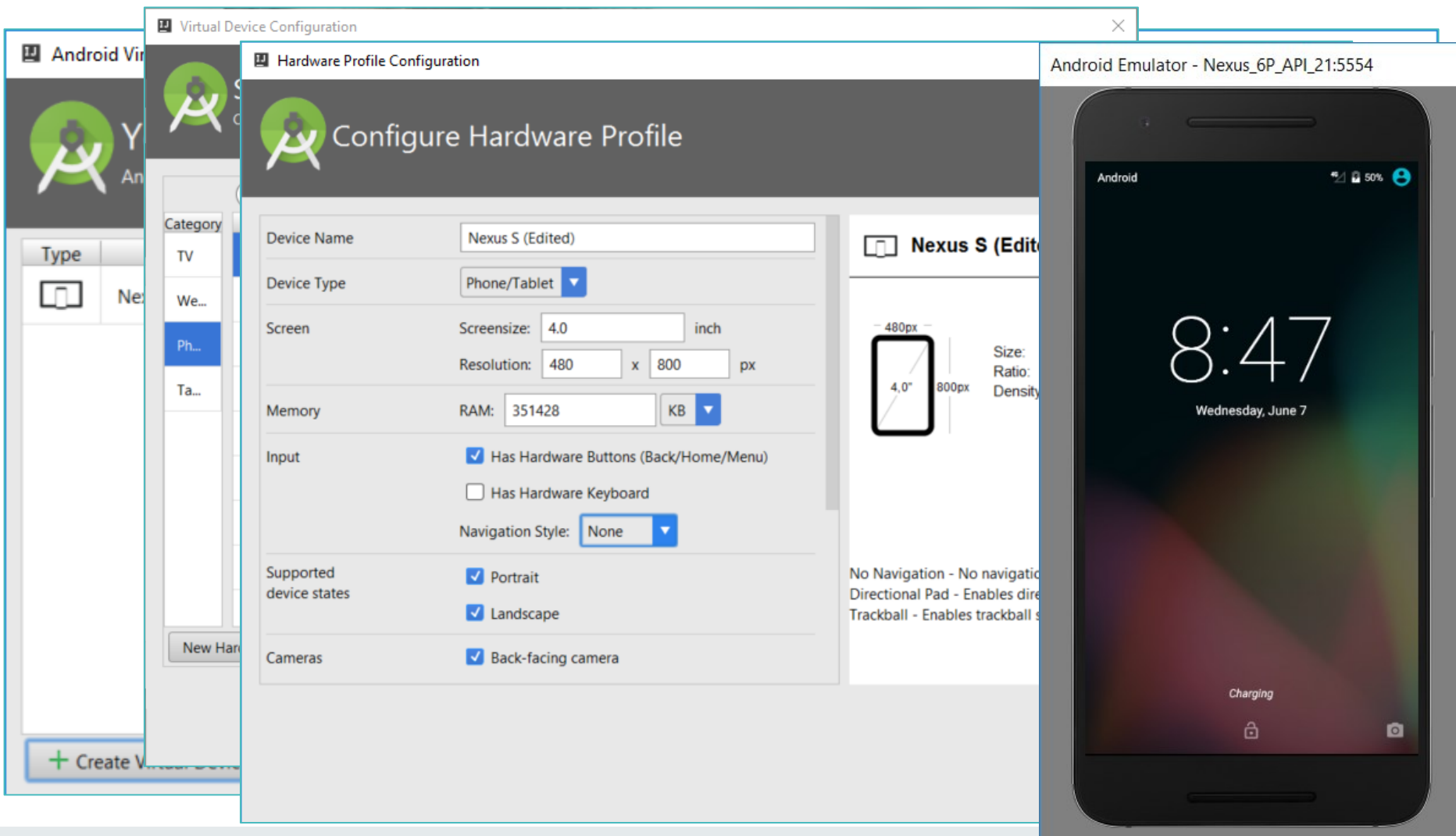
Name	API Level	Revisi...	Status
<input type="checkbox"/> Google APIs ARM 64 v8a System Image	31	10	Not installed
<input type="checkbox"/> Google APIs Intel x86 Atom_64 System Image	31	10	Not installed
<input type="checkbox"/> Google Play Intel x86 Atom_64 System Image	31	9	Not installed
<div>Android 11.0 (R)</div>			
<input checked="" type="checkbox"/> Android SDK Platform 30	30	3	Installed
<input checked="" type="checkbox"/> Sources for Android 30	30	1	Installed
<input type="checkbox"/> Android TV Intel x86 Atom System Image	30	3	Not installed
<input type="checkbox"/> China version of Wear OS 3 - Preview ARM 64 v8a System Image	30	10	Not installed
<input type="checkbox"/> China version of Wear OS 3 - Preview Intel x86 Atom System Image	30	10	Not installed
<input type="checkbox"/> Wear OS 3 ARM 64 v8a System Image	30	11	Not installed
<input type="checkbox"/> Wear OS 3 Intel x86 Atom System Image	30	11	Not installed
<input type="checkbox"/> AOSP ATD ARM 64 v8a System Image	30	1	Not installed
<input type="checkbox"/> AOSP ATD Intel x86 Atom System Image	30	1	Not installed
<input type="checkbox"/> ARM 64 v8a System Image	30	1	Not installed
<input checked="" type="checkbox"/> Intel x86 Atom_64 System Image	30	10	Installed
<input type="checkbox"/> Google TV Intel x86 Atom System Image	30	3	Not installed
<input type="checkbox"/> Google APIs ARM 64 v8a System Image	30	11	Not installed
<input type="checkbox"/> Google APIs Intel x86 Atom System Image	30	10	Not installed
<input type="checkbox"/> Google APIs Intel x86 Atom_64 System Image	30	11	Not installed

☒ Hide Obsolete Packages ☒ Show Package Details

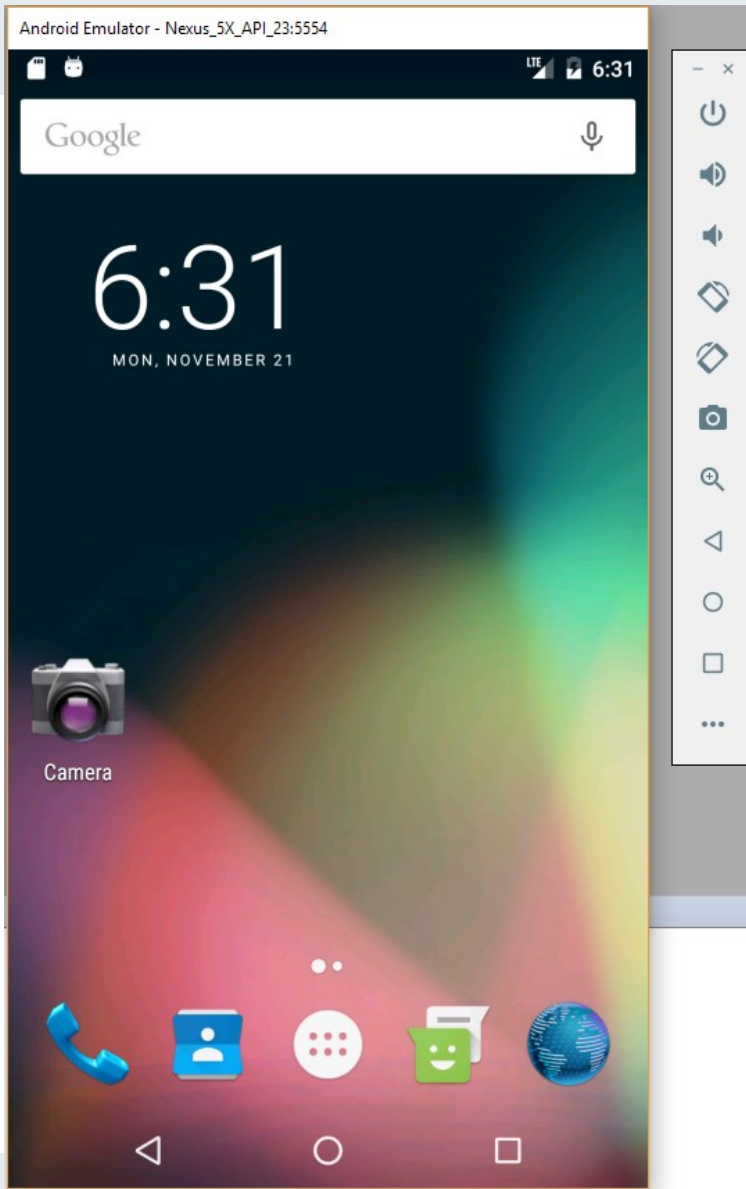
Project-level settings will be applied to new projects

OK Cancel Apply

ANDROID VIRTUAL DEVICE MANAGER



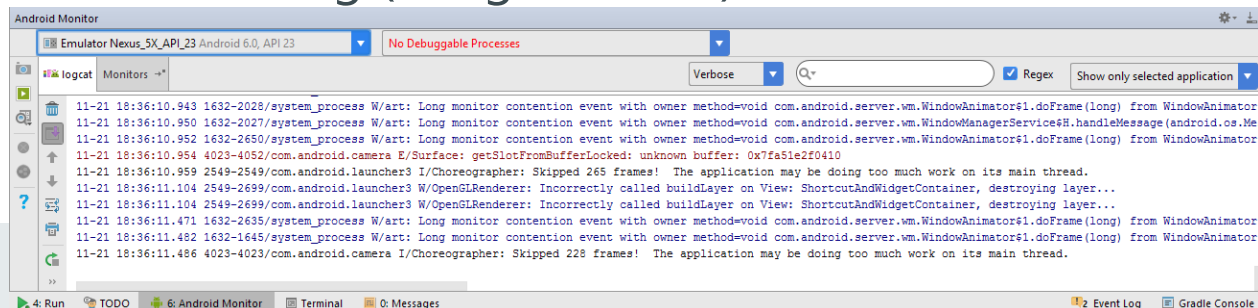
L'ÉMULATEUR ANDROID



- Emulateur de plus en plus puissant
- Il existe des alternatives viables
 - Visual Studio / Xamarin (avec l'environnement de développement)
 - Bluestacks
 - Memu
 - Genymotion
 - Nox
 - WSA (fin en mars 2025)

OUTIL ADB

- **adb devices** : permet de connaître les devices (physiques ou virtuelles) connectés en USB
- **adb help** : la liste des commandes possibles avec adb
- **adb shell** : permet de lancer un shell sur le device
- **adb push** : permet de copier un fichier de l'ordinateur vers le device
- **adb pull** : permet de copier un fichier du device vers l'ordinateur
- **adb install** : permet d'installer une application apk sur le device
- **adb logcat** : affiche le log (intégré à l'IDE)



- Android encourage la réutilisation de composants (et applications)
- Activity Manager
- Views
- Notification Manager
- Content Providers
- Resource Manager



LES TYPES D'APPLICATIONS

- Application de premier plan
- Application d'arrière-plan
- Intermittente
- Widget

CRÉATION D'UN PROJET ANDROID (INTELLIJ IDEA + PLUGIN ANDROID)

The image displays three sequential screenshots from the IntelliJ IDEA IDE, illustrating the process of creating a new Android project.

First Screenshot: New Project Dialog
The 'New Project' dialog is shown. On the left, a list of project types includes Java, Java Enterprise, JBoss, J2ME, Clouds, Spring, Java FX, **Android** (highlighted), IntelliJ Platform Plugin, Spring Initializr, Maven, Gradle, Groovy, Griffon, Grails, Scala, Kotlin, Static Web, Node.js and NPM, Flash, and Empty Project. On the right, under 'Select the form', the 'Phone and Tablet' option is checked. The 'Minimum SDK' field is visible.

Second Screenshot: Add an activity to Mobile Dialog
The 'Add an activity to Mobile' dialog is shown. It presents three options: 'Blank Activity' (selected), 'Empty Activity', and 'Fullscreen Activity'. Each option is accompanied by a small preview image of the activity's layout.

Third Screenshot: Project File Explorer
The 'Project' file explorer on the right side of the IDE is shown. It displays the file structure of the 'HelloWorld' project. The files and folders listed are: `.gradle`, `.idea`, `app` (containing `build`, `libs`, `src`), `androidTest`, `main` (containing `java`, `res`, and `AndroidManifest.xml`), `test`, `.gitignore`, `build.gradle` (highlighted with a red rectangle), `proguard-rules.pro`, `gradle`, `.gitignore`, `build.gradle`, `gradle.properties`, and `gradlew`.

CRÉATION D'UN PROJET ANDROID (ANDROID STUDIO > NEW PROJECT)

Empty Views Activity

Creates a new empty activity

No Activity

Basic Views

Navigation Drawer Views Activity

Responsive Views Activity

Game Activity (C++)

Name: MyApp

Package name: fr.ensicaen.myapp

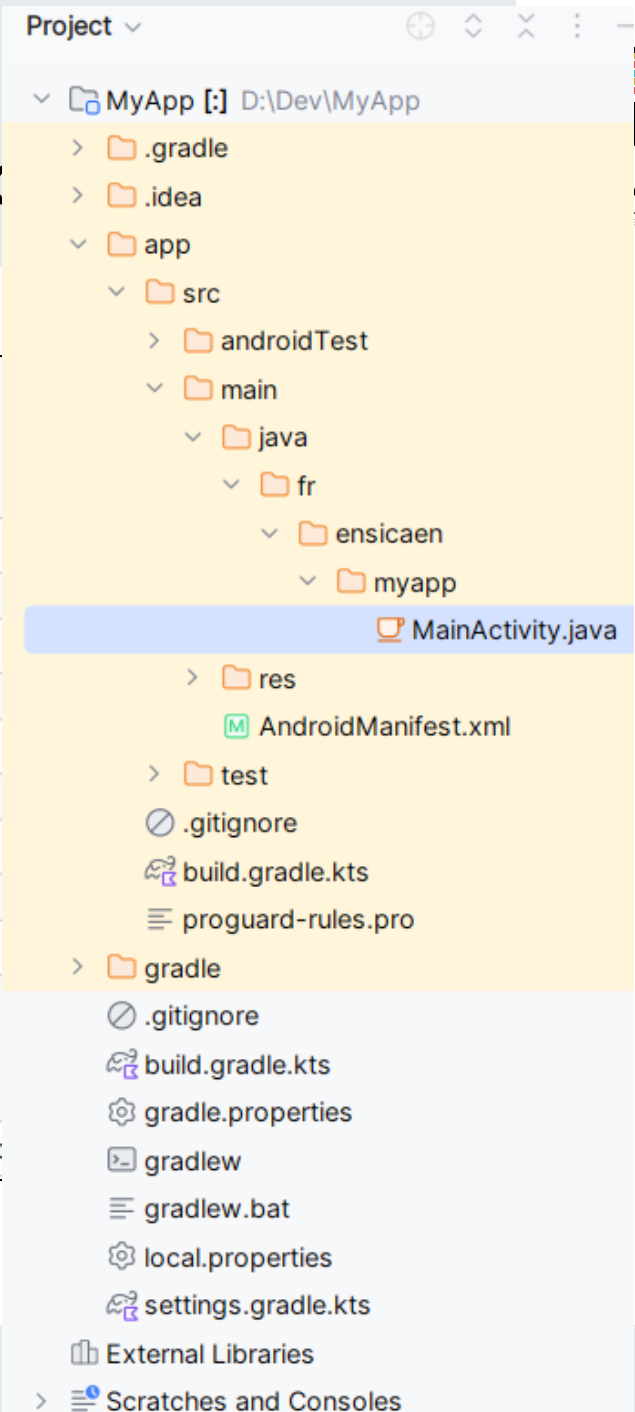
Save location: D:\Dev\MyApp

Language: Java

Minimum SDK: API 28 ("Pie"; Android 9.0)

