Guide de démarrage Android

Sommaire

Thierry Vaira - tvaira@free.fr - version v1.0 - 05/01/2021

1. Présentation

1.1. Android

Android est un système d'exploitation mobile basé sur le noyau Linux et développé actuellement par Google.

D'abord été conçu pour les smartphones et tablettes tactiles, Android s'est ensuite diversifié dans les objets connectés et ordinateurs comme les télévisions (Android TV), les voitures (Android Auto), les Chromebook (Chrome OS qui utilise les applications Android) et les smartwatch (Wear OS).

Lien : Les différentes versions d'Android

Android est défini comme étant une pile de logiciels organisée en cinq couches distinctes :

- un système d'exploitation (comprenant un noyau Linux avec les pilotes) ;
- des bibliothèques logicielles telles que WebKit, OpenGL, SQLite, … ;
- un environnement d'exécution et des bibliothèques permettant d'exécuter des programmes prévus pour la plate-forme **Java** ;
- un kit de développement d'applications **(SDK)** ;
- des d'applications standards (un environnement de bureau, carnet d'adresses, application téléphone, …).

Les différents kits de développement sont :

- l'Android **NDK** (*Android Native Development Kit*) est une API du système d'exploitation Android permettant de développer directement dans le langage C/C++ du matériel cible, par opposition au Android SDK qui est une abstraction en *bytecode* **Java**, indépendante du matériel.
- l'Android **SDK** (*Software Development Kit*) est un ensemble complet d'outils de développement (pour Linux, MAC OS ou Windows). Il inclut un débogueur, des bibliothèques logicielles, un émulateur basé sur QEMU, de la documentation, des exemples de code et des tutoriaux.

L'**ADB** (*Android Debug Bridge*) est un outil inclus dans le package Android SDK. Il se compose d'un programme client et d'un programme serveur communicant entre eux et qui permet :

- la copie de fichier ;
- l'accès à la console Android ;
- la sauvegarde de la mémoire ROM ;
- l'installation de logiciel.

Pour développer des applications Android, il faut au moins disposer du Android SDK et du kit de développement Java (JDK). Pour faciliter le développement, il est conseillé d'utiliser un EDI/IDE (*Integrated Development Environment*) comme *Android Studio*.

Ces outils sont disponibles aussi bien sous Linux que sous Windows et Mac OS. Les applications Android étant exécutées par une machine virtuelle, il n'y a pas d'avantages particuliers à développer sur un système plutôt qu'un autre …

1.2. Java

La **plate-forme Java** (*the Java Platform*) est une plate-forme logicielle permettant de développer et d'exécuter des programmes écrits en langage Java indépendants de tout processeur et de tout système d'exploitation.

Toute plate-forme Java se compose principalement d'un moteur d'exécution **JVM** (**machine virtuelle Java**) et d'un compilateur fourni avec un ensemble de bibliothèques standards.

La machine virtuelle Java (*Java Virtual Machine* ou JVM_) est un appareil informatique fictif qui exécute des programmes compilés sous forme de ***bytecode*** Java. Cet appareil fictif est émulé par un

logiciel spécifique à chaque plate-forme (machine/système d'exploitation) et permet aux applications Java compilées en *bytecode* de produire les mêmes résultats quelle que soit la plate-forme, tant que celle-ci est pourvue de la machine virtuelle Java adéquate.

En résumé, la **plate-forme Java** comprend :

- un compilateur Java (`javac`), fourni par le JDK (*Java Development Kit*), convertit les codes source Java en *bytecode* Java (un langage intermédiaire).
- une JVM (`java`), fournie par le JRE (*Java Runtime Environment*), comprend un compilateur JIT (*Just In Time*) qui convertit à la volée le *bytecode* intermédiaire en un code natif pour la machine.

Java et Android :

Java est le langage de programmation "officielle" pour développer des applications Android.

Jusqu'à sa version 4.4, Android comporte une machine virtuelle nommée **Dalvik**, qui permet d'exécuter des programmes prévus pour la plate-forme Java.

Le processus de construction d'une application Android est le suivant : le code source Java de l'application est tout d'abord compilé avec un compilateur standard pour produire un *bytecode* standard pour JVM puis celui-ci est traduit en *bytecode* pour Dalvik par un programme inclus dans Android.

À partir de la version 5.0 (Lollipop) sortie en 2014, l'environnement d'exécution **ART** (*Android RunTime*) remplaça la machine virtuelle Dalvik. Cet environnement d'exécution plus performant fut développé par Google pour pallier le potentiel limité de Dalvik (créé en 2007). Avec ART, les fichiers de *package* d'application Android (une archive portant l'extension `.apk`) ne sont plus lancés directement mais décompressés et lancés avec de nouvelles bibliothèques et API.

Google annonce en 2017 que le langage **Kotlin** devient le second langage de programmation officiellement pris en charge par Android après Java.

Kotlin est un langage de programmation orienté objet qui permet de compiler pour la machine virtuelle Java. Son développement provient principalement d'une équipe de programmeurs chez JetBrains (un éditeur de logiciels qui fournit l'IDE IntelliJ).

1.3. Android Studio

Android Studio est un environnement de développement pour développer des applications Android. Il est basé sur IntelliJ.

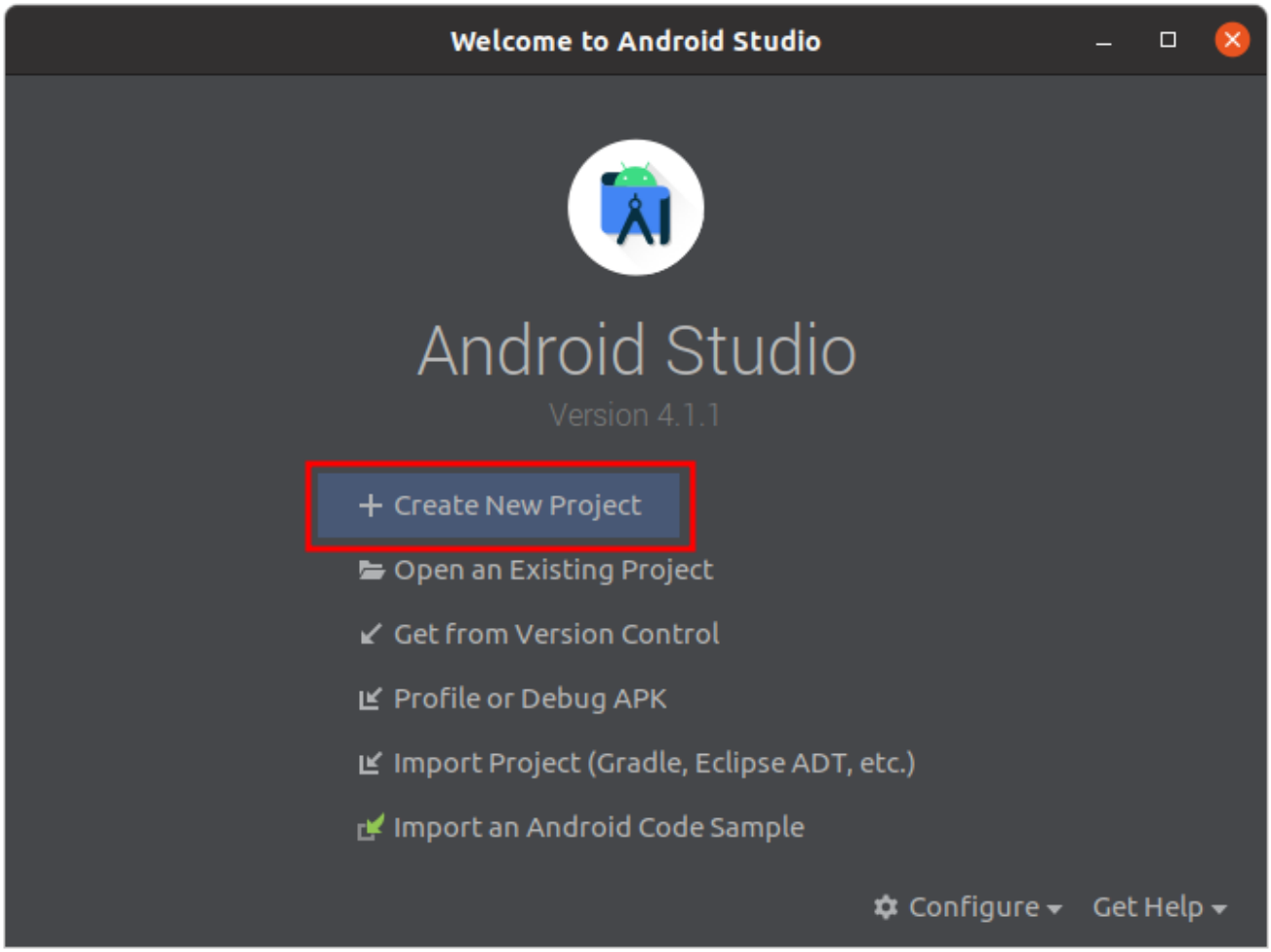
Android Studio permet principalement d'éditer les fichiers Java (ou Kotlin) et les fichiers de configuration d'une application Android.

Il propose entre autres des outils pour gérer le développement d'applications multilingues et permet de visualiser la mise en page des écrans sur des écrans de résolutions variées simultanément.