Build configuration language: Kotlin DSL (build.gradle.kts) [Recommended]

Your app will run on approximately 89,6%
[Help me choose](#)



CONFIGURATION DE LANCEMENT

Run/Debug Configurations

Name: HelloWorld

General Miscellaneous

Module: app

Installation Options

Deploy: Default APK

Install Flags: Options to 'pm install' command

Launch Options

Launch: Default Activity

Device Chooser

Choose a running device

Device	State	Compatible	Serial Number
Motorola Nexus 6 Android 7.1.1, API 25	Online	Yes	ZX1G523XVJ

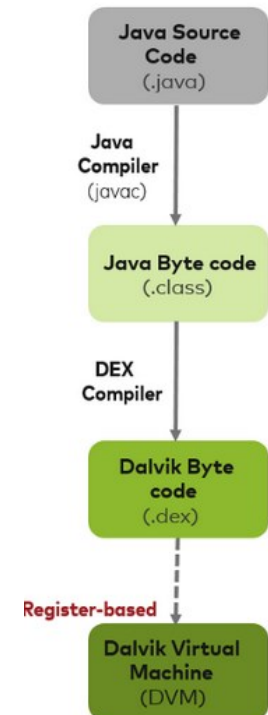
Launch emulator

Android virtual device: Nexus 6P API 21

Use same device for future launches

OK Cancel Help

- Compile les classes et génère un exécutable de type android (.dex)
- Crée un package déployable Android (.apk) à partir de l'exécutable et des ressources externes
- Démarre l'appareil virtuel (si ce n'est pas déjà le cas)
- Installe l'application sur cet appareil
- Démarre l'application



HELLO WORLD

package bundle.test.com.hellow;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity **extends** Activity {

@Override

protected void onCreate(Bundle savedInstanceState)

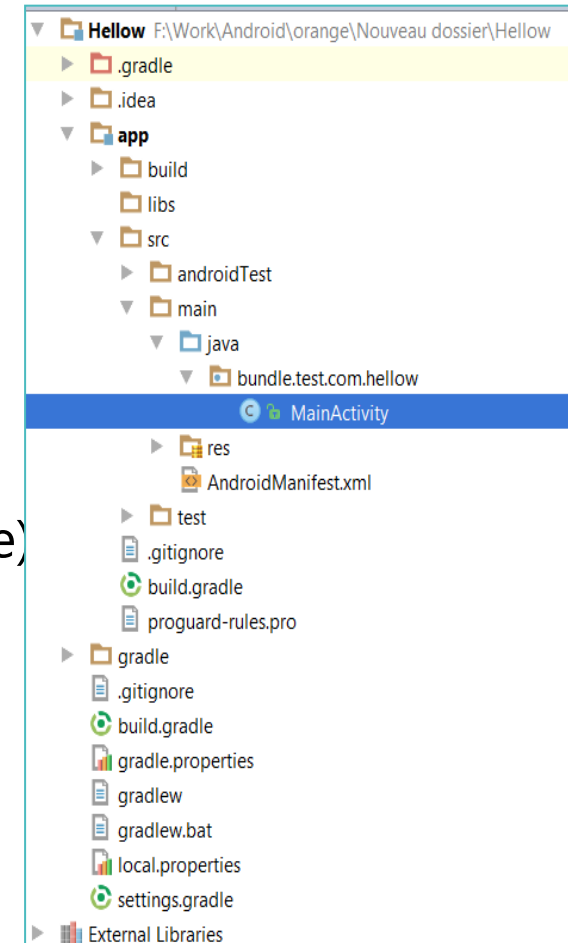
{

super.onCreate(savedInstanceState);

setContentView(R.layout.**activity_main**);

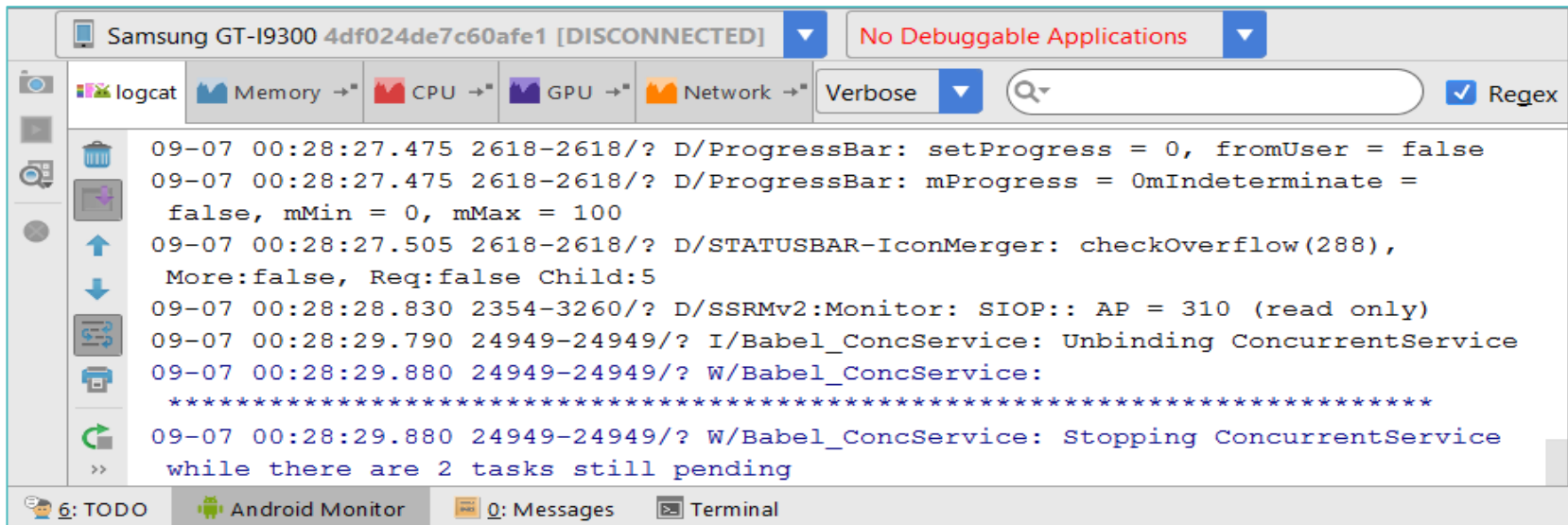
}

}



ADB ET LOGCAT

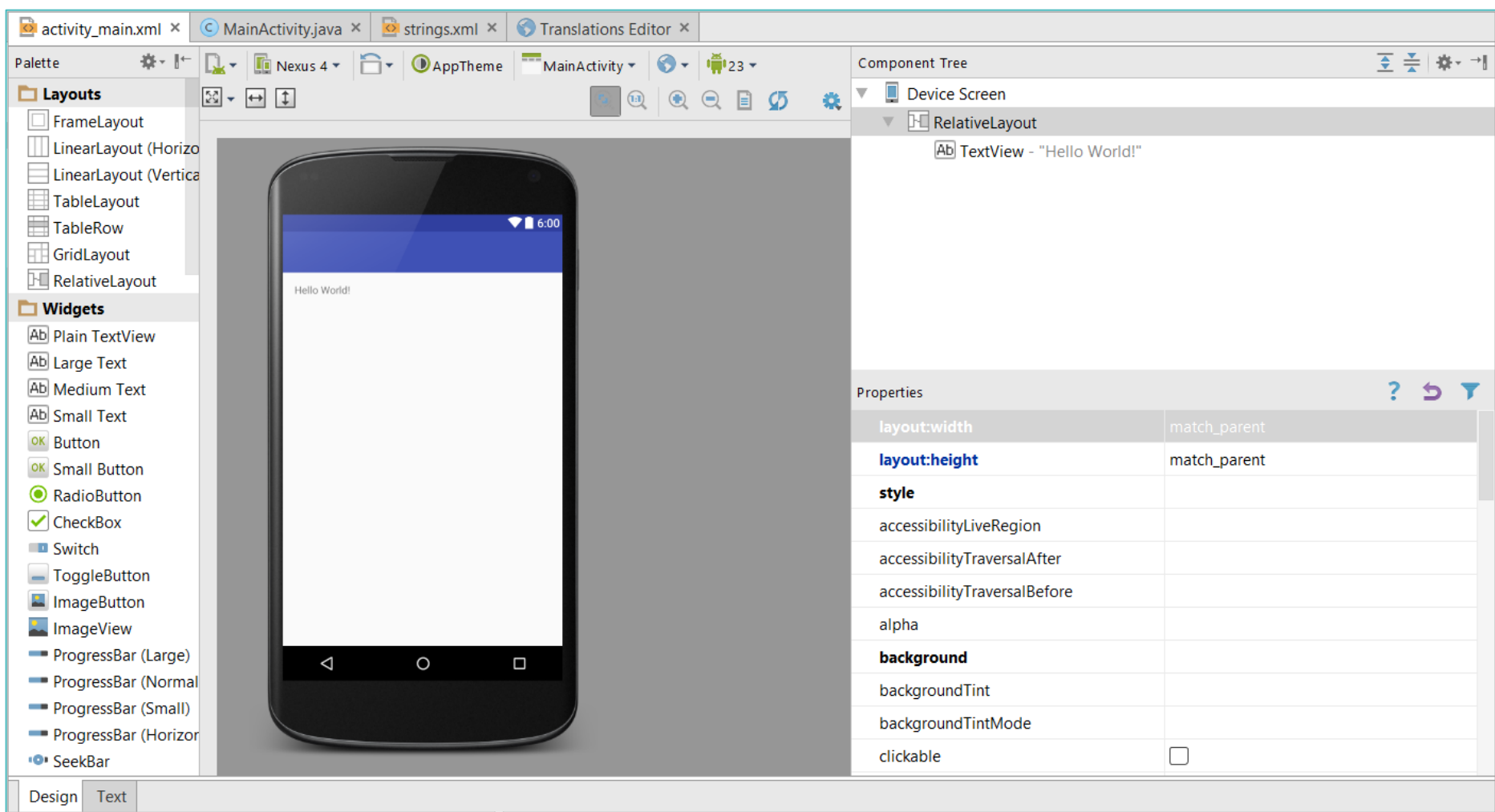
- adb start-server
- adb kill-server
- adb devices
- adb logcat (commandes et filtres, qu'on utilisera plutôt en intégré à l'IDE)





Le visuel de l'application

ÉDITEUR GRAPHIQUE



■ menu/

- Fichiers XML qui décrivent des menus de l'application

■ raw/

- Fichiers propriétaires (mp3, pdf, ...)

■ values/

- Fichiers XML pour différentes sortes de ressources (textes, styles, couleurs, dimensions, ...)

■ xml/

- Divers fichiers XML qui servent à la configuration de l'application (propriétés, infos BDD, ...)

■ anim/

- Fichiers XML qui sont compilés en animation interpolée

■ color/

- Fichiers XML décrivant des listes d'états-couleurs pour les vues

■ drawable/mipmap

- Fichiers bitmap (PNG, JPEG, or GIF), ou fichiers XML qui décrivent des objets à dessiner

■ layout/

- Fichiers XML compilés en vues (écrans, fragments, ...)

■ navigation/

- Fichiers descriptifs du parcours utilisateur

- LinearLayout : disposition simple horizontale ou verticale
- RelativeLayout : les éléments sont disposés les uns par rapport aux autres ou par rapport au bord de l'écran
- ConstraintLayout : une version plus récente et complète du RelativeLayout
- TableLayout : tableau
- FrameLayout : une seule vue est affichée à l'écran
 - ScrollView : c'est une FrameLayout mais qui permet de rendre la vue qu'elle contient « scrollable » si elle ne l'est pas.

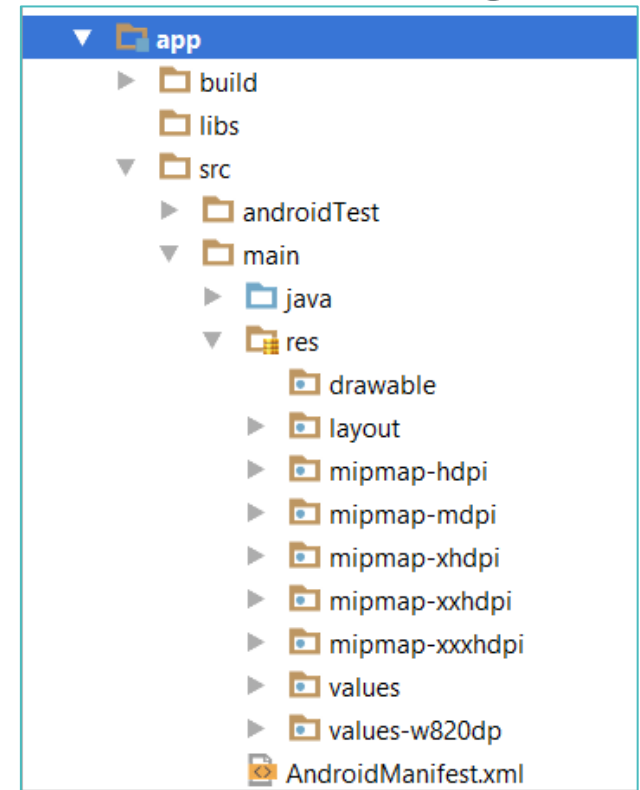
<http://developer.android.com/guide/topics/resources/more-resources.html#Dimension>

- **dip** = dp : Density-independent Pixels : basé sur un écran en 160 dpi de façon à ce que 1 dp = 1 pixel sur un écran de cette densité.
- **sp** : Scale-independent Pixels : utilisé pour le texte car c'est comme « dp » mais avec en plus la prise en compte de la taille de la fonte utilisée.
- **px** : un pixel réel sur l'écran, quasiment jamais utilisé car les écrans sont trop différents. $px = dp * (dpi / 160)$
- **mm** : Millimètres
- **pt** : Points : 1/72 inch basé sur la taille physique de l'écran, ces 3 unités ne sont utilisées que dans des cas spécifiques où l'on veut une dimension physique précise.

http://developer.android.com/guide/practices/screens_support.html

On peut via les répertoires de ressources s'adresser à des configurations différentes

- Des tailles d'écran différentes
- Des densités de pixel différentes
- Des appareils différents : Wear, TV, Auto
- Des pliables



- Utiliser **wrap_content**, **match_parent** ou les unités en **dp** quand on spécifie des dimensions dans un fichier de **layout** XML
- Ne pas utiliser des valeurs en **pixel** codées en dur
- Ne pas utiliser l'**AbsoluteLayout** (deprecated)
- Fournir des ressources alternative (pour les bitmaps notamment) qui correspondent à des densités d'écran différentes.
- Quand on veut utiliser les poids (weight), utiliser **0dp** comme dimension, cela accélère le traitement. (on peut voir ça dans le guide du LinearLayout)
- **0dp** avec ConstraintLayout permet de remplir la place.