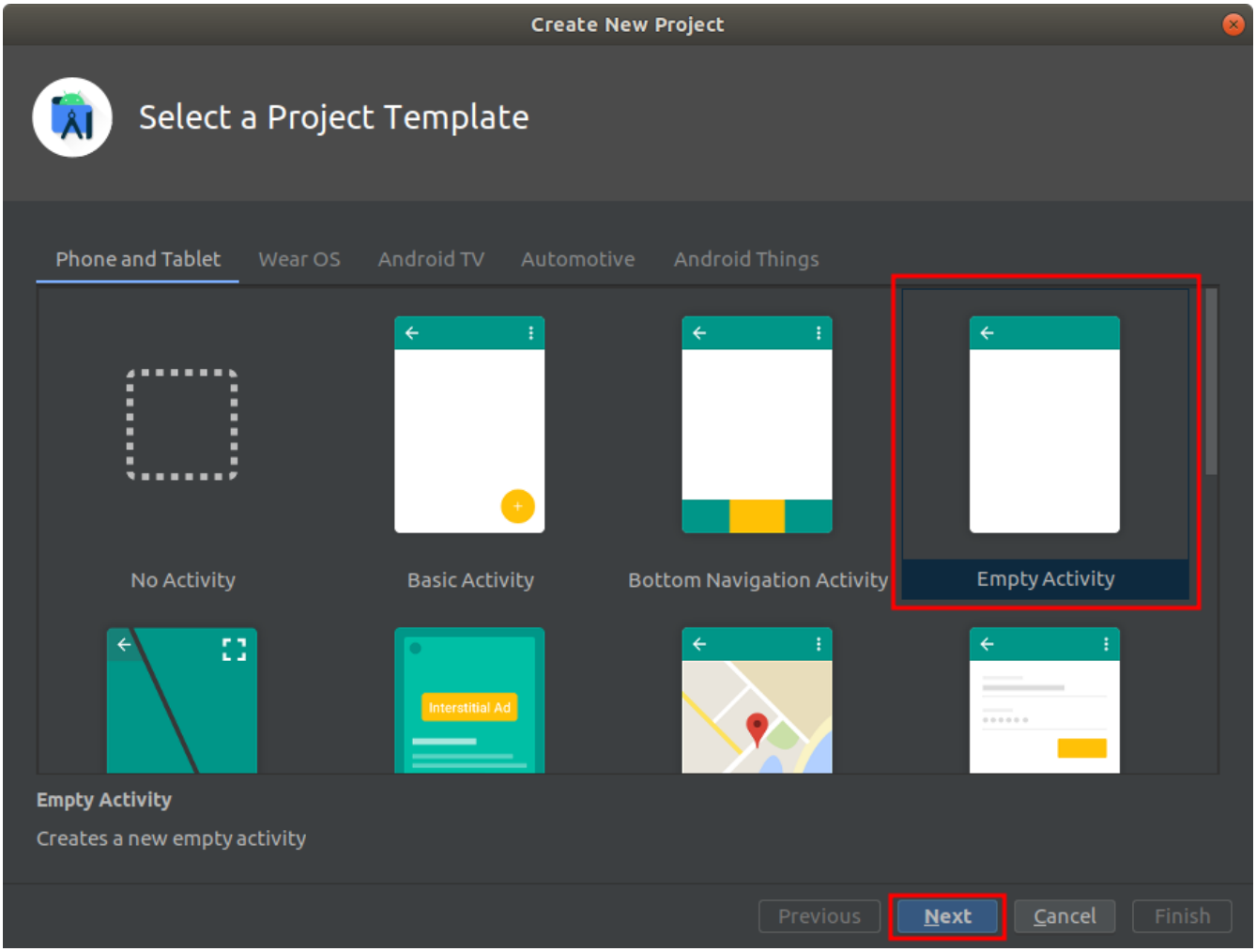
Lien : Installation Android Studio 4.1 (Ubuntu 20.04)

2. Premier projet

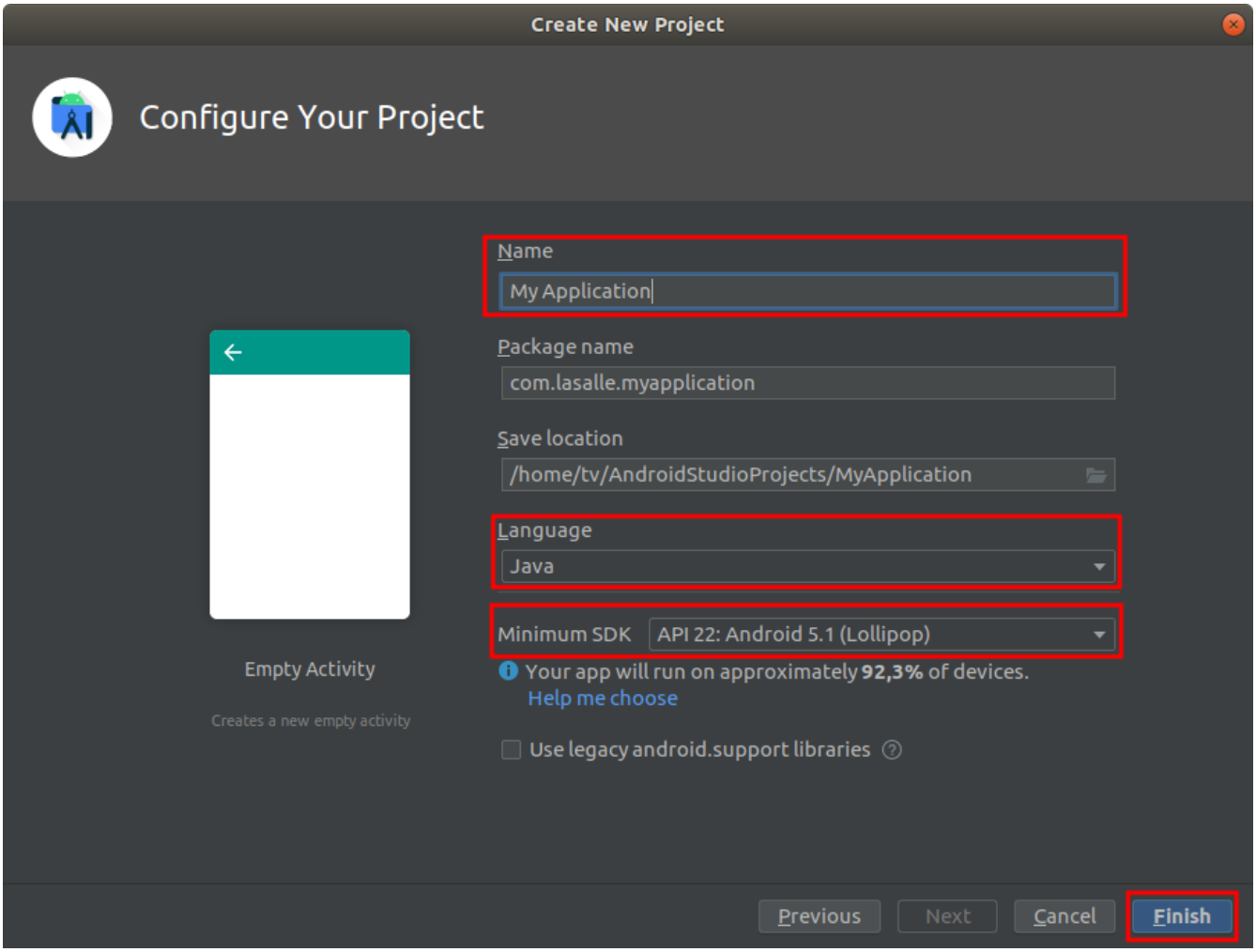
On démarre Android Studio et on sélectionne `Create New Project` :



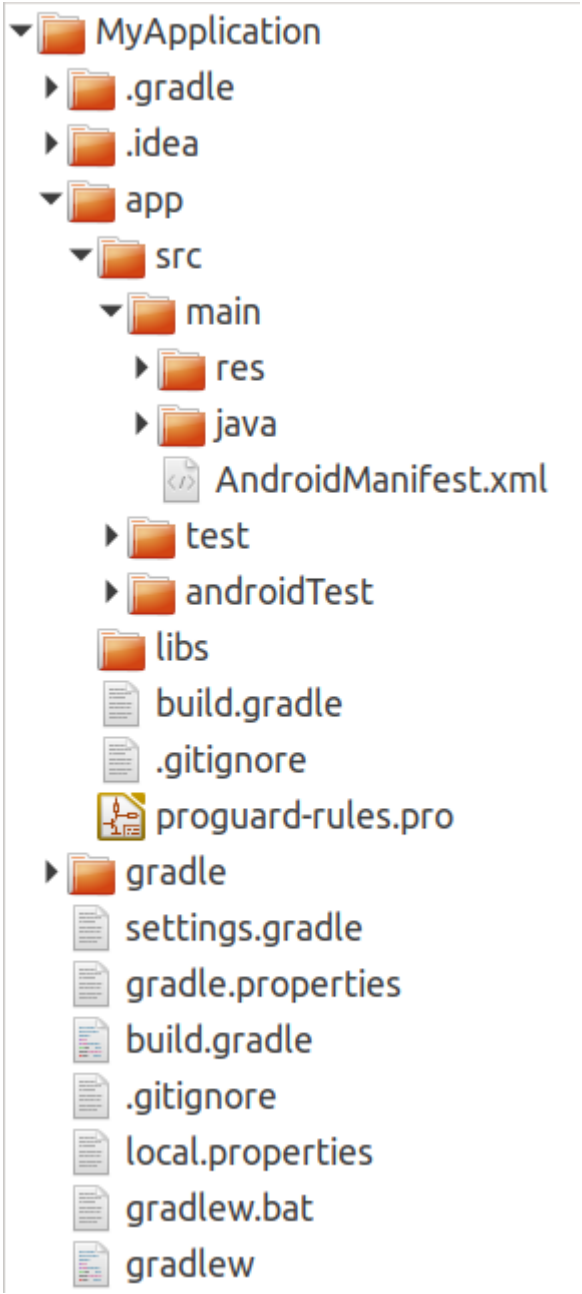
Android Studio fournit plusieurs *templates* de base pour les applications pour smartphones et tablettes tactiles (`Phone and Tablet`) mais il est conseillé de choisir `Empty Activity` :



On configure ensuite son nouveau projet en donnant un **nom** à l'application et en choisissant le langage **Java** et le **niveau d'API** :



Le projet a été créé par Android Studio. Cela a généré une structure composée de plusieurs fichiers :



La structure d'un projet Android comprend essentiellement :

- l'application Android qui est stockée dans un répertoire `app`
- l'outil `gradle` qui est le moteur de production permettant de construire le projet Android
- la personnalisation de l'IDE qui est conservée dans le répertoire `.idea`

Avant de commencer à travailler avec Android Studio, il est nécessaire d'initialiser correctement son dépôt `git` ou `svn`. Les fichiers à ne pas conserver dans son dépôt (*repository*) sont précisés dans les fichiers `.gitignore` pour `git`. Il faut donc appliquer la même chose pour **Subversion** (`svn`).