EXEMPLE DE COMPOSANT : TEXTVIEW

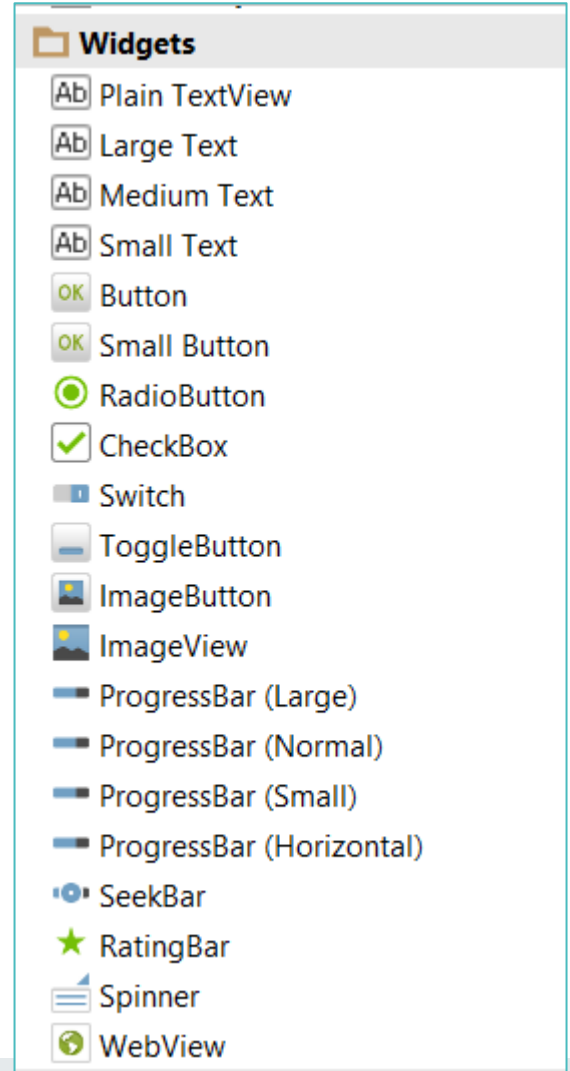
Drag & drop depuis la palette de composants.

On peut ensuite voir apparaître dans le fichier de ressource :

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

Par la programmation:

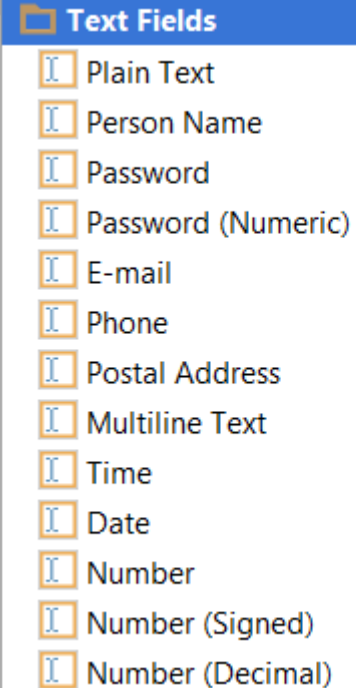
```
TextView texte= findViewById(R.id.txt_hello);  
texte.setText("Hello world!");
```



```
<EditText  
    android:text="Hello world!"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/edittext"/>
```

Par la programmation:

```
EditText edit= findViewById(R.id.edittext);  
edit.setText("TextView en java!");  
String str = edit.getText().toString();
```

- 
- A list of Android text field types, each preceded by a small icon of a text box with a cursor. The list is titled 'Text Fields' in a blue header.
- Text Fields
 - Plain Text
 - Person Name
 - Password
 - Password (Numeric)
 - E-mail
 - Phone
 - Postal Address
 - Multiline Text
 - Time
 - Date
 - Number
 - Number (Signed)
 - Number (Decimal)

AJOUT D'UNE RESSOURCE TEXTE

- ▶ manifests
- ▼ java
 - ▼ fr.ensicaen.gui_test
 - MainActivity

- ▶ fr.ensicaen.gui_test (androidTest)
- ▶ fr.ensicaen.gui_test (test)

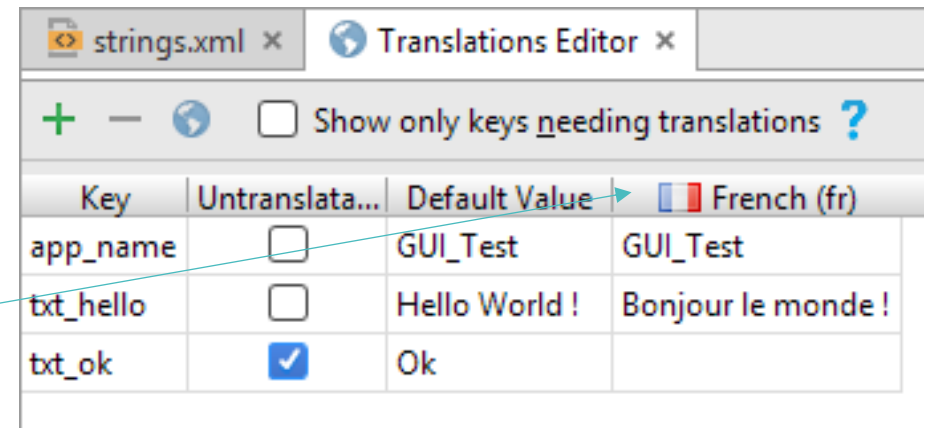
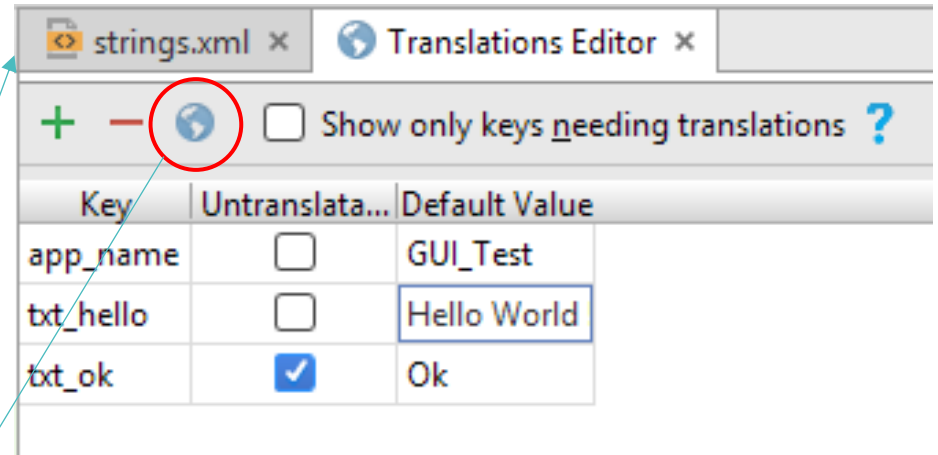
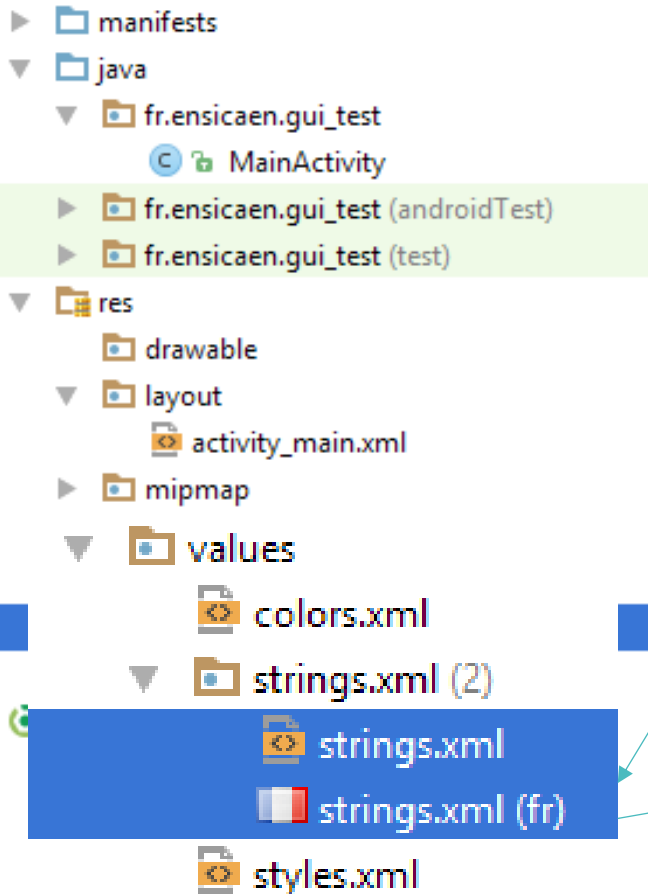
- ▼ res
 - drawable
 - ▼ layout
 - activity_main.xml

- ▶ mipmap
- ▼ values
 - colors.xml
 - strings.xml
 - styles.xml

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

```
<resources>
    <string name="app_name">GUI_Test</string>
    <string name="txt_hello">Hello World</string>
</resources>
```

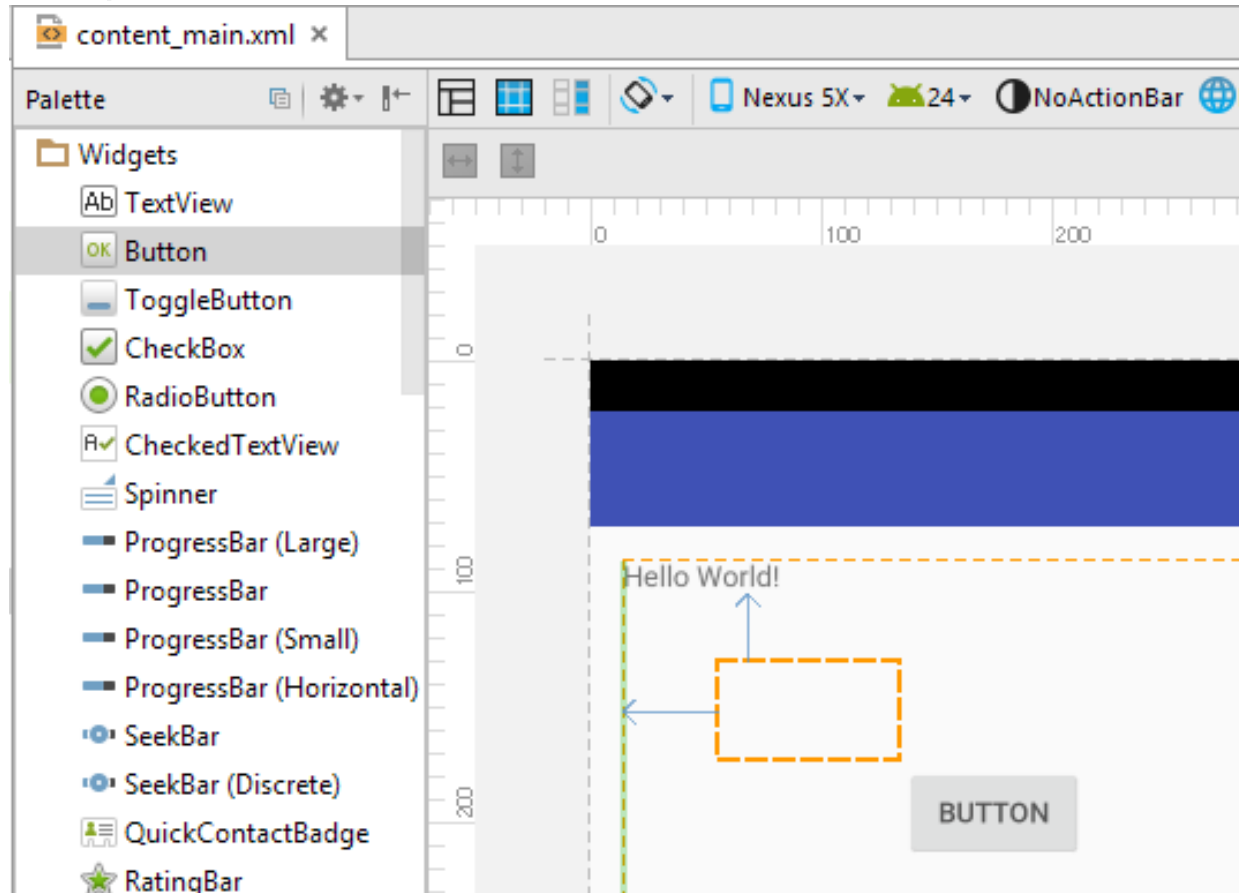
GESTION MULTI-LANGUES



AJOUT D'UN BOUTON

Placer le bouton sur le gestionnaire de disposition.

Un simple Drag & Drop suffit.

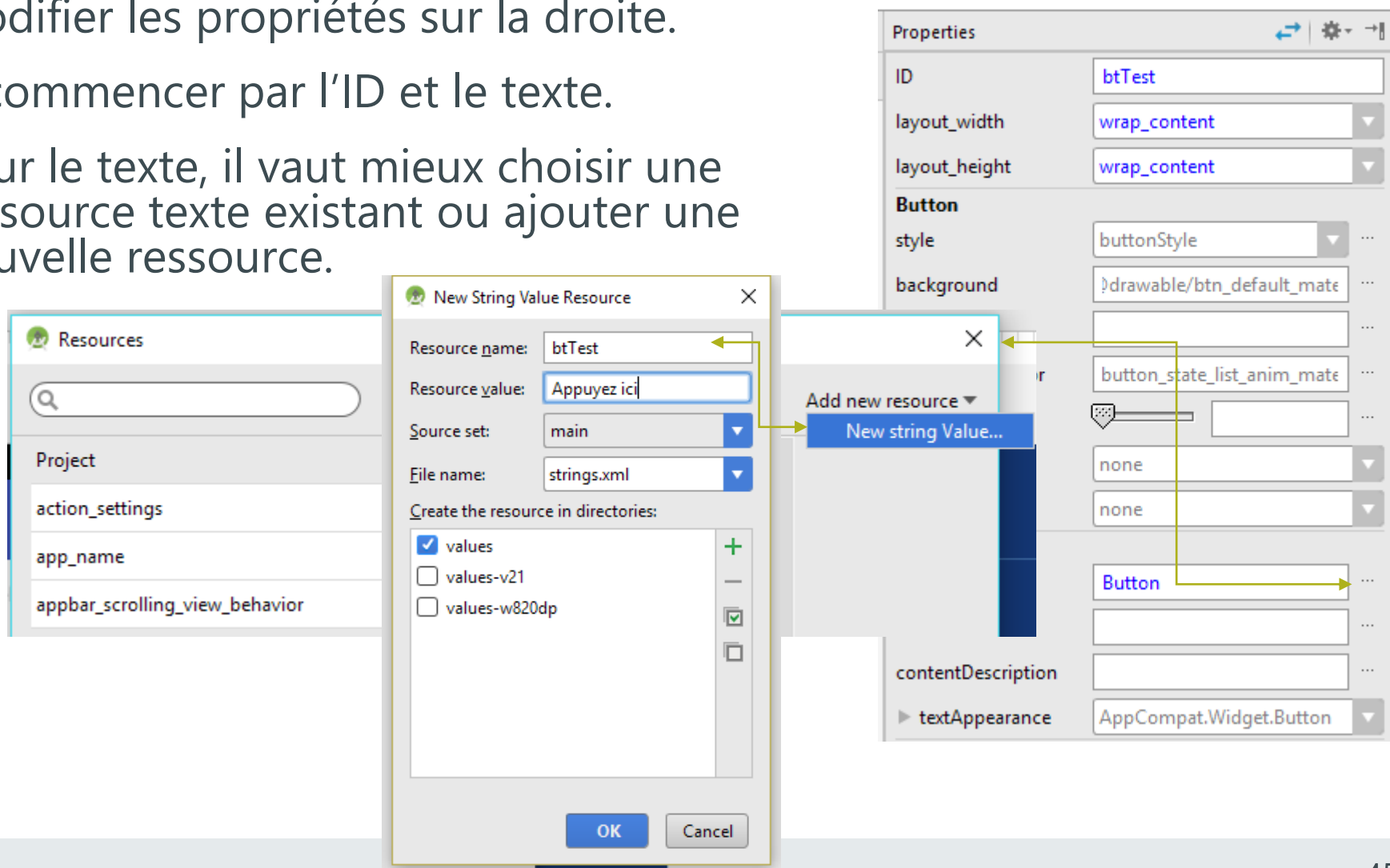


AJOUT D'UN BOUTON (SUITE)

Modifier les propriétés sur la droite.

A commencer par l'ID et le texte.

Pour le texte, il vaut mieux choisir une ressource texte existant ou ajouter une nouvelle ressource.



LES BOUTONS (1ÈRE VERSION)

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

    <TextView ... />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content "
        android:text=« @string/txtMyButton"
        android:id="@+id/btMyButton« />
</LinearLayout>
```

LES BOUTONS : AVEC CLASSE ANONYME

```
package bundle.test.com.hellow;  
import android.widget.Button;  
import android.widget.Toast;
```

```
public class MainActivity extends Activity {
```

```
    @Override
```

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

```
        Button b = (Button)findViewById(R.id.button);  
        b.setOnClickListener(new View.OnClickListener() {
```

```
            @Override
```

```
            public void onClick(View v) {  
                Toast.makeText(v.getContext(), "Stop !", Toast.LENGTH_LONG).show();  
            }
```

```
        });
```

```
    }
```

```
}
```

LES BOUTONS : AVEC CLASSE ANONYME + LAMBDA

```
import android.widget.Button;  
import android.widget.Toast;
```

```
public class MainActivity extends Activity {
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
```

```
        Button b = findViewById(R.id.button);
```

```
        b.setOnClickListener((view) ->
```

```
            { Toast.makeText(v.getContext(), "Stop !", Toast.LENGTH_LONG).show(); }  
        );
```

```
    }  
}
```

LES BOUTONS : VERSION CENTRALISÉE

```
package bundle.test.com.hellow;  
import android.widget.Button;  
import android.widget.Toast;
```

```
public class MainActivity extends Activity implements OnClickListener {  
    private Button btn;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        btn = (Button)findViewById(R.id.button);  
        btn.setOnClickListener(this) ;  
    }  
    @Override  
    public void onClick(View v) {  
        if (v == btn) {  
            Toast.makeText(v.getContext(), "Stop !", Toast.LENGTH_LONG).show();  
        }  
    }  
}
```


LES BOUTONS : EN LIEN AVEC LE XML

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

    <TextView ... />

    <Button
        android:layout_width="200dp"
        android:layout_height="50dp"
        android:text="Bouton"
        android:onClick="clickBouton"
        android:layout_marginTop="57dp"
        android:layout_below="@+id/textView"
        android:layout_alignParentStart="true"/>
</RelativeLayout>
```

LES BOUTONS : EN LIEN AVEC LE XML

```
package bundle.test.com.hellow;
```

```
....
```

```
import android.widget.Button;
```

```
import android.widget.Toast;
```

```
public class MainActivity extends Activity {
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
```

```
    }
```

```
    public void clickBouton(View v)
```

```
    {
```

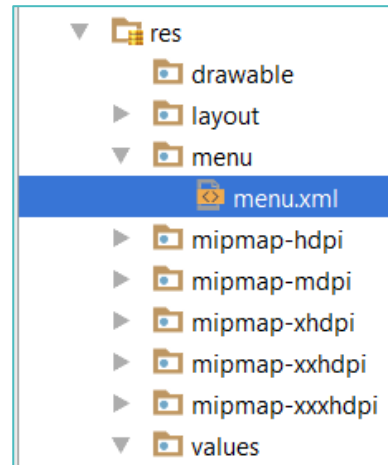
```
        Toast.makeText(v.getContext(), "Stop", Toast.LENGTH_LONG).show();
```

```
    }
```

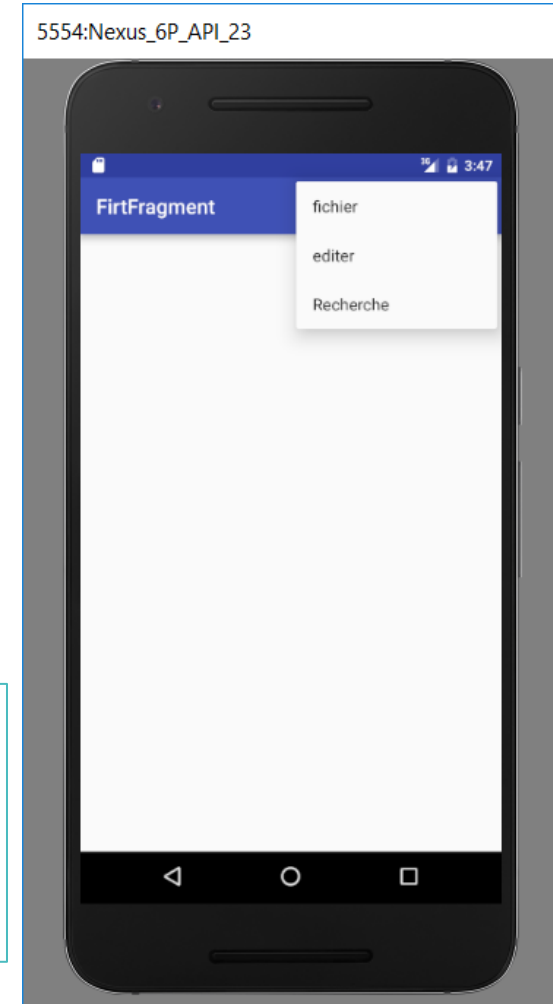
```
}
```

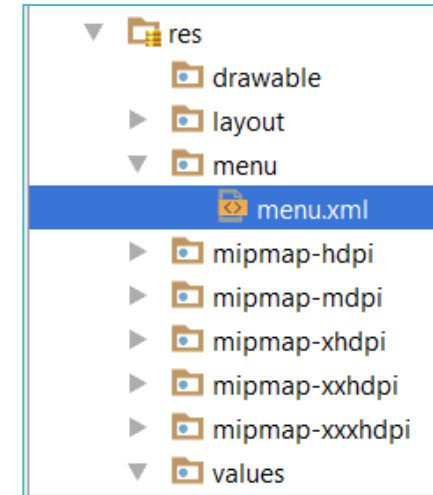
LES MENUS

Le menu de l'action bar est contrôlé par la méthode **onOptionsItemSelected(Menu menu)** qui charge un fichier menu depuis les ressources



```
public boolean onOptionsItemSelected(Menu menu)
{
    getMenuInflater().inflate(R.menu.menu, menu);
    return true;
}
```





```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
  <item android:id="@+id/fichierId" android:title="fichier" />
  <item android:id="@+id/editerId" android:title="editer" />
  <item android:id="@+id/rechercheld" android:title="Recherche"/>
</menu>
```

la méthode **onOptionsItemSelected()** permet de gérer les actions sur les boutons

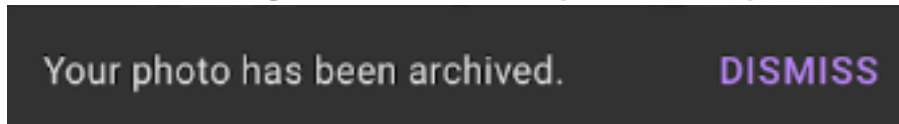
```
public boolean onOptionsItemSelected(MenuItem item)
{
    int id = item.getItemId();
    if (id == R.id.fichierId)
    {
        Toast.makeText(getApplicationContext(), "Fichier !", Toast.LENGTH_LONG).show();
    } else if (id == R.id.editerId)
    {
        Toast.makeText(getApplicationContext(), "Editer !", Toast.LENGTH_LONG).show();
    } else if (id == R.id.recherchedId)
    {
        Toast.makeText(getApplicationContext(), "Recherche !",
        Toast.LENGTH_LONG).show();
    }
    return super.onOptionsItemSelected(item);
}
```

LES MESSAGES « POPUP »

Toast : une petite « popup » contenant du texte

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

Snackbar : un message de basse priorité personnalisable et optionnellement interactif



[Snackbars - Material Design](#)

Banner : un message de moyenne priorité demandant une interaction

[Banners - Material Design](#)

Dialog : une boîte de dialogue complète de haute priorité

[Dialogs | Android Developers](#) - [Dialogs – Material Design](#)

Pour une application de type service : faire appel aux notifications.

findViewById est lent et ajoute beaucoup de code.

La « feature » ViewBinding permet de générer une classe de raccourcis pour les vues XML.

- Ajouter dans gradle `buildFeature { viewBinding = true }`
- l'attribut XML `tools:viewBindingIgnore="true"` permet de ne pas générer pour certaines vues

Dans le XML : `<TextView android:id="@+id/name" />`

Dans le code : `binding.name.text = "Lorem Ipsum"`

Plus d'info ici : [View binding | Android Developers](#)

Bibliothèque qui permet de lier des composants graphiques avec les sources de données de l'application.

- Ajouter dans gradle `buildFeature { dataBinding = true }`

- Dans le XML :

```
<data>
    <variable name="user" type="com.example.User"/>
</data>
...
<TextView ... android:text="@{user.firstName}"/>
```

- Dans le code :

```
binding.setUser(user);
```

Plus d'info ici : [Data Binding Library | Android Developers](#)



Le cycle de vie

7 MANIÈRES DE DÉPLOYER L'APPLICATION SUR UN SMARTPHONE

- Brancher l'appareil, activer le débogage USB, exécuter ...
- Copier le « .apk » sur le téléphone et le lancer via un gestionnaire de fichier (ex: Astro, File Manager, ...)
- Installer et lancer le « .apk » avec ADB depuis une console
- Exécuter l'application sur l'appareil Android via le Wi-Fi
- Application de partage en ligne (ex: DropBox, Google Drive, OneDrive)
- Lien HTML vers le fichier (lien dans un QR code par exemple)
- Play Store / Amazon Apps

ACTIVITÉ : CYCLE DE VIE

Une activité permet de gérer les interactions avec un utilisateur à travers une interface.

Le cycle de vie d'une activité est particulier

- Il est parfaitement décrit dans la documentation de la classe Activity.

Les raisons de la structure de ce cycle sont :

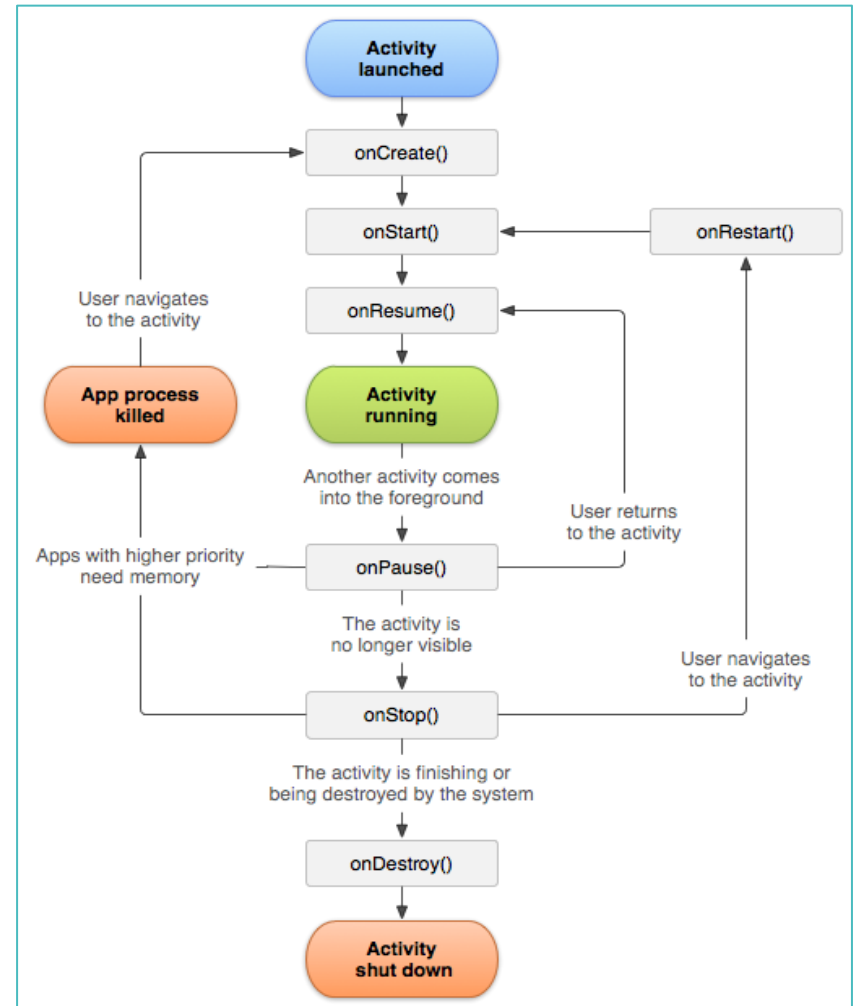
- Permettre une gestion optimale de la mémoire
- Prise en compte que le système Android gère une pile d'activités avec 4 états possibles

•**Active** : Au premier plan en cours d'exécution.

•**En pause** : Ayant perdue le focus, mais toujours visible (plus ou moins partiellement).

•**Stoppé** : Elle n'est plus visible.

•**Détruite** : En cours de destruction.



La prise en compte du cycle de vie d'une activité est réalisée en redéfinissant les méthodes :

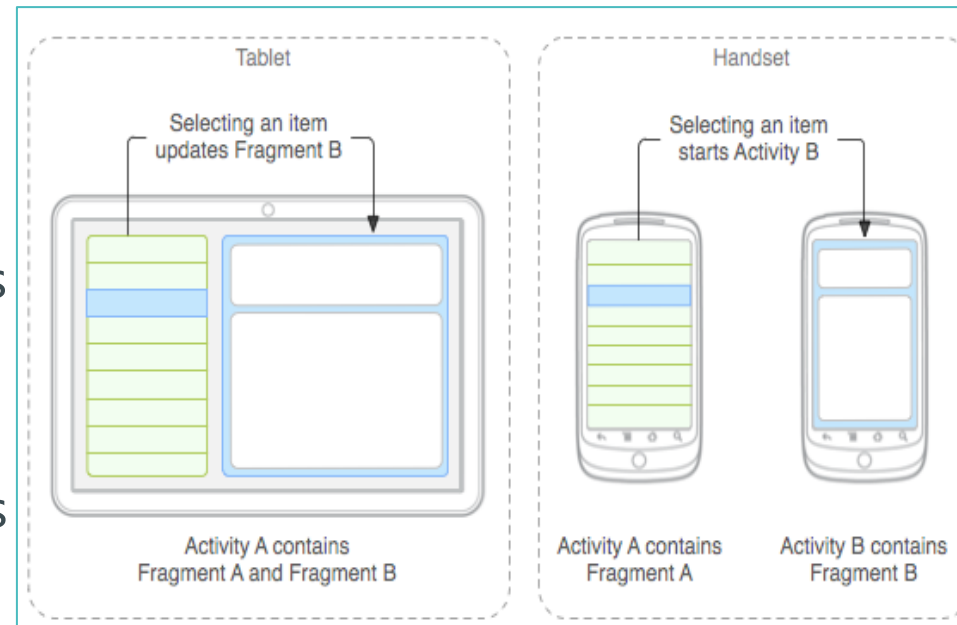
- onCreate() : démarrage de l'activité
- onRestart() : L'activité est relancée après avoir été stoppée.
- onStart() : L'activité devient visible, mais n'a pas le focus
- onResume() : L'activité a le focus avec l'utilisateur
- onPause() : L'activité perd le focus
- onStop() : L'activité n'est plus visible
- onDestroy() : L'activité va être détruite.
 - Arrêt volontaire ou destruction par le système.
- Le système Android peut détruire une application sans passer par onStop et onDestroy.
 - Les données capitales doivent être sauvegardé dans onPause