À la racine de son projet Android :

```
$ vim .svnignore
*.iml
.gradle
local.properties
.idea/caches
.idea/libraries
.idea/modules.xml
.idea/workspace.xml
.idea/navEditor.xml
.idea/assetWizardSettings.xml
.idea/shelf
app/build
.DS_Store
build
captures
.externalNativeBuild
.cxx
local.properties

$ svn propset svn:ignore -F .svnignore .
$ svn commit --message "Fichiers à ignorer"
$ svn proplist -v
```

Sinon, il est possible de faire un *shell* script qui force les suppressions des fichiers à ignorer dans le dépôt `svn` :

```
#!/bin/bash
svn del --force *.iml
svn del --force .gradle
svn del --force local.properties
svn del --force .idea/caches
svn del --force .idea/libraries
svn del --force .idea/modules.xml
svn del --force .idea/workspace.xml
svn del --force .idea/navEditor.xml
svn del --force .idea/assetWizardSettings.xml
svn del --force .idea/shelf
svn del --force app/build
svn del --force .DS_Store
svn del --force build
svn del --force captures
svn del --force .externalNativeBuild
svn del --force .cxx
svn del --force local.properties
```

Ou tout simplement un *shell* script qui supprime ces fichiers à ignorer :

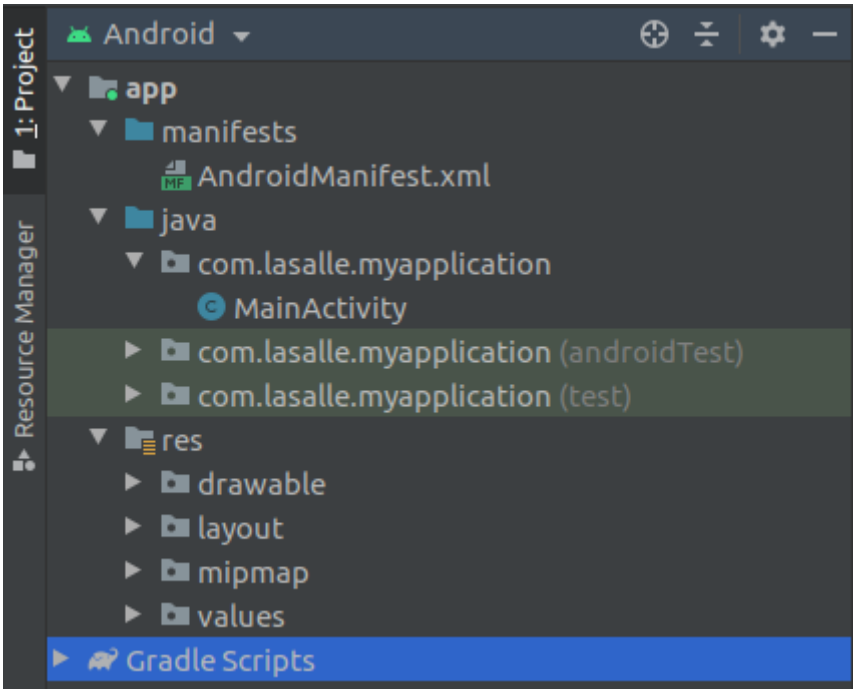
```
#!/bin/bash
if [ $# -lt 1 ]
then
    echo "Usage: $0 <dossier>"
    exit
fi

dossier=$1

if [ ! -d $dossier ]
then
    echo "Erreur: $dossier introuvable"
    exit
fi

rm -rf $dossier/*.iml
rm -rf $dossier/.gradle
rm -rf $dossier/local.properties
rm -rf $dossier/.idea/caches
rm -rf $dossier/.idea/libraries
rm -rf $dossier/.idea/modules.xml
rm -rf $dossier/.idea/workspace.xml
rm -rf $dossier/.idea/navEditor.xml
rm -rf $dossier/.idea/assetWizardSettings.xml
rm -rf $dossier/.idea/shelf
rm -rf $dossier/app/build
rm -rf $dossier/.DS_Store
rm -rf $dossier/build
rm -rf $dossier/captures
rm -rf $dossier/.externalNativeBuild
rm -rf $dossier/.cxx
rm -rf $dossier/local.properties
```

3. L'environnement de développement intégré Android Studio 4.1

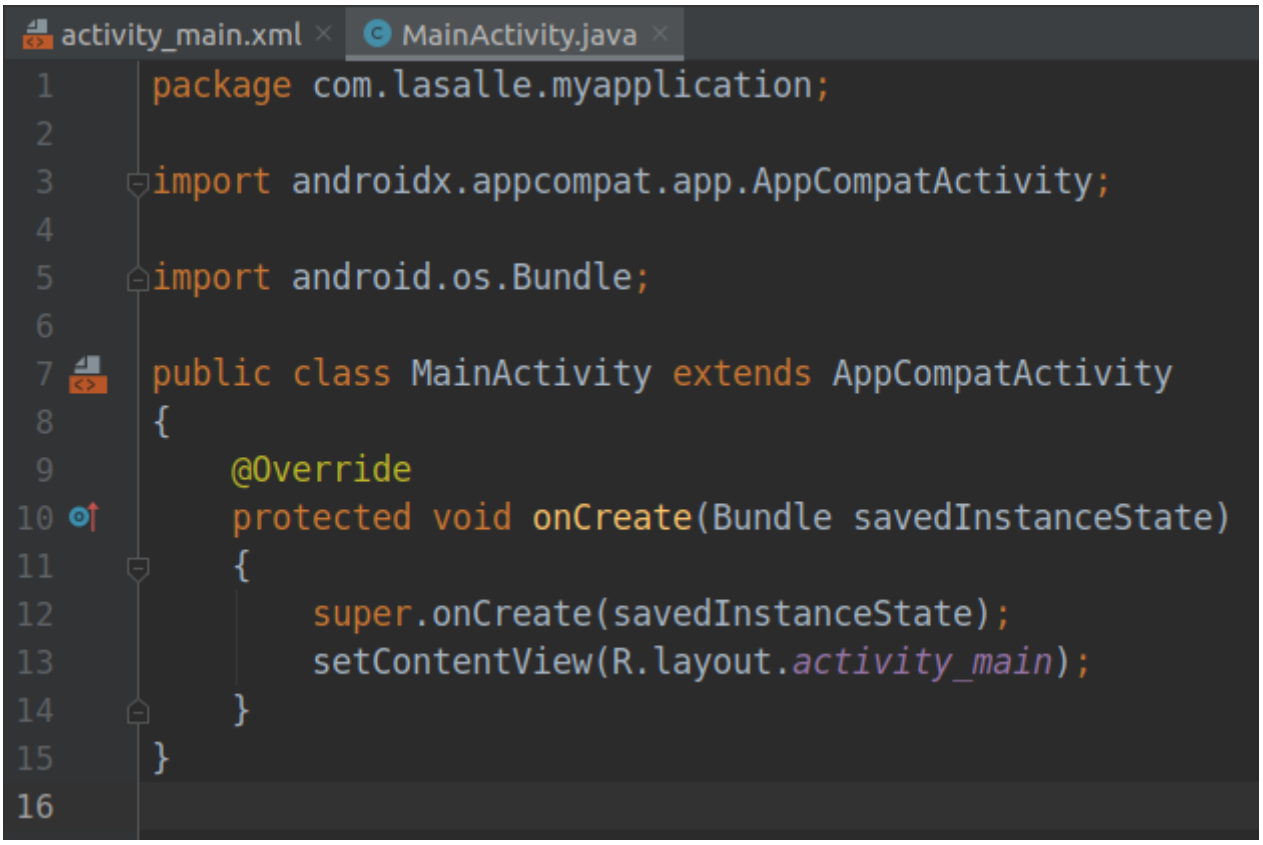
Android Studio est un environnement de développement intégré (IDE) que l'on peut qualifier de très complet !

Petit tour d'horizon des différentes fenêtres :

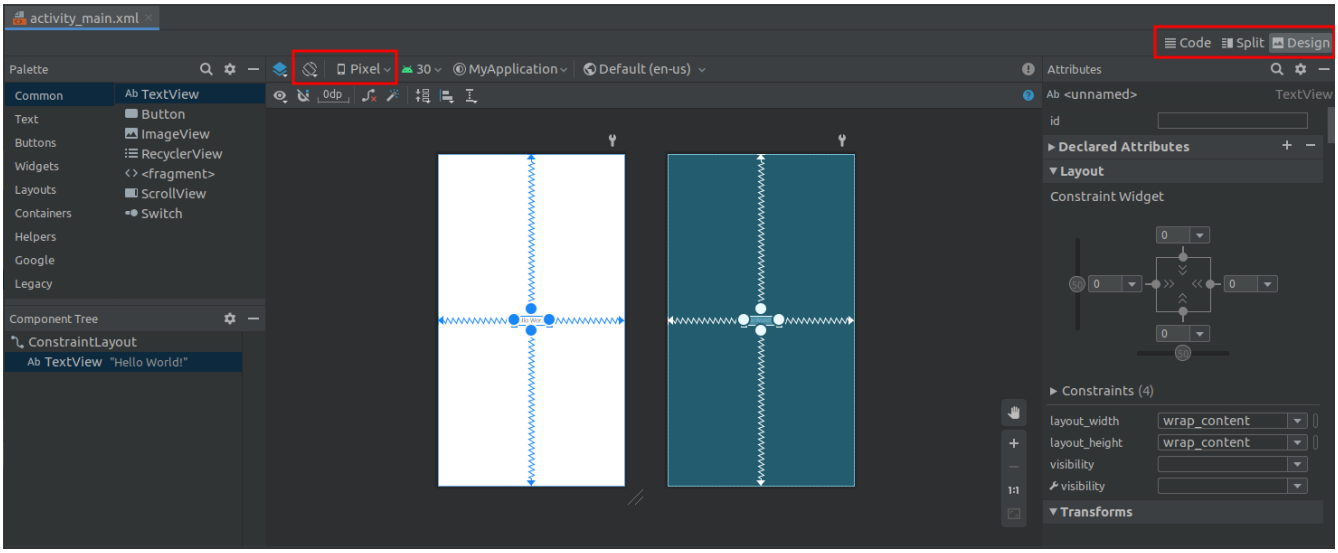
- la fenêtre du projet Android en haut à droite permet d'accèder rapidement aux trois composantes qui sont le code source Java (les classes de l'application), les ressources GUI (`res`) et le fichier `AndroidManifest.xml` :



- la fenêtre centrale est bien évidemment la zone d'édition des fichiers de l'application. Par exemple pour un code source Java :