LES FRAGMENTS

De même que pour une Activity, un fragment à un layout en XML et une classe associée, cette classe doit hériter de la classe **android.app.Fragment**. Il est introduit depuis la version 3.0

Il permet:

- de découper un écran en plusieurs partie (ou en plusieurs fragments).
- de gérer les tablettes et les téléphones de la même façon



Un fragment est dépendant d'une activité:

- si l'activité passe en pause, les fragments aussi
- si l'activité est détruite, les fragments aussi

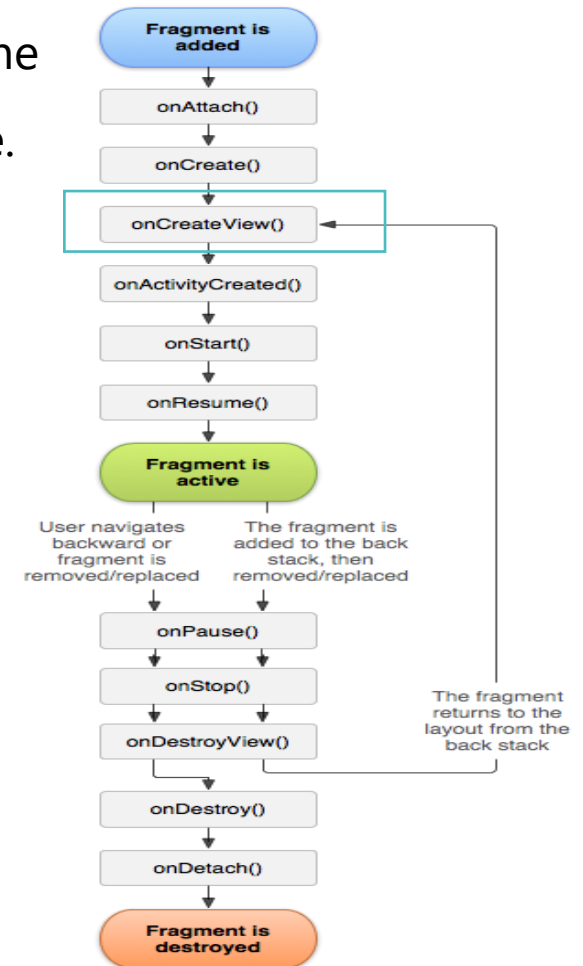
Il peut être statique ou dynamique

LES FRAGMENTS : CYCLE DE VIE

Le cycle de vie d'un fragments est très similaire et en même temps étroitement lié avec l'activité à laquelle il est attaché.

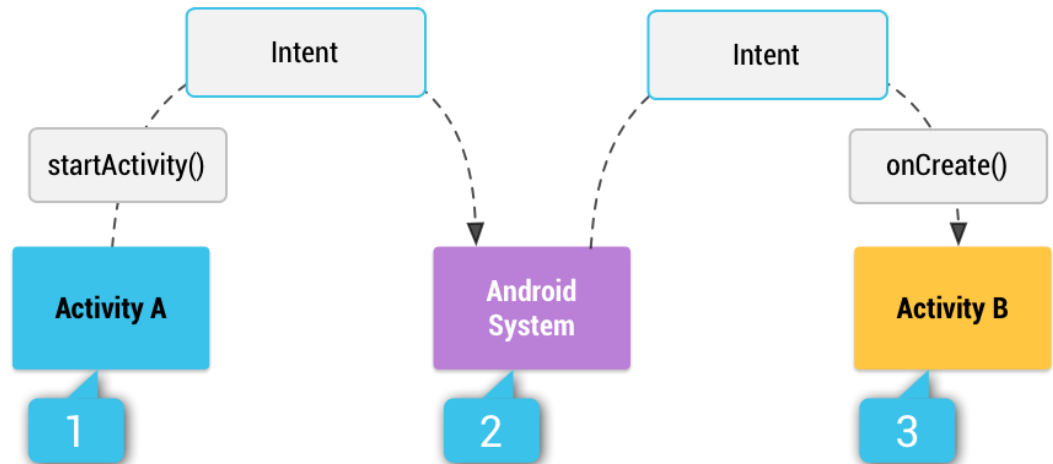
On retrouve toutes les méthodes callback d'une activité:

- ❑ onCreate,
- ❑ onStart,
- ❑ onResume,
- ❑ onPause,
- ❑ onStop,
- ❑ onDestroy.



- ❑ **onCreate** permet de créer les objets de la classe pour qu'ils soient instanciés une seule fois dans le cycle de vie du fragment ;
- ❑ **onCreate** ne crée pas l'interface graphique ;
- ❑ **onCreateView** crée l'interface graphique ;
- ❑ **onActivityCreated** permet de récupérer un pointeur vers l'activité ;
- ❑ **onDestroy** n'est pas toujours appelée (que ce soit pour l'activité comme pour le fragment) ;
- ❑ **onAttach** permet de récupérer une instance de l'activité parente, mais attention, elle n'a pas fini son initialisation (il vaut mieux attendre `onActivityCreated`).

- Explication : <http://developer.android.com/guide/components/intents-filters.html>



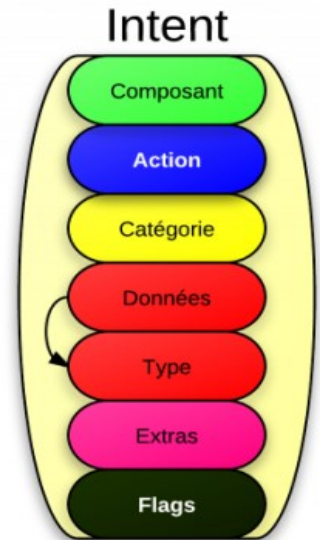
- Commencer par les intents les plus classiques :
<http://developer.android.com/guide/components/intents-common.html>

Un Intent est constitué de:

- action à réaliser
- donnée sur laquelle réaliser l'action sous forme d'URI (setData()) ou d'un type MIME (setType())
- Paramètre optionnels (EXTRA)

Création d'un Intent

- **Intent**(Context, Class<?>) pour l'appels explicite
- **Intent**(String action, URI) pour l'appel implicite
- **addCategory**(String category) ajout de catégories
- **putExtra**(String key,value)
- **setFlags**(flags) permission sur les données, relation activité/BackStack



APPELER UNE ACTIVITÉ EXPLICITEMENT

- Dans le code de mon activité source, je souhaite atteindre l'activité de destination nommée NouvelleActivite.
- Je veux lui passer un paramètre nommé « TEXT » qui contient une chaîne de caractères.

```
Intent intent = new Intent( this, NouvelleActivite.class);  
intent.putExtra("TEXT", editTextFormulaire.getText());  
startActivity(intent);
```

- Récupération du paramètre dans l'activité destination (dans le onCreate par exemple) :

```
String param = getIntent().getStringExtra("TEXT");
```

APPELER UNE ACTIVITÉ EXPLICITEMENT

- Lancement d'une activité d'une autre application

Soit une seconde application dans le package **fr.ensicaen.appli2**.

Une activité peut la lancer ainsi :

```
Intent intent = new Intent(Intent.ACTION_MAIN);  
intent.addCategory(Intent.CATEGORY_LAUNCHER);  
intent.setClassName("fr.ensicaen.appli2",  
    "fr.ensicaen.appli2.MainActivity");  
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);  
startActivity(intent);
```

APPELER UNE ACTIVITÉ IMPLICITEMENT

- Il existe bon nombre d'activités possibles que l'on peut appeler par défaut sous Android via une description

Exemple: le téléphone

```
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:02-31-53-XX-XX"));
startActivity(intent);
```

La liste des contacts :

```
Intent intent = new Intent(Intent.ACTION_PICK,
                           ContactsContract.Contacts.CONTENT_URI);
```

...

APPELER UNE ACTIVITÉ POUR EN OBTENIR UN RÉSULTAT

- Appeler l'activité dans le but d'obtenir un résultat

`startActivityForResult(intent, intValue);`

- Récupérer le résultat dans la méthode :

```
public void onActivityResult(int requestCode, int resultCode, Intent intent)
{ ... }
```

- En dehors du *requestCode*, c'est dans l'*Intent* que l'on transfère les paramètres avec la méthode *putExtra*.

RETOURNER LE RÉSULTAT À L'ACTIVITÉ PRÉCÉDENTE

- Le retour de paramètre à destination de l'activité appelante se fait par la méthode : ***setResult***.
- Par exemple, sur l'appui d'un bouton :

```
Intent result = new Intent();  
result.putExtra("nom" , editTextNom.getText().toString());  
result.putExtra("prenom" , editTextPrenom.getText().toString());  
setResult(RESULT_OK, result);  
finish();
```

- Prévoir aussi l'annulation :

```
setResult(RESULT_CANCELED, null);  
finish();
```

Les services sont des programmes lancés en arrière plan (il n'y aura donc aucune interface avec l'utilisateur).

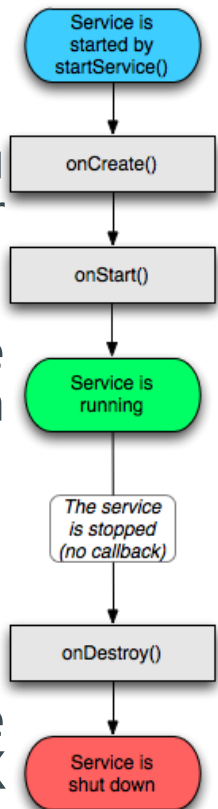
Il existe deux types de services:

- ❑ les services locaux: `LocalService`
- ❑ les services distants

- Un **service** n'a pas de durée définie , il est là pour exécuter sa tâche et il fonctionnera tant que c'est nécessaire.

On va présenter les différentes méthodes qui correspondent au **cycle de vie d'un service**. Ces méthodes seront à surcharger quand vous créerez votre service :

- **OnCreate()** : Cette méthode est appelée à la création du service et est en général utilisée pour initialiser ce qui sera nécessaire à votre service.
- **OnStart(Intent i)** : Le service démarre. Valable uniquement pour les versions du **SDK inférieur à 2.0**.
- **OnStartCommand(Intent i, int flags, int startId)** : Le service démarre. Valable uniquement pour les versions du **SDK supérieur à 2.0**.
- **OnDestroy()** : Appelé à la fermeture du **service**.

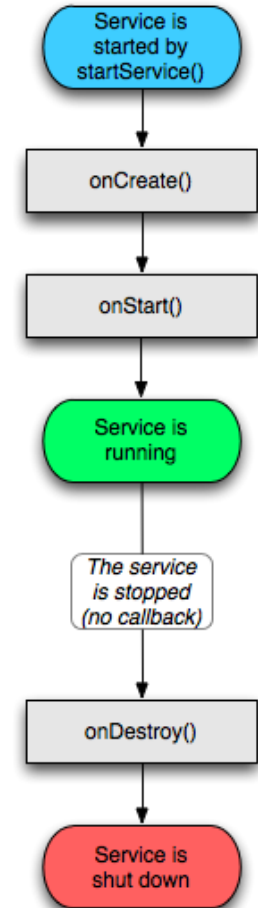


Programmation

```
startService(new Intent(nomActivy.this, NomService.class));
```

Nom de l'activité appelante

Nom du service



Programmation

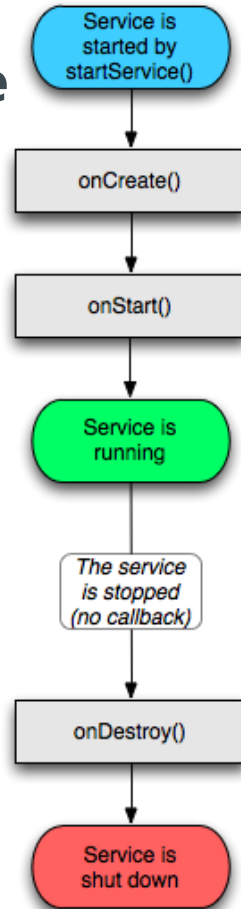
Une classe service doit impérativement hériter la classe **Service**

Exemple

```
public class MonPremierService extends Service
{
    ....
}
```

il faut déclarer le **service** dans le fichier **AndroidManifest.xml**

```
<service android:name ="com.tuto.android.MonPremierService"/>
```



LE FICHER MANIFEST

Il contient :

- Le package
- La version de l'application
- La version minimum du SDK pour pouvoir utiliser l'application
- L'icône et le nom de votre application
- La description de l'activité principale
 - Le nom de la classe qui implémente Activity.
 - Le titre pour l'activité.
 - Des filtres sur l'activité .

LE FICHER MANIFEST

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

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".FooFragment">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>

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

</manifest>
```

- La balise “**application**” sert à déclarer les différentes propriétés de votre application :
- **android:icon** : L'emplacement où se trouve l'icône de votre application.
- **android:label** : Le nom de votre application (il se trouve dans strings.xml).
- **android:theme="@style/AppTheme"** : Spécifie le thème utilisé par votre application. Ce thème pointe vers le style AppTheme déclaré dans le fichier styles.xml
- La balise “**activity**” permet de déclarer une activité, à chaque nouvelle activité il faut remettre cette balise.
- **android:name** : le nom de la classe java qui représente l'activité. Le nom doit commencer par un . et on ne met pas le .java à la fin.
- **android:label** : le label de l'activité en question
- **intent-filter** : c'est pour spécifier une action.
 - la sous balise **action** est pour spécifier l'action à exécuter, dans notre cas c'est le main.
 - la sous balise **category** est là pour spécifier la catégorie de l'action.

- **uses-feature** : la prise en charge de la fonctionnalité correspondante sur le périphérique avant d'installer une application.
- **supports-screens** : permet de définir quels types d'écran supportent votre application.
- **uses-sdk** : Pour indiquer la version minimale avec laquelle votre application est compatible
- **uses-configuration** : Indique les fonctionnalités d'entrée spécifiques requises par l'application.
- etc...

- Certaines fonctionnalités sont soumises à des droits d'accès
- A l'installation, GooglePlay ou tout autre moyen d'installation, les permissions nécessaires au fonctionnement de l'application sont présentées à l'utilisateur
- Pour ajouter une permission à votre application, modifier le fichier « AndroidManifest.xml » et ajouter des entrées :

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

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

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

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


Activité

```
<activity>  
<intent-filter>  
    ...les savoir-faire  
</intent-filter>  
</activity>
```

Ecouteur d'intention diffusée

```
<receiver>  
<intent-filter>  
    ... les savoir-faire  
</intent-filter>  
</receiver>
```

Service

```
<service>  
<intent-filter>  
    ... les savoir-faire  
</intent-filter>  
</service>
```

Fournisseur de contenu

```
<provider>  
<grant-uri-permission .../>  
<path-permission .../>  
</provider>
```

- De nombreuses applications Android sont dédiées à la présentation de données.
- Pour présenter une liste de données, le moyen le plus communément utilisé est la **ListView** ou le **ListFragment** (ListView encapsulée dans un Fragment).
- Ce mécanisme permet d'afficher une liste d'objets en offrant une souplesse de personnalisation graphique.

Pour afficher une liste de données, nous avons besoin :

- D'une liste d'objets contenant les données,
- Une ListView placée sur l'interface,
- Un adaptateur (Adapter) qui va faire le lien entre les données et l'affichage,
- L'Adapter hérite de l'interface « ListAdapter ». On utilise souvent « ArrayAdapter<T> ».

- Lier la ListView et son Adapter :

```
setListAdapter( ListAdapter )
```

- Utiliser un Adapter existant :

```
ArrayAdapter<String> adapter = new    ArrayAdapter<String>(
    this, android.R.layout.simple_list_item_1,
    android.R.id.list, values);
```

values : String[] ou List<String>

■ Xml

```
<ListView android:id="@android:id/list"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content">  
</ListView>
```

■ Java

```
private String[] mStrings = { "France", "Belgique", "Canada", "Japon" };
```

...