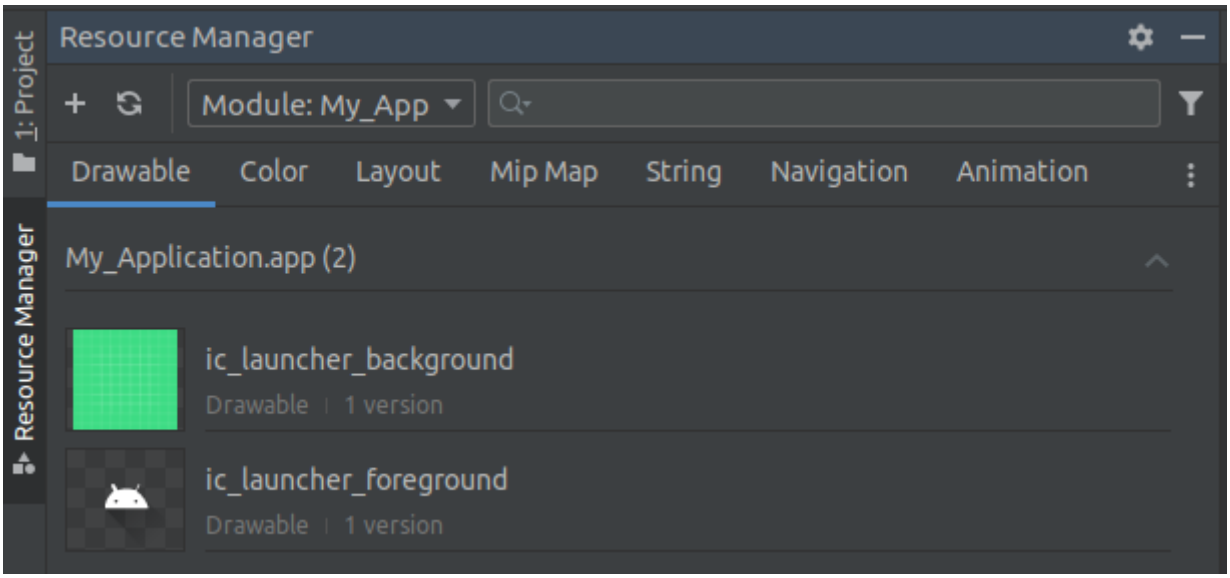


- le fenêtre centrale d'édition change de comportement pour une ressource GUI. On retrouve classiquement les *widgets* et *layouts* (ici à droite), une prévisualisation (au centre) que l'on peut personnaliser pour un modèle (ici un `Pixel`) et l'édition des propriétés (à gauche) :

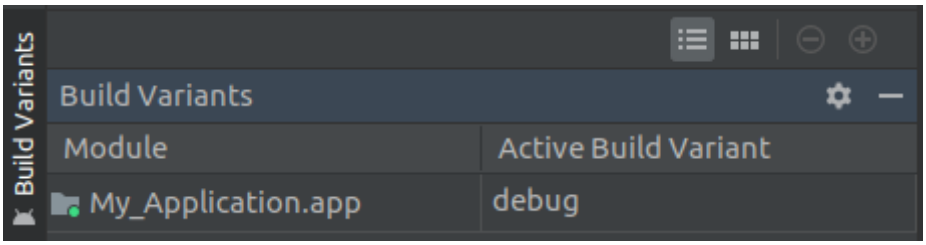


Les ressources GUI (*Graphical User Interface*) sont en réalité de simples fichiers XML (*Extensible Markup Language*) que l'on peut aussi éditer directement (onglets `Code`/`Design`).

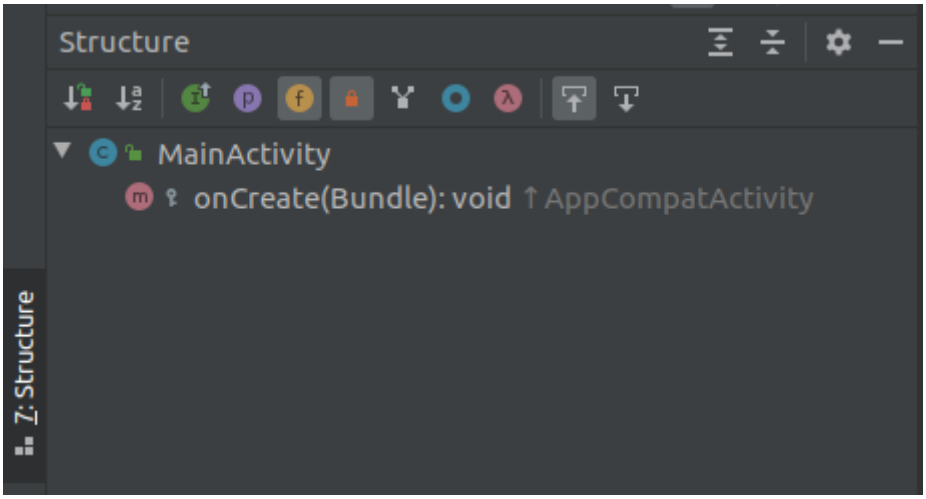
- il est possible d'utiliser la fenêtre de gestion des ressources GUI :



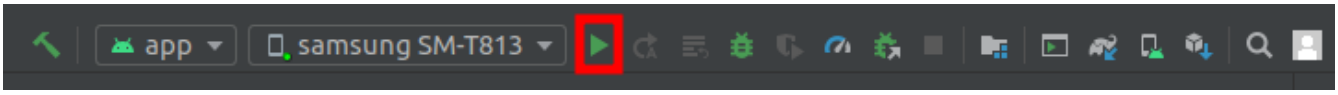
- il en va de même de la fenêtre de gestion des versions (*debug*/*release*) :



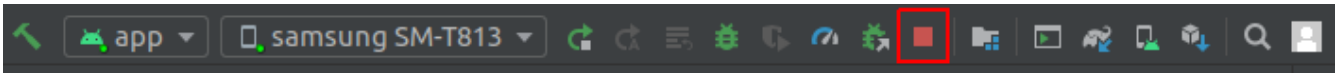
- mais probablement la plus utile est la fenêtre de structure qui permet de naviguer facilement dans le code source Java (attributs et méthodes) :



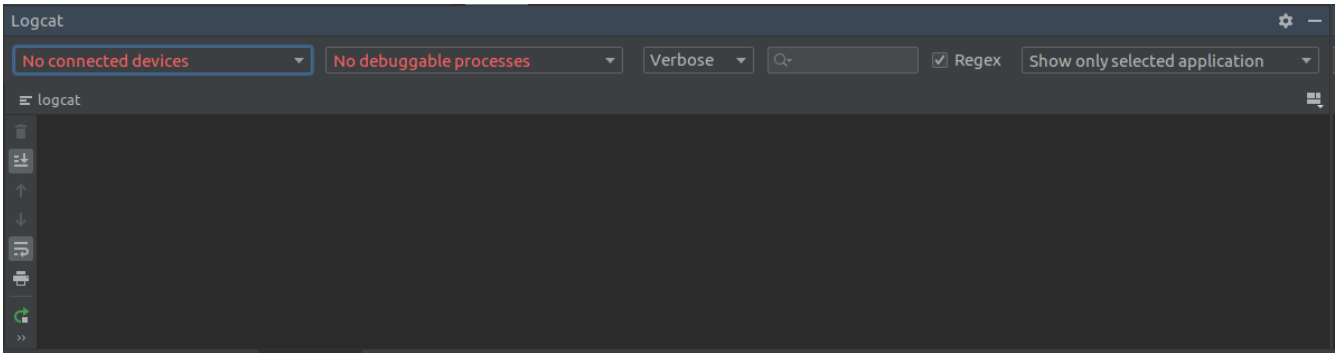
Ensuite, on utilisera fréquemment la barre d'outil qui permet notamment de lancer l'exécution de l'application (cliquer sur la flèche `Run app`) :



Pour arrêter l'application :



Lors de l'exécution de l'application en phase *debug*, on aura besoin obligatoirement de la fenêtre de *log* en bas :



Evidemment, il y a aussi de nombreux menus que l'on découvrira au fur et à mesure.

4. Notions de base

4.1. Android

Le *framework* Android permet :

- de définir des interfaces utilisateurs graphiques (en XML)
- de créer des activités (composant de base d'une application *Activity*)
- de fournir et partager des données (composant *Content Providers*)

- d'interagir avec le système (composants *Intents*, *Broadcast Receivers*, *Services*, ...)

Android permet aussi d'accéder aux fonctionnalités avancées du *smartphone* / tablette :

- Stockage (fichiers, base de données, ...)
- Communication Réseau (Wifi, Bluetooth, 3G/4G, ...)
- Multimédia (audio, photo, caméra)
- GPS
- Téléphonie (appels, SMS)

Une application Android est écrite en **Java** ou en Kotlin.

Des parties bas-niveau seront écrites en C/C++.

À chaque version du système est associé un **niveau d'API** (Exemple : Android 4.4 KITKAT → API 19).

4.2. Gradle

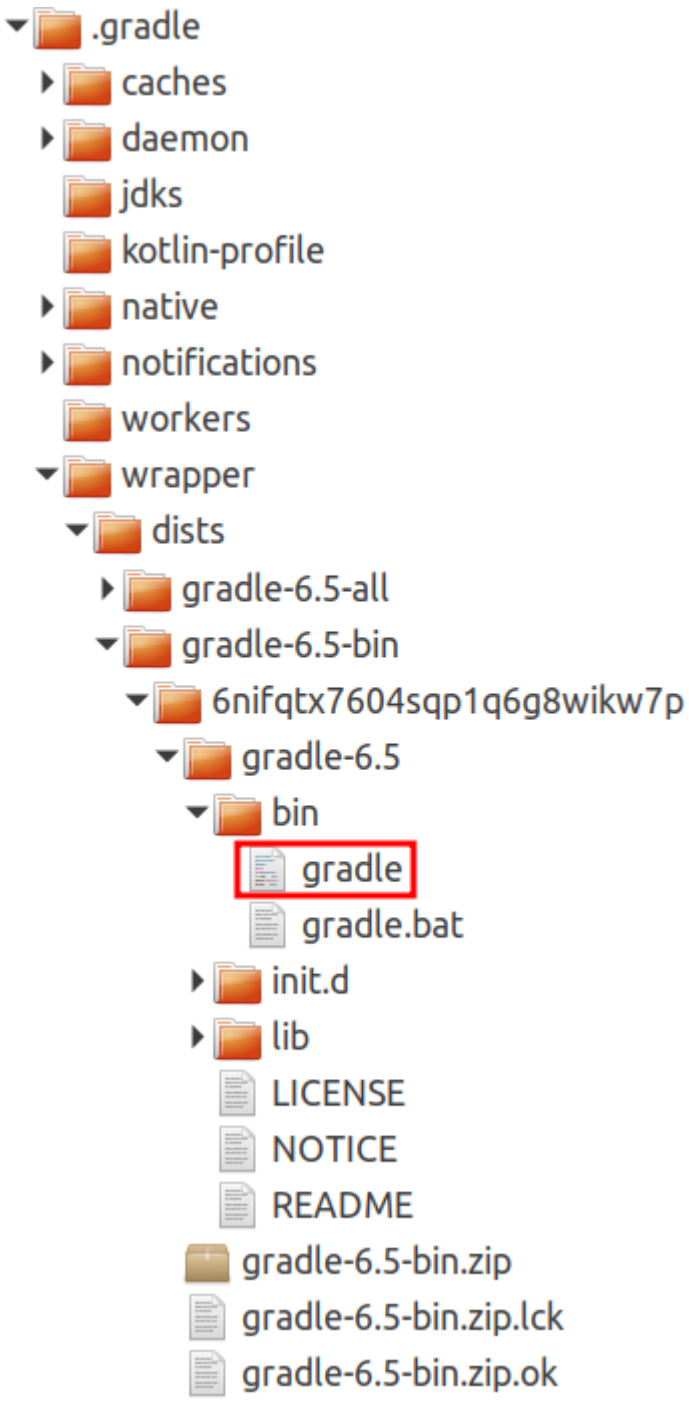
L'outil `gradle` est le moteur de production qui permet de construire (`build`) le projet Android.