```
listView = (ListView) findViewById(R.id.list);  
ArrayAdapter<String> adapter = new ArrayAdapter<String>(MainActivity.this,  
    android.R.layout.simple_list_item_1, mStrings);
```

```
listView.setAdapter(adapter);
```

Problème de performance : **inflate** et **findViewById** sont lents

On utilise donc un mécanisme de recyclage qui permet de ne faire appel qu'une seule fois à ces méthodes : utilisation d'un « **ViewHolder** ».

```
View row = convertView;
if(row == null)
{
    LayoutInflater inflater =
        ((Activity)getContext()).getLayoutInflater();
    row = inflater.inflate(layoutResourced, parent, false);
    holder = new ViewHolder();
    holder.viewNom = (TextView)row.findViewById(R.id.clientNom);
    holder.viewPrenom = (TextView)row.findViewById(R.id.clientPrenom);

    row.setTag(holder);
} else {
    // RECYCLED VIEW
    holder = (ViewHolder)row.getTag();
}
```

RECYCLERVIEW EN LOLLIPOP ET +

Avec Android 5, Google introduit une nouvelle classe :

RecyclerView qui prend en charge le recyclage de vues.

Avec cette classe, on sait précisément à quel endroit on agit sur chaque item de la liste :

- affichage,
- événements,
- animations.

A ajouter dans le fichier build.gradle

compile 'com.android.support:recyclerview-v7:23.1.0'

- Ecrire dans un fichier
- Sauvegarder dans une base de données
- **SharedPreferences**

Création

Les SharedPreferences sont à récupérer depuis un context, avec

context.getSharedPreferences(NAME,MODE)

`context.getSharedPreferences("data_prefs",Context.MODE_PRIVATE);`

Sauvegarde

- Ouvrir un éditeur, avec **sharedPreferences.edit()**,
- Accéder aux méthodes **.putString(KEY,VALUE)**, **.putInt(KEY,VALUE)**, etc.
- appeler la méthode **.apply()**

```
sharedPreferences.edit()
```

```
    .putInt("cle_integer", 3)
```

```
    .putString("cle_string", "monString")
```

```
    .apply();
```

Récupération

La récupération d'un élément persisté se fait avec les méthodes :

- **.getString(KEY,DEFAULT_VALUE),**
- **.getInt(KEY,DEFAULT_VALUE),** etc.

On peut vérifier la présence d'un élément avec **.containsKey(KEY)**

```
int prefsInteger = sharedPreferences.getInt("key_integer", 0);
```

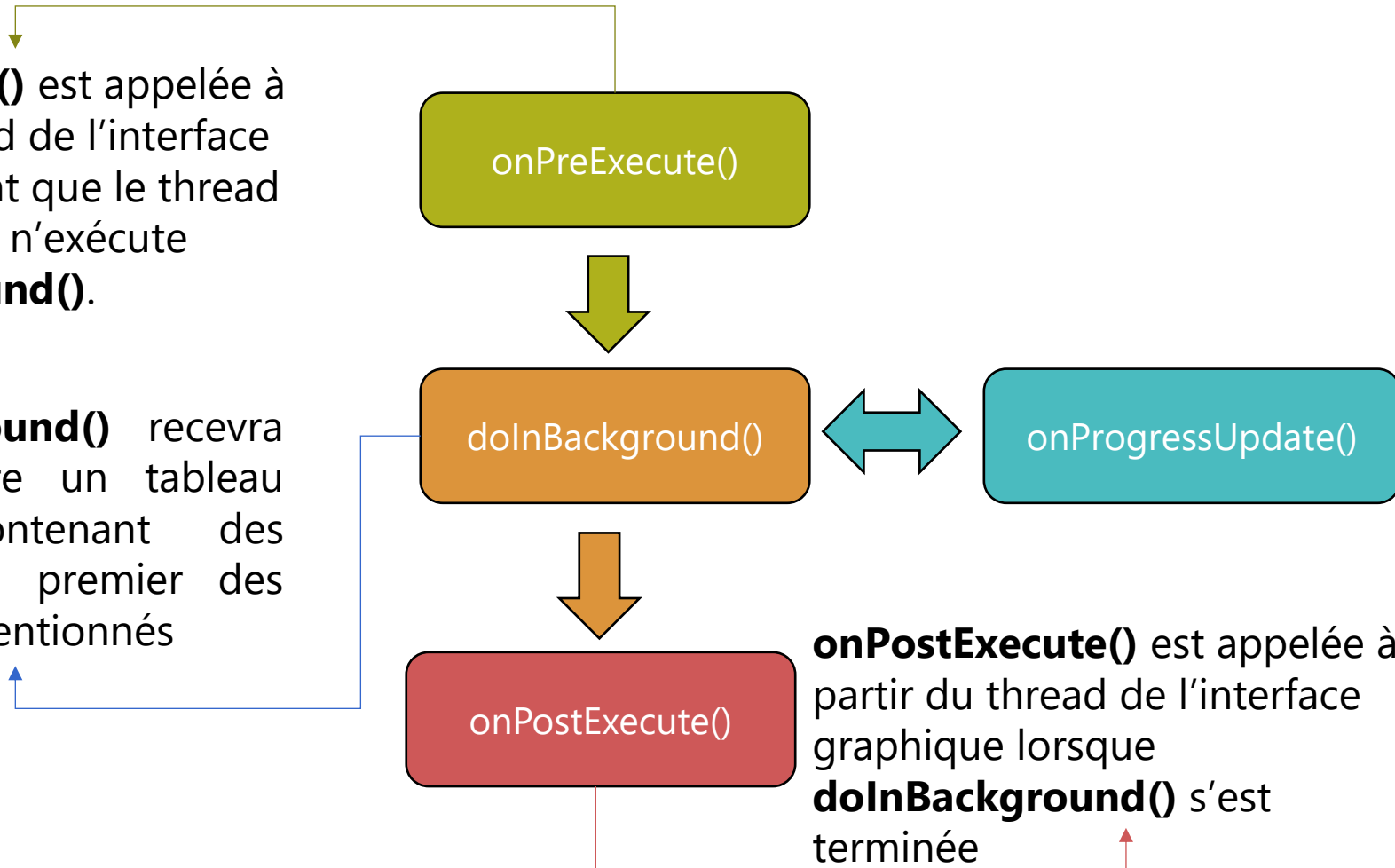
```
String prefsString = sharedPreferences.getString("key_string", null)
```

- une **AsyncTask** permet de réaliser des tâches de manière asynchrone, à la manière de la classe **Thread**
- 3 paramètres
 - Le type de l'information nécessaire pour le traitement de la tâche (les URL à télécharger, par exemple).
 - Le type de l'information passée à la tâche pour indiquer sa progression.
 - Le type de l'information passée au code après la tâche lorsque celle-ci s'est terminée.

Declaration = `class AddStringTask extends AsyncTask<Void, String, Void> {`

onPreExecute() est appelée à partir du thread de l'interface utilisateur avant que le thread en arrière-plan n'exécute **doInBackground()**.

doInBackground() recevra en paramètre un tableau variable contenant des éléments du premier des trois types mentionnés



Exemple avec barre de progression

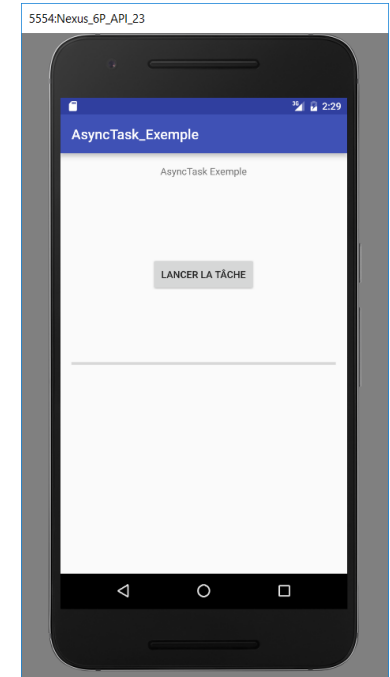
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin">

    <TextView
        android:text="@string/asynctask"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView2" android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"/>

    <Button
        android:layout_width="150dp"    android:layout_height="50dp"
        android:text="Lancer la tâche"  android:id="@+id/button"
        android:layout_above="@+id/progressBar" android:layout_centerHorizontal="true"
        android:layout_marginBottom="98dp"/>

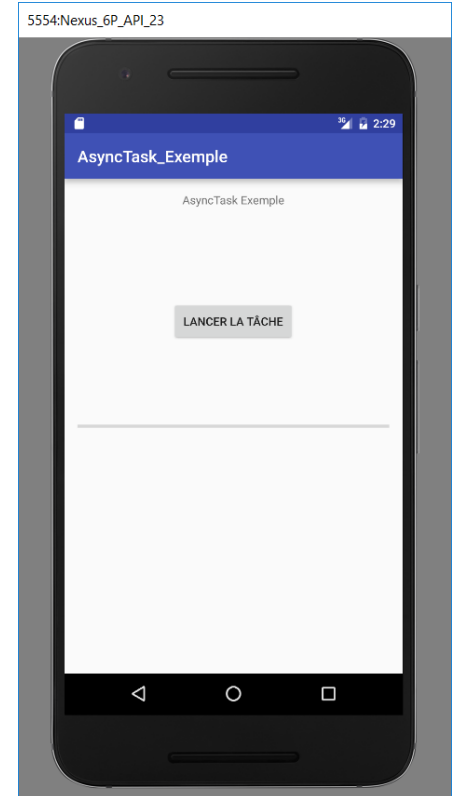
    <ProgressBar
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="fill_parent"
        android:layout_height="10dp"    android:id="@+id/progressBar"
        android:layout_centerVertical="true"    android:layout_alignParentStart="true"/>

</RelativeLayout>
```



Fichier java

```
public class AsyncTaskActivity extends AppCompatActivity {  
  
    private ProgressBar mProgressBar;  
    private Button tacheButton;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_async_task);  
        // On récupère les composants de notre layout  
        mProgressBar = (ProgressBar) findViewById(R.id.progressBar);  
        tacheButton = (Button) findViewById(R.id.button);  
  
        // On met un Listener sur le bouton  
        tacheButton.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View arg0) {  
                ProgressBarLanceur calcul = new ProgressBarLanceur();  
                calcul.execute();  
            }  
        });  
    }  
}  
//Déclaration de la classe
```



Fichier java

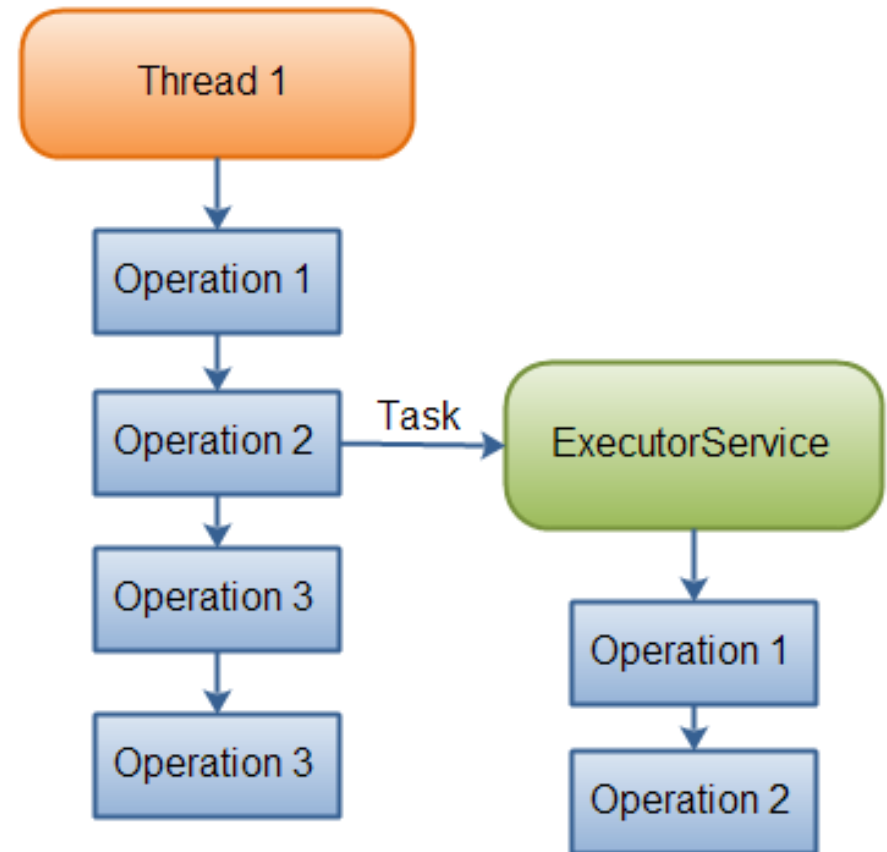
```
private class ProgressBarLanceur extends AsyncTask<Void, Integer, Void>
{
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        Toast.makeText(getApplicationContext(), "Début traitement asynchrone", Toast.LENGTH_LONG).show();
    }
    @Override
    protected void onProgressUpdate(Integer... values){
        super.onProgressUpdate(values);
        // Mise à jour de la ProgressBar
        mProgressBar.setProgress(values[0]);
    }
    @Override
    protected Void doInBackground(Void... arg0) {

        int progress;
        for (progress=0;progress<=200;progress++){
            for (int i=0; i<2000000; i++){
                //la méthode publishProgress met à jour l'interface en invoquant la méthode onProgressUpdate
                publishProgress(progress);
                progress++;
            }
            return null;
        }
        @Override
        protected void onPostExecute(Void result) {
            Toast.makeText(getApplicationContext(), "Fin traitement asynchrone", Toast.LENGTH_LONG).show();
            mProgressBar.setProgress(0);
        }
    }
}
```

A partir de Android 30, fini les AsyncTask

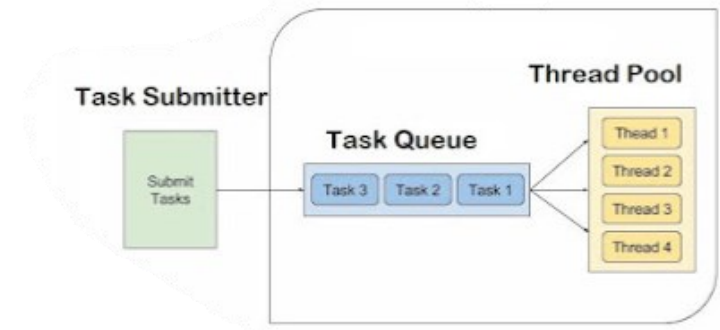
ExecutorService

- en Java est une interface qui fait parti du package **java.util.concurrent**.
- permet d'exécuter simultanément des tâches asynchrones.



ExecutorService

- Fil unique: Crée une seule instance de thread à l'aide d'un **ExecutorService**
- **Pool de threads**: Crée un **pool** de threads en spécifiant le nombre de threads dans le paramètre
- A pool de threads planifié



1. `ExecutorService executor = Executors.newSingleThreadExecutor();`
2. `ExecutorService executor = Executors.newCachedThreadPool();`
3. `ExecutorService executor = Executors.newFixedThreadPool(int count);`
4. `ExecutorService executor = Executors.newScheduledThreadPool(int count);`

TACHES SYNCHRONISÉES

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import android.os.Handler;
import android.os.Looper;
```

```
ExecutorService executor = Executors.newSingleThreadExecutor();
```

```
Handler handlerUI = new Handler(Looper.getMainLooper());
```

```
executor.execute(new Runnable() {                                // Traitement à déporter dans un thread
```

```
    @Override
```

```
    public void run() {
```

```
        System.out.println("Exemple de méthode d'exécution");
```

```
        handlerUI.post(new Runnable() {                            // Instructions à exécuter par le thread UI
```

```
            @Override
```

```
            public void run () {
```

```
                .....
```

```
            }
```

```
        });
```

```
    }
```

```
});
```

```
executor.shutdown();
```

```
System.out.println("Est-ce que ExecutorService est arrêté ? : " + executor.isShutdown());
```

```
}
```

Quelques méthodes

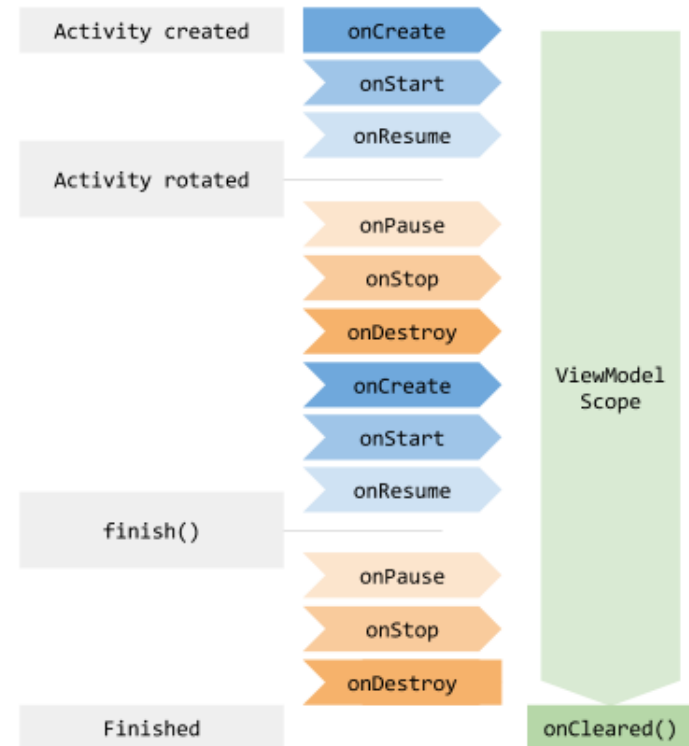
- **submit()**
- **invokeAny()**
- **invokeAll()**
- **shutdownNow()**

VIEWMODEL (PATTERN M-V-VM)

- Le modèle M-V-VM est un patron de conception vu comme une amélioration du pattern MVC.
- Android l'implémente afin d'apporter une meilleure conservation des données et mise à jour de l'affichage
- Quelques règles définissent ce pattern :
 - Règle 1 : La vue ne connaît pas son contrôleur, elle sait juste afficher
 - Règle 2 : Le contrôleur ne connaît pas le modèle (différence avec le MVC)
 - Règle 3 : le modèle ne connaît pas le ViewModel
 - Règle 4 : le ViewModel possède le modèle (en MVC, c'est souvent le contrôleur)
 - Règle 5 : le contrôleur possède la vue
 - Règle 6 : le contrôleur possède le ViewModel

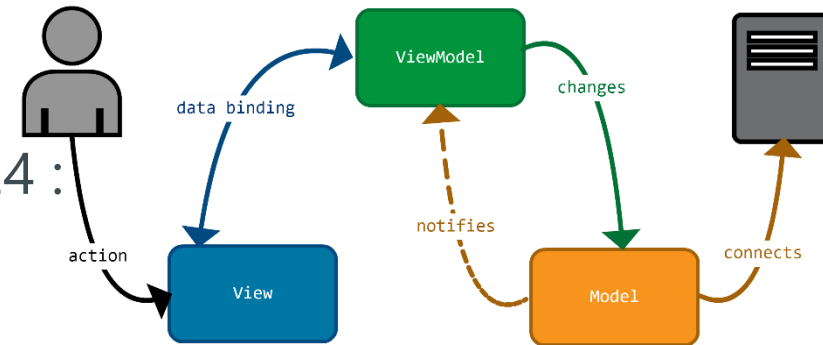
VIEWMODEL – CYCLE DE VIE

- Le contrôleur dans le cas d'Android est un peu hybride, réparti entre l'Activité et l'application
 - ViewModel : stocké dans un contexte d'activité évitant ainsi de le perdre quand un Fragment ou une activité sont détruits puis reconstruits
 - Le contrôleur peut-être l'activité ou un fragment



CONSTRUIRE RAPIDEMENT UNE APPLICATION INCLUANT UN MVVM

- A la création d'un projet avec Android Studio, certains projets sont construits sur la base du MVVM
- Par exemple avec la version Koala 2024 :
 - Bottom Navigation Views Activity
 - Navigation Drawer Views Activity
 - Responsive Views Activity
- Ce sont toutes les 3 des classiques d'applications.



Sources

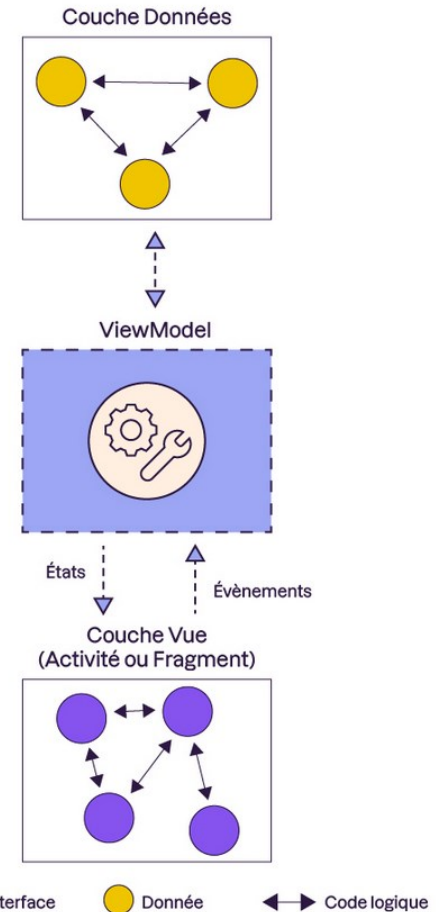
- [What Are the Benefits of Model-View-ViewModel](#)
- [ViewModel overview | Android Developers](#)



CRÉER UN VIEWMODEL

- Java → class XY extends ViewModel { ...
- Kotlin → class XY : ViewModel() { ...

```
public class RandomIntegerViewModel extends ViewModel {  
    private final MutableLiveData<Integer> randomInt  
        = new MutableLiveData(new Integer());  
    public LiveData<Integer> getRandomInt() { return randomInt; }  
  
    public RandomIntegerViewModel() { randomize(); }  
  
    public void randomize() {  
        Random = new Random();  
        randomInt.setValue(random.nextInt(99));  
    }  
}
```



MVVM - LE CONTRÔLEUR (ACTIVITÉ OU FRAGMENT)

- La première fois, le ViewModel est créé. Par la suite, la même instance est récupérée.

```
public class MyActivity extends AppCompatActivity {  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        RandomIntegerViewModel model  
            = new ViewModelProvider(this).get(RandomIntegerViewModel.class);  
        model.getRandomInt().observe(this, i -> {  
            // update UI  
        });  
    }  
}
```

Pour un Fragment : on remplace le **"this"** par **"getLifecycleOwner()"**

SQLite étant intégré au moteur d'exécution d'Android, toute application peut créer des bases de données SQLite.

Pour créer et ouvrir une base de données, la meilleure solution consiste à créer une sous-classe de [SQLiteOpenHelper](#).

- Un constructeur qui appelle celui de sa classe parente et qui prend en paramètre le Context (une Activity), le nom de la base de données, une éventuelle fabrique de curseur (le plus souvent, ce paramètre vaudra null) et un entier représentant la version du schéma de la base.
- onCreate(), à laquelle vous passerez l'objet SQLiteDatabase que vous devrez remplir avec les tables et les données initiales que vous souhaitez.
- onUpgrade(), à laquelle vous passerez un objet SQLiteDatabase ainsi que l'ancien et le nouveau numéro de version. Pour convertir une base d'un ancien schéma à un nouveau, l'approche la plus simple consiste à supprimer les anciennes tables et à en créer de nouvelles.

```
public class BDD extends SQLiteOpenHelper {

    private static final String TABLE_USER = "table_user";
    private static final String idUser = "idUser";
    private static final String nom = "nom";
    private static final String prenom = "prenom";

    private SQLiteDatabase bdd;

    public String PATH = "";

    private static final String CREATE_USER = "CREATE TABLE " + TABLE_USER + " ("
        + " idUser" + " INTEGER PRIMARY KEY AUTOINCREMENT," + " nom CHAR(50)," + " prenom CHAR(50));";
    public BDD(Context context, String name, CursorFactory factory, int version) {
        super(context, name, factory, version);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

        db.execSQL("DROP TABLE IF EXISTS " + TABLE_USER + ";");
        System.out.println("onupgrade");
        //onCreate(db);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        //on crée la table à partir de la requête écrite dans la variable CREATE_BDD
        System.out.println("oncreate");
        db.execSQL(CREATE_USER);
        System.out.println("FINoncreate");
    }
}
```

```
public void open(){  
    System.out.println("opendatabase");  
    bdd=this.getWritableDatabase();  
}  
public void close(){  
    bdd.close();  
}
```

```
// supprimer tous les utilisateurs  
public int SupprimeAllUtilisateur() {  
    String delete ="DELETE FROM "+TABLE_USER ;  
    bdd.execSQL(delete);  
    return 0;  
}
```

```
// recherche par nom et prenom  
public int contientUtilisateur(String nom2,String prenom2) {  
    Cursor c = bdd.query(TABLE_USER, new String[] {"idUser", "nom", "prenom"}, "nom" + " LIKE \"" + nom2 + "\"" + " AND  
    prenom LIKE \"" + prenom2 + "\"", null, null, null, null);  
    if (c.getCount()!=0){  
        c.moveToFirst();  
        System.out.println(c.getInt(0)+c.getString(1)+c.getString(2));  
    }  
    return c.getCount();  
}
```

Déclaration

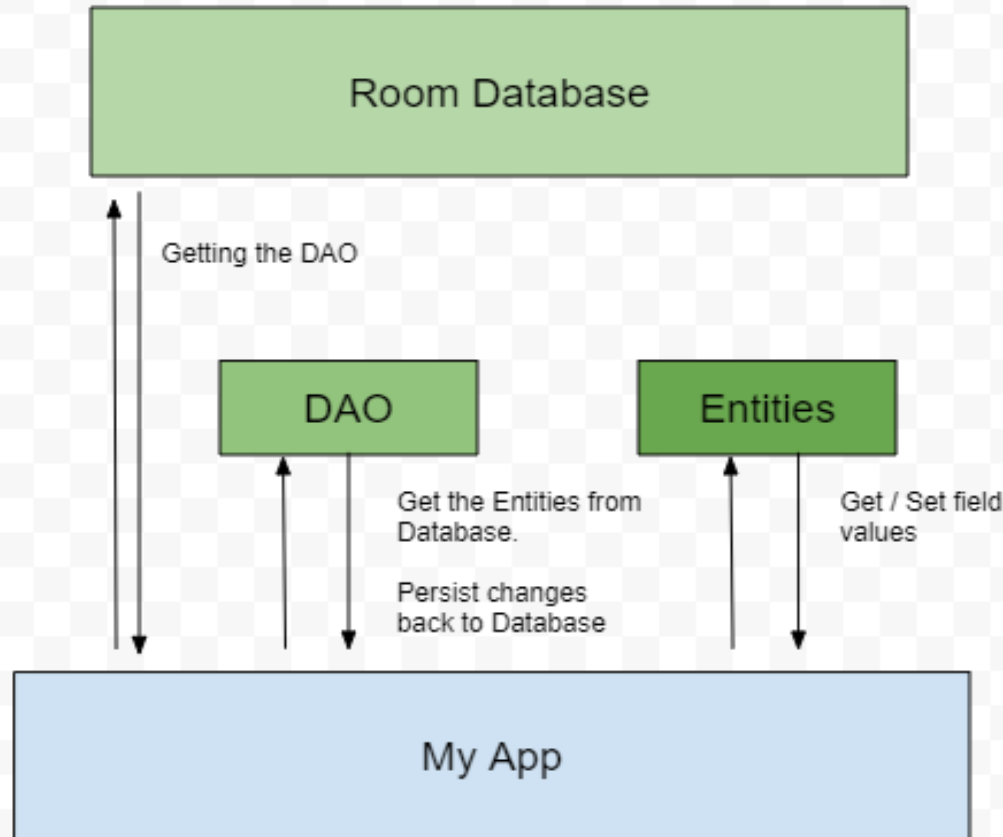
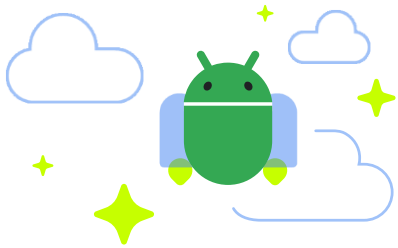
```
public Context c;  
public BDD bdd;
```

```
bdd= new BDD(this , "test17.db",null, 1);  
bdd.open();
```

```
EditText nom = (EditText)findViewById(R.id.nom);  
EditText prenom=(EditText)findViewById(R.id.prenom);  
String nom2 = nom.getText().toString();  
String prenom2 = prenom.getText().toString();  
int reponse = bdd.contientUtilisateur(nom2, prenom2);  
bdd.close();
```

il est nécessaire de disposer de la permission

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



// dépendances Room

```
dependencies {
```

```
.....
```

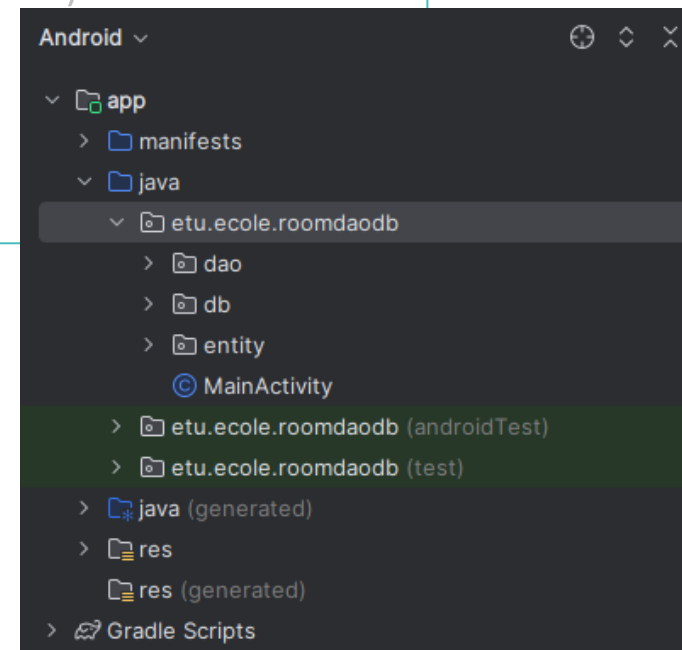
```
val room_version = "2.6.1"
```

```
implementation("androidx.room:room-runtime:$room_version")
```

```
annotationProcessor("androidx.room:room-compiler:$room_version")
```

```
implementation("androidx.annotation:annotation:1.8.2")
```

```
}
```



// création entité User

```
@Entity
public class User {
    @PrimaryKey
    public int uid;

    @ColumnInfo(name = "first_name")
    public String firstName;

    @ColumnInfo(name = "last_name")
    public String lastName;

    public User(int uid, String firstName, String lastName) {
        this.uid = uid;
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```

// création DAO

```
import androidx.room.Dao;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.Query;
@Dao
public interface UserDao {
    @Query("SELECT * FROM user")
    List<User> getAll();

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    List<User> loadAllByIds(int[] userIds);

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " + "last_name LIKE :last LIMIT 1")
    User findByName(String first, String last);

    @Insert
    void insertAll(User... users);

    @Insert
    void insert(User user);

    @Delete
    void delete(User user);
}
```


// création d'une instance de base de données

```
import androidx.room.Database;
import androidx.room.RoomDatabase;

import etu.ecole.roomdaodb.dao.UserDao;
import etu.ecole.roomdaodb.entity.User;

@Database(entities = {User.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract UserDao userDao();
}
```

// codes dans l'activity

.....

```
AppDatabase db = Room.databaseBuilder(getApplicationContext(),AppDatabase.class, "database-  
name").build();
```

```
UserDao userDao = db.userDao();
```

```
User useur= new User(1,« ensi", "ndiag");
```

```
userDao.insertAll(useur,useur);
```

```
List<User> users = userDao.getAll();
```

```
for(User list:users)  
{  
    Log.d("user",list.firstName);  
}  
...
```

L'état de la téléphonie est géré par la classe **TelephonyManager**.