C'est un outil comparable `make`, ou `ant` pour Java.

Son usage est **"transparent"** dans l'IDE Android Studio mais on va quand même apporter quelques précisions.

Gradle fonctionne à base de tâches décrivant le processus de construction. Il est souvent utilisé à travers un *wrapper* (le script `gradlew`).

Il arrive fréquemment qu'Android Studio télécharge automatiquement une nouvelle version de `gradle` pour le projet. Ces informations sont stockées dans le fichier `$HOME/AndroidStudioProjects/MyApplication/gradle/wrapper/gradle-wrapper.properties` qui précise notamment l'URL de téléchargement et le chemin d'installation (qui normalement est `$HOME/.gradle/wrapper/dists`). Par exemple, pour une version 6.5 :



Bien évidemment, plusieurs versions de `gradle` de peuvent cohabiter.

Les fichiers utilisées par `gradle` sont au nombre de quatre :

- pour la configuration :
  - `$HOME/AndroidStudioProjects/MyApplication/gradle.properties`
  - `$HOME/AndroidStudioProjects/MyApplication/settings.gradle`

```
$ cd $HOME/AndroidStudioProjects/MyApplication

$ cat gradle.properties | grep -E "^[^#]"
org.gradle.jvmargs=-Xmx2048m -Dfile.encoding=UTF-8
android.useAndroidX=true
android.enableJetifier=true

$ cat settings.gradle
include ':app'
rootProject.name = "My Application"
```

- pour la fabrication :
  - $HOME/AndroidStudioProjects/MyApplication/build.gradle
  - $HOME/AndroidStudioProjects/MyApplication/app/build.gradle

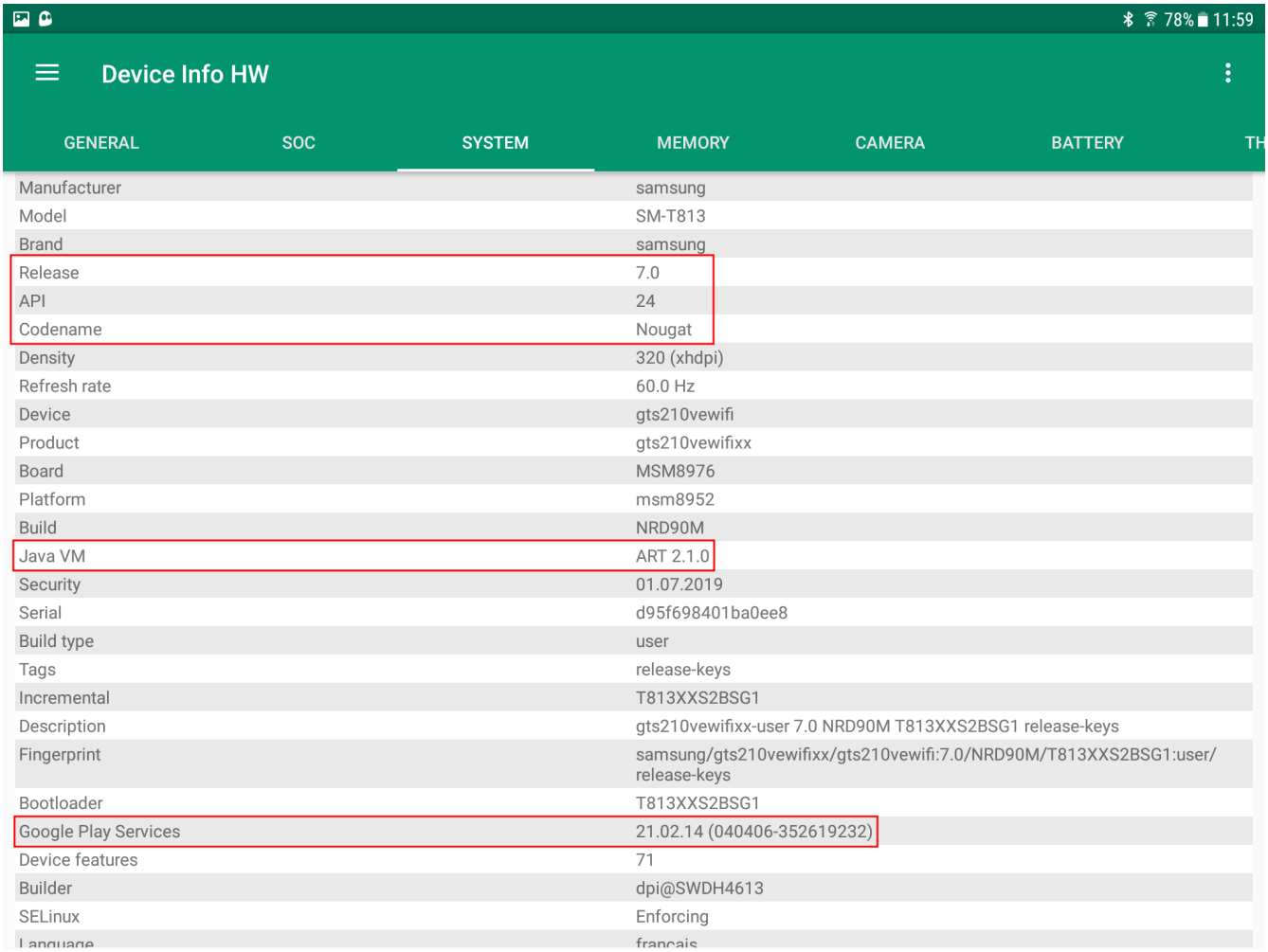
```
$ cd $HOME/AndroidStudioProjects/MyApplication

$ cat build.gradle
// Top-level build file where you can add configuration options common to all sub-
projects/modules.
buildscript {
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath "com.android.tools.build:gradle:4.1.2"

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}

$ cat app/build.gradle
plugins {
    id 'com.android.application'
}
```

```
android {
    compileSdkVersion 30
    buildToolsVersion "30.0.3"

    defaultConfig {
        applicationId "com.lasalle.myapplication"
        minSdkVersion 22
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}

dependencies {
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.2.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
}
```

En cours de projet, on aura probablement besoin de modifier le fichier `app/build.gradle` pour spécifier :

- les niveaux d'API
- les dépendances vers des bibliothèques

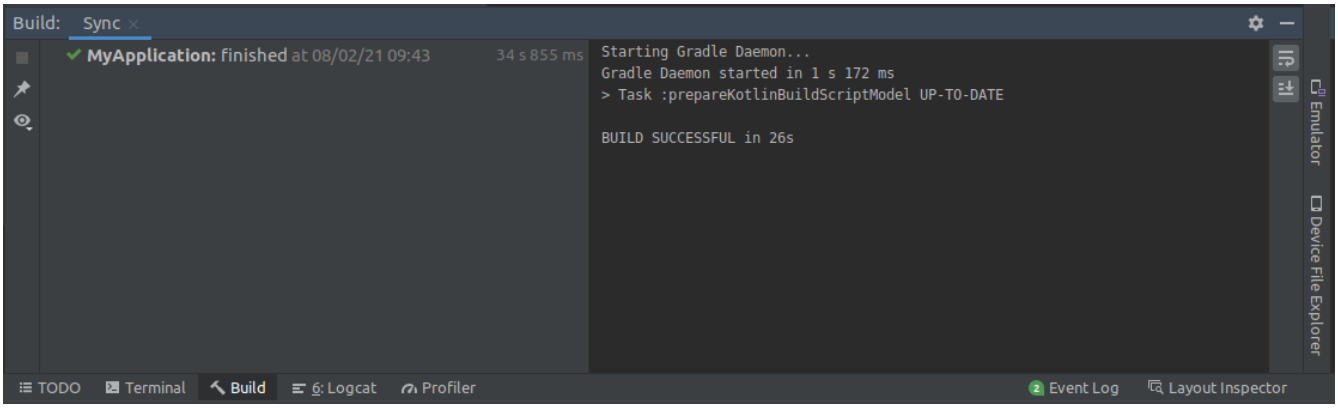
Une application comme `Device Info HW` peut être utile pour collecter des informations sur le *smartphone* / tablette :



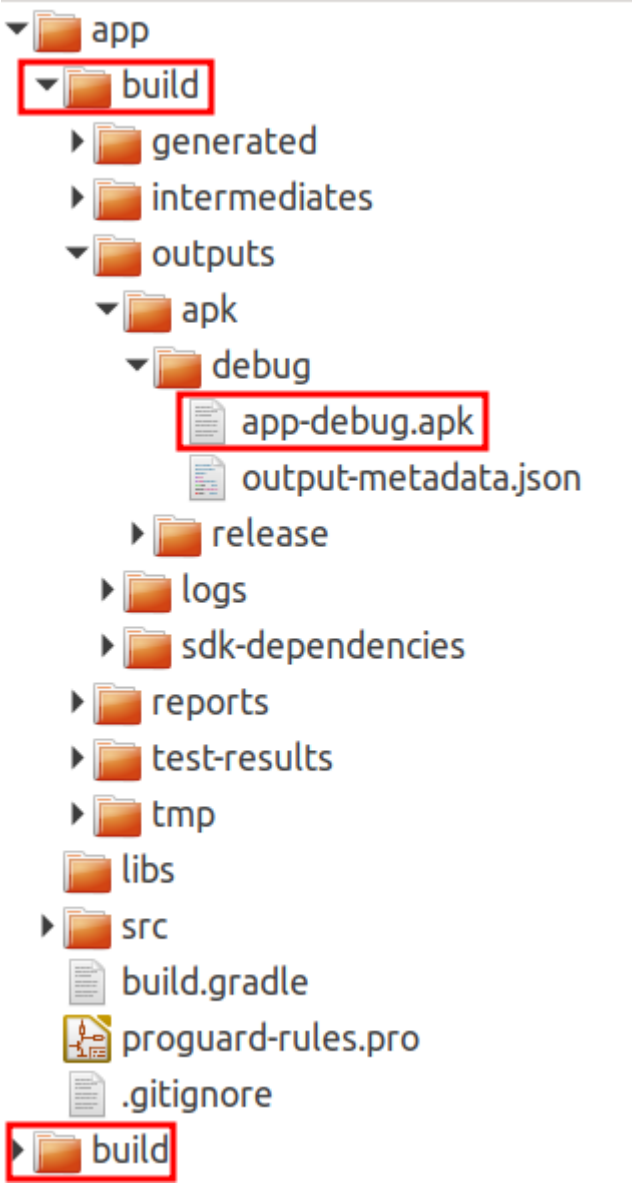
Exemple de fabrication de l'application à partir de la ligne de commande :

```
$ ~/.gradle/wrapper/dists/gradle-6.5-bin/6nifqtx7604sqp1q6g8wikw7p/gradle-
6.5/bin/gradle build

Welcome to Gradle 6.5!

Here are the highlights of this release:
 - Experimental file-system watching
 - Improved version ordering
 - New samples

For more details see https://docs.gradle.org/6.5/release-notes.html

Starting a Gradle Daemon, 1 incompatible Daemon could not be reused, use --status for
details

> Task :app:lint
Ran lint on variant debug: 5 issues found
Ran lint on variant release: 3 issues found
Wrote HTML report to
file:///home/tv/AndroidStudioProjects/MyApplication/app/build/reports/lint-
results.html
Wrote XML report to
file:///home/tv/AndroidStudioProjects/MyApplication/app/build/reports/lint-results.xml

BUILD SUCCESSFUL in 44s
61 actionable tasks: 61 executed
```

On obtient la même chose (sans effort) dans l'IDE Android Studio :

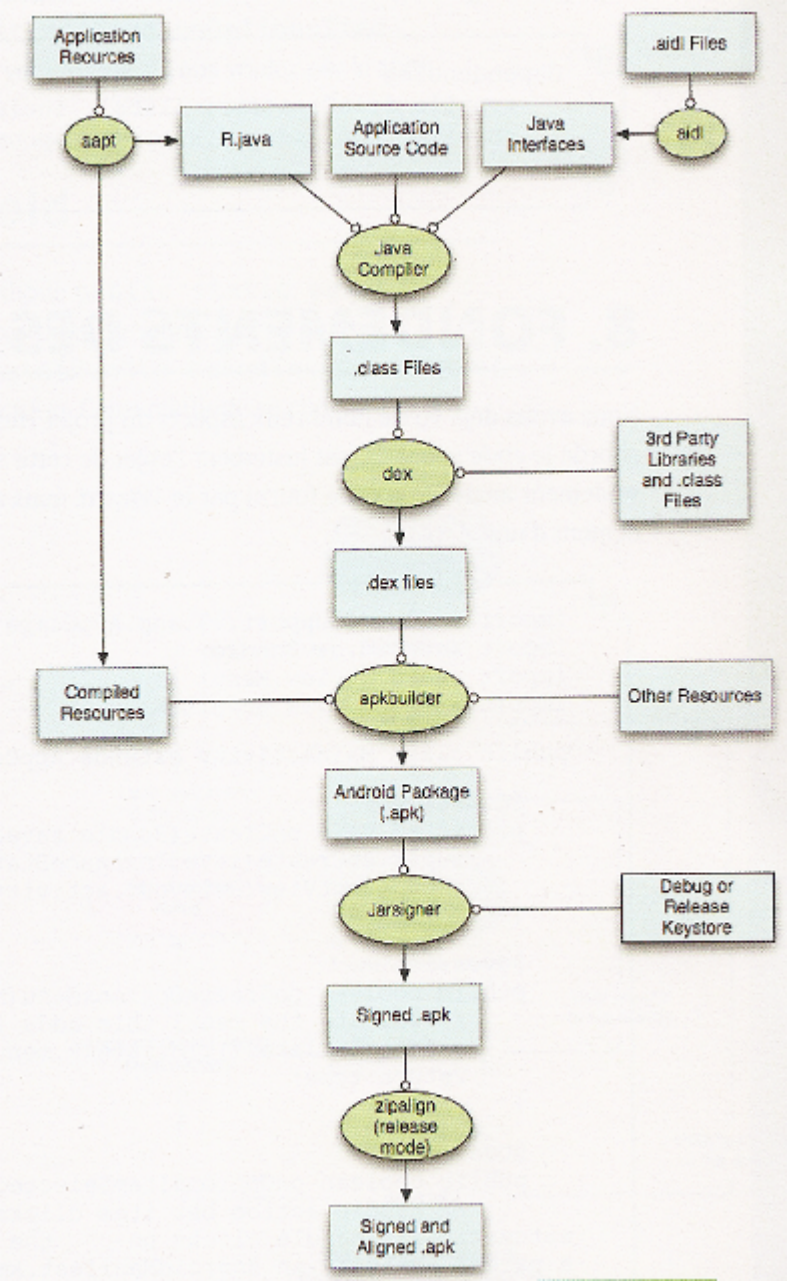


`gradle` a généré deux répertoires de `build` et il a bien fabriqué l' `apk` de l'application Android (ici en version *debug*) :



Dans Android Studio, la phase de *build* et déploiement est très simple puisqu'elle consiste simplement à cliquer sur le bouton Run !

En fait le processus est plus complexe :



Au final, un fichier apk est une simple archive Zip :

```
$ file app/build/outputs/apk/debug/app-debug.apk
app/build/outputs/apk/debug/app-debug.apk: Zip archive data
```

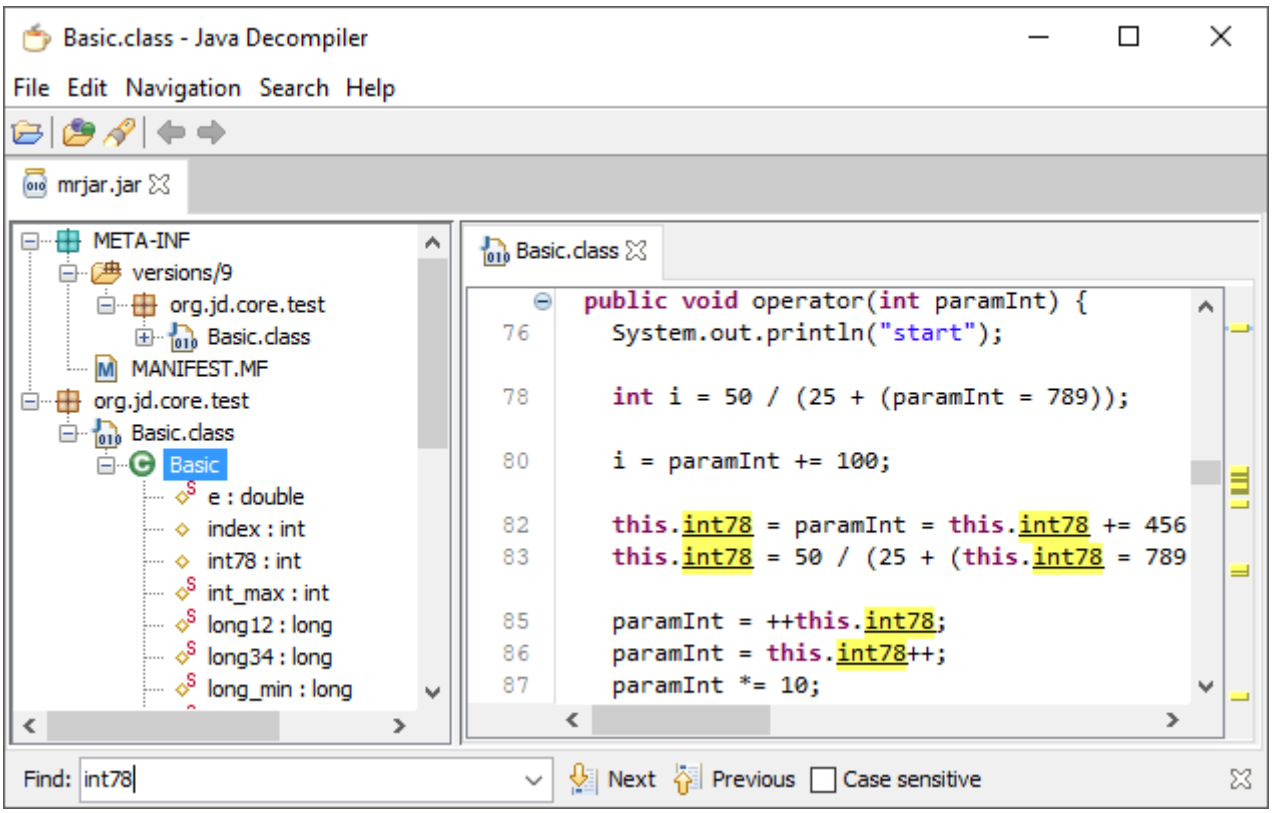
Reverse engineering :

```
$ unzip app/build/outputs/apk/debug/app-debug.apk -d extract
$ tree extract
extract
├──── AndroidManifest.xml
├──── classes2.dex
├──── classes.dex
├──── META-INF
...
├──── res
│    ├──── anim
│    │    ├──── abc_fade_in.xml
...
```

On retrouve le fichier `AndroidManifest.xml`, l'ensemble des ressources dans le répertoire `res` et le *bytecode* dans les fichiers d'extension `.dex`.

L'utilitaire dex2jar permet de convertir les fichiers `.dex` en fichiers Java `.class` archivés dans un fichier `jar`.

On peut ensuite utiliser un décompilateur comme JD-GUI pour parcourir le code source des classes Java de l'application :



4.3. Le manifeste AndroidManifest.xml

Chaque projet Android doit posséder un fichier XML nommé `AndroidManifest.xml` (localisé dans `app/src/main/` et stocké à la racine de la hiérarchie du projet).