Il permet de récupérer:

- le nom de l'opérateur,
- le nom du téléphone,
- l'IMEI,
- l'état du téléphone ...

Pour lire ces informations, il est nécessaire de disposer de la permission

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

```
TelephonyManager  
tm=(TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);  
  
String imei=tm.getDeviceId().toString();  
String numseri=tm.getSimSerialNumber().toString();
```

- Permettre à l'utilisateur d'appeler un n°composé

```
Uri numero = Uri.parse("tel:0559574320");
```

```
Intent composer = new Intent(Intent.ACTION_DIAL, numero);
```

```
startActivity(composer);
```

Il faut activer la permission :

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

- Appeler directement un n°

```
Uri numero = Uri.parse("tel:0559574320");
```

```
Intent appeler = new Intent(Intent.ACTION_CALL, numero);
```

```
startActivity(appeler);
```

Il faut activer la permission :

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

SmsManager est la classe qui va gérer les SMS.

Il n'est pas possible d'avoir une instance de cette classe, mais il faudra utiliser une méthode statique **SmsManager.getDefault()** pour en récupérer une par défaut.

- Envoi d'un message par : `sendTextMessage` en précisant
- le n° et le texte

Il faut activer la permission :

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

```
SmsManager sms = SmsManager.getDefault();  
  
sms.sendTextMessage("5556", null, "Essai", null, null);
```

Pour pouvoir recevoir des **intents**, il faudra créer une classe qui implémente **BroadcastReceiver** (conçus pour recevoir des intents et appliquer des comportements spécifiques à votre code).

- Déclaration d'un écouteur d'intentions diffusées

```
<receiver android:name=".receiver.monRecepteur" android:enabled="true">
```

```
<intent-filter>
```

```
<action android:name="android.provider.Telephony.SMS_RECEIVED" />
```

```
</intent-filter>
```

```
</receiver>
```

- Ecriture de l'écouteur d'intention diffusées par héritage de BroadcastReceiver et surcharge de la méthode onReceive
- L'Intent reçu en paramètre de onReceive contient les messages reçus sous forme brute désignés par la clé "pdu"
- Ces message bruts peuvent être convertis en objets de classe SmsMessage par la méthode createFromPdu de cette classe.
- Un SmsMessage permet de récupérer l'expéditeur, le corps du message ainsi que l'expéditeur et le corps d'un mail

Il faut activer la permission : `<uses-permission android:name ="android.permission.RECEIVE_SMS" />`

Le réseau peut être disponible ou indisponible, suivant que le téléphone utilise une connexion Wifi, 3G, bluetooth,,,

la classe **NetworkInfo** (depuis **ConnectivityManager**) permet de lire l'état de la connexion réseau parmi les constantes de la classe

Il faut activer la permission :

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

```
ConnectivityManager connMgr =  
(ConnectivityManager)getSystemService(Context.CONNECTIVITY_SERVICE);  
NetworkInfo networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);  
boolean isWifiConn = networkInfo.isConnected();
```

On utilise la méthode **setNetworkPreference** sur l'objet **ConnectivityManager** pour lui donner l'entier correspondant au type de connexion voulu.

Exemple

```
manager.setNetworkPreference(ConnectivityManager.TYPE_WIFI);
```

```
WifiManager wifi = (WifiManager) getSystemService(Context.WIFI_SERVICE);  
if (!wifi.isWifiEnabled())  
    wifi.setWifiEnabled(true );
```

Il faut activer la permission :

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


Le bluetooth est géré par les 3 classes suivantes:

- **BluetoothAdapter**: similaire au **WifiManager**, cette classe permet de gérer les autres appareils bluetooth et d'initier les communications avec ceux-ci.
- **BluetoothDevice**: objet représentant l'appareil distant.
- **BluetoothSocket** et **BluetoothServerSocket**: gère une connexion établie

il faut activer les permissions

- ✓ **android.permission.BLUETOOTH**
- ✓ **android.permission.BLUETOOTH_ADMIN**

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();  
if (mBluetoothAdapter == null) {  
    // Device does not support Bluetooth  
}  
if (!mBluetoothAdapter.isEnabled()) {  
    mBluetoothAdapter.enable();  
}
```

Liste des appareils connectés

```
private Set<BluetoothDevice> devices;  
.....  
devices = mBluetoothAdapter.getBondedDevices();  
for (BluetoothDevice blueDevice : devices) {  
    Toast.makeText(getApplicationContext(), "Device = "  
+ blueDevice.getName(), Toast.LENGTH_SHORT).show();  
}
```

```
lm = (LocationManager) this.getSystemService(LOCATION_SERVICE);
```

Associer un écouteur d'événements par :requestLocationUpdate en précisant :

- Le mode (GPS ou réseau)
- Le rythme
- La distance minimale

Récupérer les informations dans l'écouteur

- Latitude, longitude, altitude
- Précision

Possibilité de calculer une distance

Il faut activer les permissions :

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

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

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

- La classe **Camera** permet la prise de photo par **takePicture** en associant un écouteur d'événement pour récupérer la photo en raw ou JPEG. La méthode **onPictureTaken** de cet écouteur est appelée quand la photo est faite, l'image est reçue en tableau d'octets.
- Nécessite une prévisualisation par un objet de classe **SurfaceView** dans l'interface auquel on associe un écouteur d'événements pour :
 - ❑ Démarrer/arrêter l'appareil photo (méthodes **open** et **release** de la classe **camera**)
 - ❑ Lancer/arrêter la prévisualisation (méthodes **startPreview** et **stopPreview** de la classe **Camera**)
- On peut enregistrer le tableau d'octets reçu par **onPictureTaken** dans un fichier ou utiliser **BitmapFactory** pour le convertir en image

Il faut activer la permission :

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

Pour accéder au vibreur on utilise la classe **Vibrator**

dont une instance est obtenue par :

```
Vibrator vibreur= (Vibrator)getSystemService(Context.Vibrator_SERVICE);
```

Puis on peut faire vibrer le téléphone pour une durée donnée par la méthode : `vibreur.vibrate(int)` où le paramètre est la durée de vibration en ms.

Il faut activer la permission :

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

Les types de capteurs disponibles sont les suivants (selon le modèle certains capteurs peuvent ne pas être disponibles) :

- ☐ Sensor.TYPE_ACCELEROMETER
- ☐ Sensor.TYPE_GRAVITY
- ☐ Sensor.TYPE_GYROSCOPE
- ☐ Sensor.TYPE_LIGHT
- ☐ Sensor.TYPE_MAGNETIC_FIELD
- ☐ Sensor.TYPE_ORIENTATION
- ☐ Sensor.TYPE_PRESSURE
- ☐ Sensor.TYPE_PROXIMITY
- ☐ Sensor.TYPE_TEMPERATURE

Pour accéder à un capteur on utilise la classe `SensorManager` dont une instance est obtenue par :

- `SensorManager gestionnaireCapteurs =
(SensorManager) getSystemService(Context.SENSOR_SERVICE);`

Puis on récupère un capteur particulier par :

- `SensormonCapteur =
gestionnaireCapteurs.getDefaultSensor(type_de_capteur);`

Le paramètre `type_de_capteur` est l'un des types indiqués ci-dessus.

Créer un MediaPlayer : `MediaPlayer lecteur = MediaPlayer.create(Context, int)`

- Le premier paramètre est l'activité elle-même
- Le second paramètre est l'identificateur du fichier son

Utiliser le MediaPlayer :

- `lecteur.start()` pour jouer le son
- `lecteur.pause()` pour suspendre le son, il sera repris par `start()`
- `lecteur.stop()` pour arrêter le son

- Mettre un VideoView dans l'interface

```
<VideoView android:id="@+id/ecran_video"  
    android:layout_width="fill-parent"  
    android:layout_height="fill-parent" />
```

- Définir le chemin de la vidéo (placée dans res/raw)

```
Uri chemin = Uri.parse("android.resource://paquetage_de_l_application/"  
    + R.raw.nom_du_fichier_video);
```

- Associer un lecteur vidéo à la vue:

```
VideoView lecteur = (VideoView) findViewById (R.id.ecran_video);
```

```
lecteur.setVideoURI(chemin);
```

```
lecteur.setMediaController(new MediaController(activité)); //
```

facultatif

```
lecteur.requestFocus();
```

- Si on a mis setMediaController, lors d'un clic long sur la vue une fenêtre de contrôle apparaît avec :
 - ☐ Un bouton Play/pause
 - ☐ Un bouton Avance rapide
 - ☐ Un bouton Recul rapide
 - ☐ Un curseur indiquant la position courante et permettant de se déplacer

Le développement sous Android est différent d'un développement Java classique

Il existe des bonnes pratiques liées à cette plateforme :

- codage
- ergonomie
- compatibilité entre matériels
- design

codage

- Ne pas perdre de vue que l'application est exécutée sur une plateforme dont les ressources sont limitées
- Minimiser la création d'objets
- Ne pas utiliser les getteurs / setteurs au sein de la classe
 - préférer la lecture ou l'affectation directe des propriétés
- Déclarer **static** les méthodes qui peuvent l'être
- Préférer les constantes plutôt que les énumérations

ergonomie

- Souvent sans clavier et dispositif de pointage
- le doigt est utilisé sur l'écran
- Ne pas dessiner des widgets trop petits
- taille recommandée aux alentours de 9mm, soit 48dp
 - ❑ dp : density-independant pixels
 - ❑ <http://developer.android.com/design/style/metrics-grids.html>

design

- Les icônes de lancement d'applications font une taille de 48x48 dp

- les icônes de Play Store font 512 x 512 px

- Utiliser les icônes prédéfinies pour la barre d'action

- Exemple :



https://dl-ssl.google.com/android/design/Android_Design_Icons_20120229.zip

- Depuis Java 8
- aident à éliminer le code de référence pouvant rendre la syntaxe trop détaillée et moins claire.

■ Syntaxe

Une expression lambda comprend les éléments suivants:

- Une liste séparée par des virgules de paramètres formels entre parenthèses.
- Le jeton flèche ->
- Un corps qui contient une seule expression ou un bloc d'instructions.

■ Exemple avec une instruction

```
myButton . setOnClickListener ( new View . OnClickListener () {  
    @Override  
    public void onClick ( View v )  
    {  
        Log . d ( "debug" , "Test " );  
    }  
});
```

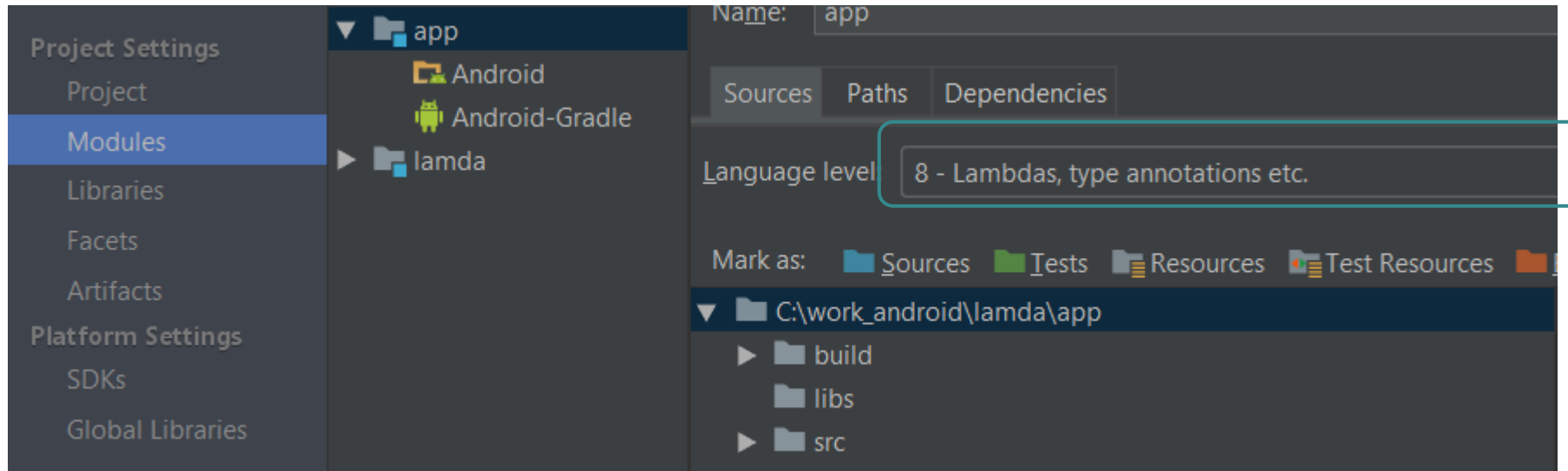


```
myButton . setOnClickListener ( v -> Log . d ( "debug" , "Test" ));
```

■ Exemple avec plusieurs instructions

```
myButton . setOnClickListener ( v -> {  
    Log . d ( "debug" , "Test " );  
    Toast . makeText ( MainActivity . this , "Mon test" , Toast . LENGTH_LONG ). show ();  
});
```

■ Configuration



PROTECTION CONTRE LE CLONAGE

- Rien d'absolu mais des précautions basiques
- Souvent un changement de nom de package au moment du clonage
 - `if (!ctx.getPackageName().equals("your.package.name")){`
- Bien sûr, une signature différente
 - `...getPackageInfo(ctx.getPackageName(),
PackageManager.GET_SIGNING_CERTIFICATES).signingInfo.getApkContentsSigners()`
 - `...getPackageInfo(ctx.getPackageName(),
PackageManager.GET_SIGNATURES).signatures`
- Et de l'obfuscation sur ces portions de code

- Développement d'applications professionnelles avec Android 2 (Reto Meier - Pearson)
- Android 4 : Développement d'applications avancées (Reto Meier – Pearson, ISBN 2744025445)
- <http://developer.android.com>
- Autres sources, forums développeurs
 - <https://developers.google.com/>
 - <http://www.vogella.com/tutorials/>
 - <http://stackoverflow.com/>