Le manifeste décrit les composants, leurs interactions et les métadonnées de l'application, dont

notamment ses exigences en matière de plateforme et de matériel. Il contient généralement :

- le nom du *package* de l'application (un identifiant unique)
- tous les composants de l'application (les activités mais aussi les `Services`, `BroadCast Receivers`, ...)
- les classes qui implémentent les composants et leurs capacités (par exemple les `Intents`)
- les permissions nécessaires pour l'application
- les informations sur les versions d'API pour exécuter l'application
- les bibliothèques utilisées par l'application

Le fichier `AndroidManifest.xml` de l'exemple de base :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.lasalle.myapplication">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApplication">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Ici, il précise par exemple :

- le nom, l'icône et le thème
- l'activité principale à démarrer (`MainActivity`)

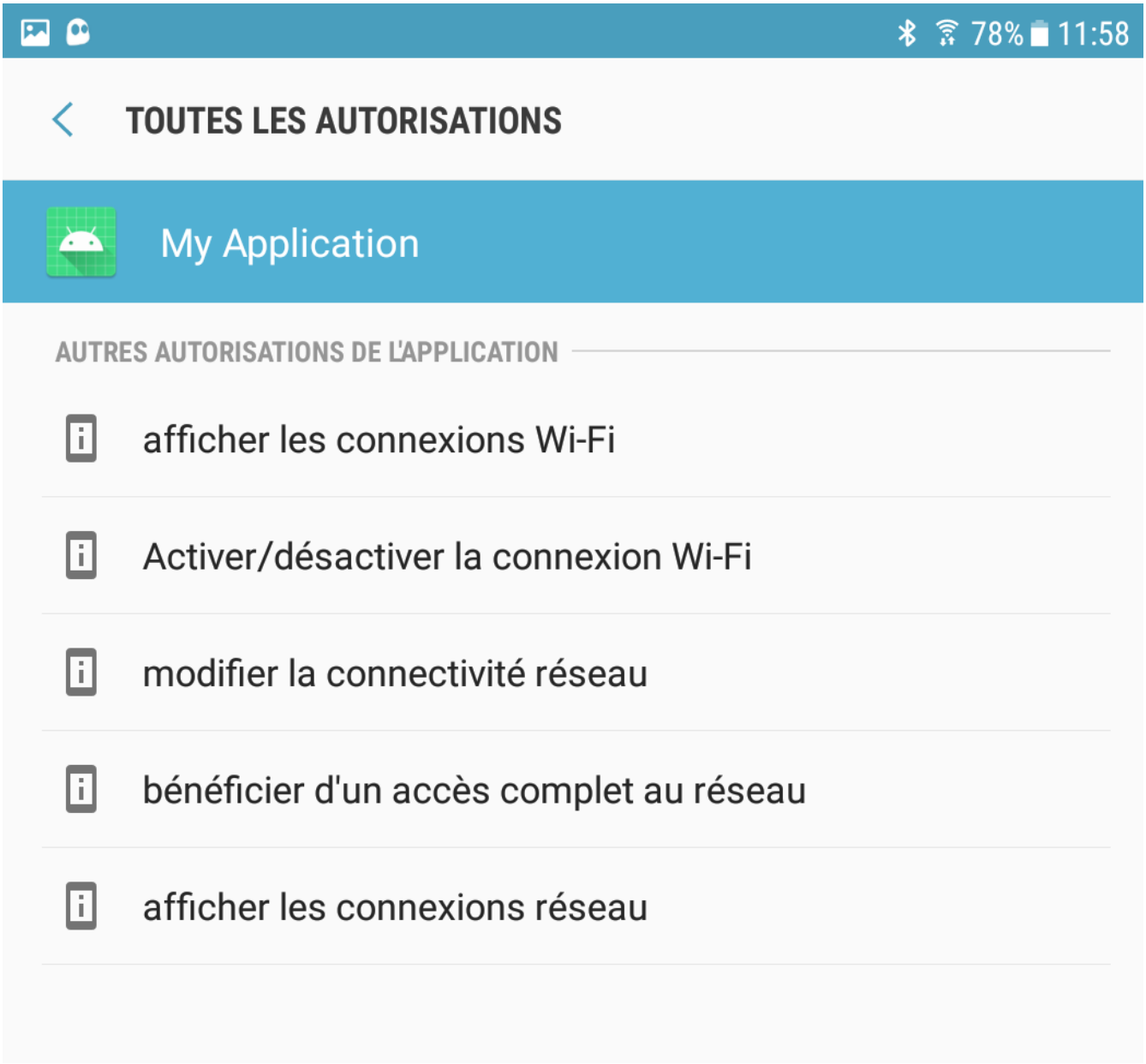
En cours de projet, on aura probablement besoin de modifier ce fichier `AndroidManifest.xml` pour spécifier :

- des permissions (WiFi, Internet, ...)
- des nouvelles activités

Par exemple :

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

    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApplication">

        <activity android:name=".Activity1" />
        <activity android:name=".Activity2" />
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
</manifest>
```

Ces autorisations pourront être visibles dans les paramètres `Applications` du *smartphone* / tablette :



4.4. Les activités

La notion d'activité est fondamentale dans le développement d'une application Android.

Une activité (*activity*) est la composante principale d'une application sous Android. L'activité est une fonctionnalité (couche présentation) de l'application et elle possède généralement une vue (`View`) au minimum, c'est-à-dire une interface graphique utilisateur unique (un "écran").

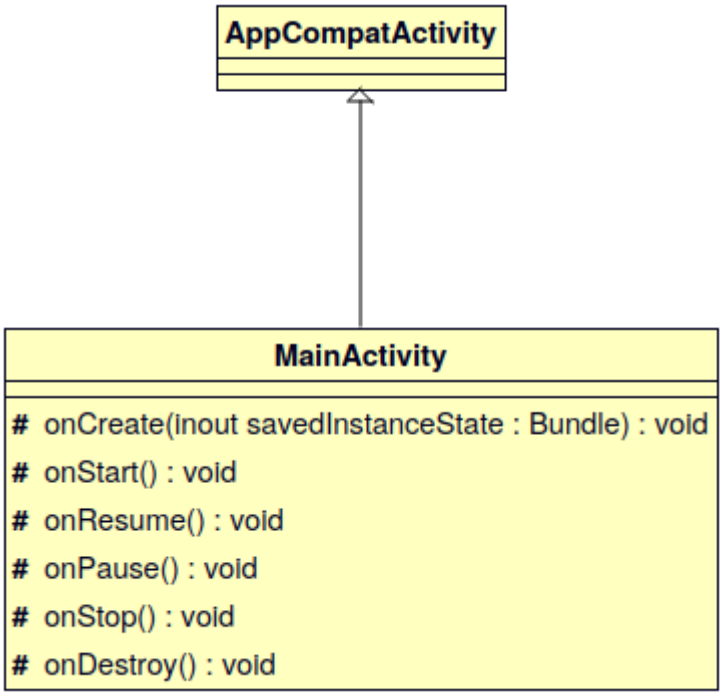
Une application contient en général plusieurs activités. Par exemple dans une application de domotique, une activité permettrait de lister les pièces d'une habitation, puis une autre activité pour ajouter un nouvel appareil à piloter, et encore une activité pour afficher le détail d'une zone. Chaque activité peut fonctionner de manière autonome.

Chaque activité doit impérativement être déclarée dans le fichier `AndroidManifest.xml` dans une balise `<activity>` :

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.lasalle.myapplication">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApplication">
        <activity android:name=".OtherActivity" />
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Comme une seule activité peut être lancée au démarrage d'une application, il faut l'indiquer avec la balise `<intent-filter>`. Celle-ci contient deux balises `<action>` (**MAIN**) et `<category>` (**LAUNCHER**). On la nomme activité principale de l'application.

Sauf exception, une application Android comporte au moins une classe héritant de `Activity` (ou `AppCompatActivity` pour des raisons de compatibilité) et peut évidemment en avoir plusieurs.

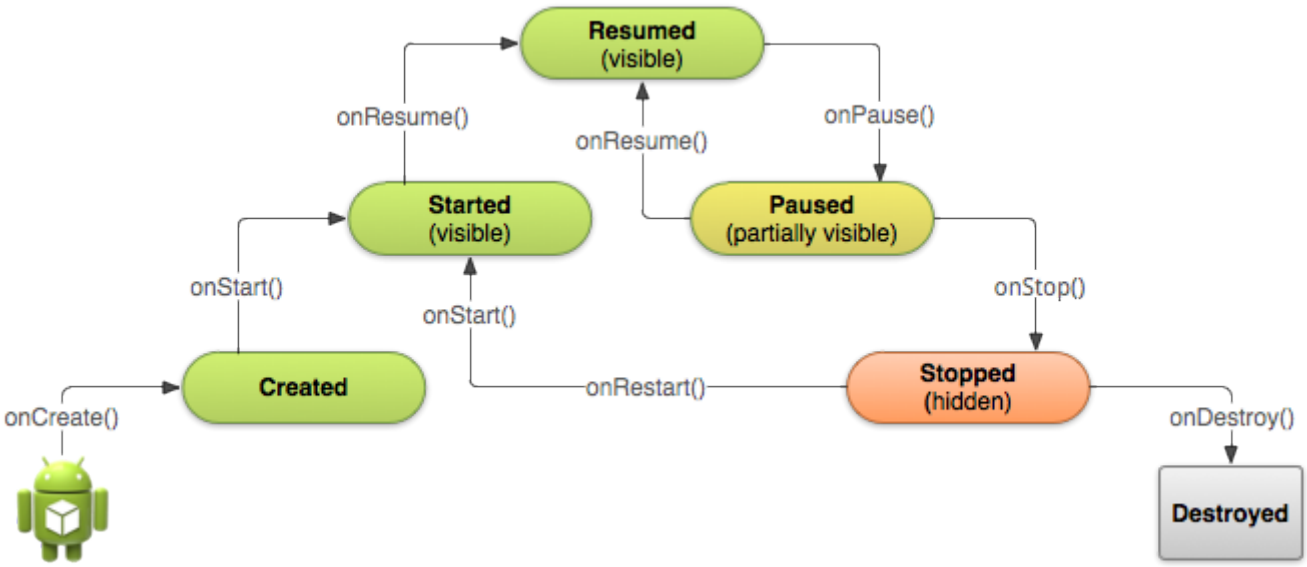


```
public class MainActivity extends AppCompatActivity
{
    // ....
}
```

Une activité a un **cycle de vie** contrôlé par le système Android : il la crée, il la démarre, ... il la détruit. Une activité est toujours dans l'un des quatre états suivants : active, en pause, stoppée ou morte.

Sur un *smartphone* ou tablette, les ressources (et notamment la mémoire) sont plus limitées. Des activités peuvent être tuées pour permettre à d'autres de s'exécuter. Par exemple lors d'un changement d'orientation, Android supprime et recrée toutes les activités. D'autre part, on ne peut pas effectuer des traitements consommateurs en temps dans une activité (le *thread* UI) car celle-ci se "figerait" et ne répondrait plus aux actions de l'utilisateur. Il faut savoir que si une activité ne réagit en plus de cinq secondes, elle sera tuée par l'`ActivityManager` qui la considérera comme morte.

En fonction de l'état de l'activité, il est possible de contrôler celle-ci à l'aide des méthodes héritées de la classe `Activity` (ce sont des méthodes appelées lors de la transition d'un état à un autre) :



Il est conseillé de "loguer" le cycle de vie de l'activité :

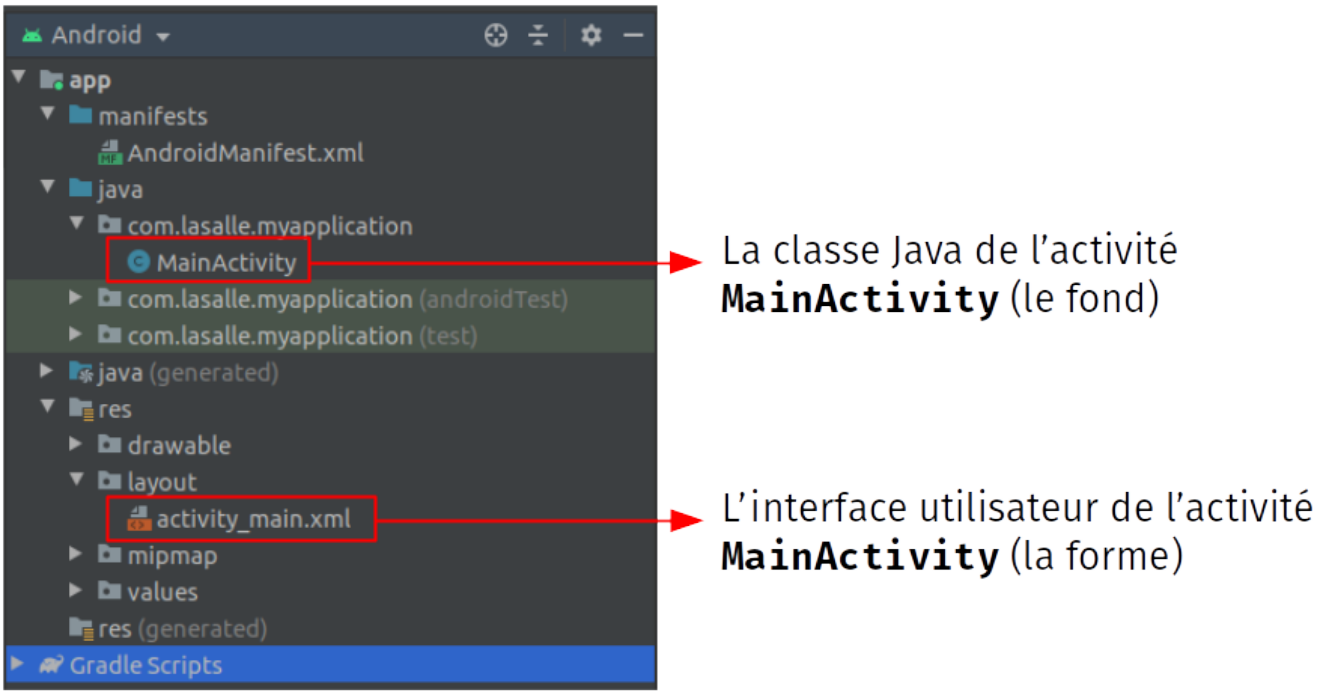
```
package com.lasalle.myapplication;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends AppCompatActivity
{
    private static final String TAG = "MainActivity";
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        Log.i(TAG,"onCreate()");

        // associe le fichier d'interface activity_main.xml à l'activité MainActivity
        setContentView(R.layout.activity_main);
    }

    /**
     * @brief Méthode appelée au démarrage après le onCreate() ou un restart après un
onStop()
     */
    @Override
    protected void onStart()
    {
        super.onStart();
        Log.i(TAG,"onStart()");
    }

    /**
     * @brief Méthode appelée après onStart() ou après onPause()
     */
    @Override
    protected void onResume()
    {
        super.onResume();
        Log.i(TAG,"onResume()");
    }

    /**
     * @brief Méthode appelée après qu'une boîte de dialogue s'est affichée (on
reprend sur un onResume()) ou avant onStop() (activité plus visible)
     */
    @Override
    protected void onPause()
    {
        super.onPause();
        Log.i(TAG,"onPause()");
    }

    /**
     * @brief Méthode appelée lorsque l'activité n'est plus visible
     */
    @Override
    protected void onStop()
    {
        super.onStop();
        Log.i(TAG,"onStop()");
```

```
    }

    /**
     * @brief Méthode appelée à la destruction de l'application (après onStop() et
détruite par le système Android)
     */
    @Override
    protected void onDestroy()
    {
        super.onDestroy();
        Log.i(TAG,"onDestroy()");

    }
}
```

Android permet de sauvegarder et restaurer l'état d'une instance. La méthode `onSaveInstanceState()` fournit un `Bundle` dans lequel une activité peut sauver des données. La méthode `onCreate()` reçoit en argument le `Bundle` réalisé précédemment.

Android met aussi en oeuvre une séparation entre la présentation (la GUI) et son comportement (le code qui la contrôle) :



Les interfaces graphiques sont des *layouts* XML qui sont stockés dans `res/layout`.

Exemple de fichier XML de l'activité principale (un simple affichage d'un texte à l'écran) :

```
<?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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

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

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

Chaque fichier de disposition XML est compilé dans une ressource `View`. Il faut charger cette ressource à partir du code de l'activité (dans `onCreate()` généralement). Pour réaliser cela, on appelle `setContentView()` en lui passant la référence de la ressource de mise en page sous la forme : `R.layout.layout_file_name` :

```
public class MainActivity extends AppCompatActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        // associe le fichier d'interface activity_main.xml à l'activité MainActivity
        setContentView(R.layout.activity_main);
    }
}
```

La méthode de rappel (*callback*) `onCreate()` d'une activité est appelée par le *framework* Android lorsque l'activité est lancée.

La classe `MainActivity` peut interagir avec son interface graphique.

Chaque composant possède un **ID** afin de l'identifier de manière unique dans l'arborescence.

Lorsque l'application est compilée, cet ID est référencé sous la forme d'un entier, mais l'ID est généralement attribué dans le fichier XML sous la forme d'une chaîne dans l'attribut `id` :

```
<TextView
        android:id="@+id/texteMessage"
        ...  />
```

Il est donc possible d'interagir avec un composant de la GUI en utilisant son **ID** :

```
public class MainActivity extends AppCompatActivity
{
    private TextView texteMessage;

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

        texteMessage = (TextView)findViewById(R.id.texteMessage);

        texteMessage.setText("Bonjour le monde !");
    }
}
```

Les actions de l'utilisateur (par exemple, un clic) sont abordées notamment dans les documents GUI Android et Activité n°1 : IHM. Elles nécessitent un gestionnaire d'écoute (`Listener`).

4.5. La GUI

Tous les éléments d'une GUI (*Graphical User Interface*) sont des objets `View` et `ViewGroup` :

- Les objets `View` sont généralement appelés des ***widgets*** (comme `Button` ou `TextView` par exemple). Ils représentent quelque chose que l'utilisateur peut voir.
- Les objets `ViewGroup` sont généralement appelés ***layout*** (comme `LinearLayout` ou `ConstraintLayout` par exemple). Ce sont des conteneurs invisibles qui définissent la structure de mise en page pour des objets `View` et d'autres objets `ViewGroup`.

Une GUI sera donc une hierarchie d'objets `View` et `ViewGroup` imbriqués.

La GUI d'une application Android est décrite dans un **fichier XML**.

Les notions avancées de la GUI sont abordées notamment dans les documents GUI Android, RecyclerView et Activité n°1 : IHM.

4.6. Les autres composants de base

La spécificité du système Android nécessite certains composants que l'application peut utiliser. Il existe 7 autres types de composants :

- Les `Intents` permettent de communiquer entre différentes activités. Ils sont en quelque sorte le « messager » pour lancer une activité. Ainsi une activité peut en lancer une autre soit en passant un intent vide, soit en y passant des paramètres.
- Les `Intent Filters` jouent le rôle de filtre. Ils permettent de contrôler d'où provient l'Intent (ou d'autres paramètres) afin de lancer ou non l'activité.
- Les `Services` tournent en arrière-plan pour déclencher des `Notifications` et mettre à jour les sources de données et les `Activity`.
- Les `Content Providers` permettent d'accéder à un ensemble de données depuis une application (sources de données partageables). Cela permet par exemple d'accèder aux contacts, à l'agenda, aux photos, … L'application peut définir ses propres `Content Providers` pour permettre à d'autres applications d'utiliser ses données.
- Les `Broadcast Receiver` permettent d'écouter ce qui se passe sur le système Android et de déclencher éventuellement une action. Ce sont des consommateurs de messages diffusés par les `Intents`. Ils sont à l'écoute des intentions envoyées par les autres composants applicatifs et peuvent démarrer automatiquement l'application pour répondre à un message entrant. On les utilise aussi pour lancer un `Service`.
- Les `Widgets`, des variantes des `Broadcast Receivers`, permettent de créer des composants interactifs et dynamiques incorporables dans l'écran d'accueil.
- Les `Notifications` permettent d'envoyer un signal aux utilisateurs sans dérober le focus ni interrompre les `Activity` en cours.

4.7. Déboguage

Il est essentiel de pouvoir déboguer des applications en Java sous Android Stuido.

4.7.1. Afficher des messages de journalisation

Android fournit une classe Log pour afficher des messages de journalisation qui seront visibles dans la console logcat.

Une bonne pratique est de déclarer une constante `TAG` dans chaque classe :

```
public class IHMPrincipal extends AppCompatActivity
{
    /**
     * Constantes
     */
    private static final String TAG = "IHMPrincipal"; //!< le TAG de la classe pour
les logs

    ...
}
```

Il existe plusieurs niveaux de journalisation avec `Log.v()` (*verbose*), `Log.d()` (*debug*), `Log.i()` (*info*), `Log.w()` (*warning*), et `Log.e()` (*error*) :

```
public class IHMPrincipal extends AppCompatActivity
{
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        Log.d(TAG, "onCreate()");

        super.onCreate(savedInstanceState);
        ...

        Log.v(TAG, "index=" + i);
    }
}
```

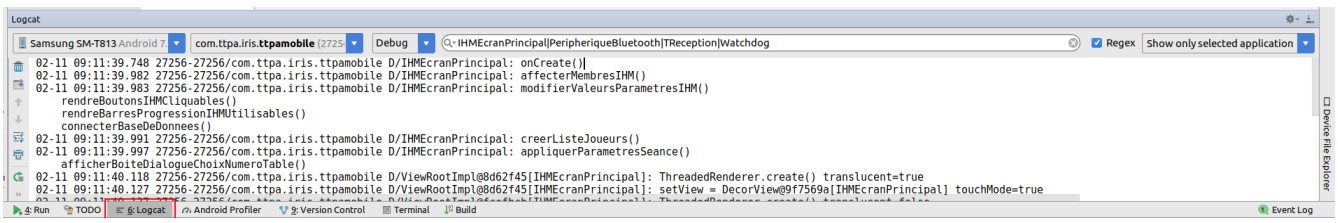
Le message de journalisation aura le format :

```
date time PID-TID/package priority/tag: message
```

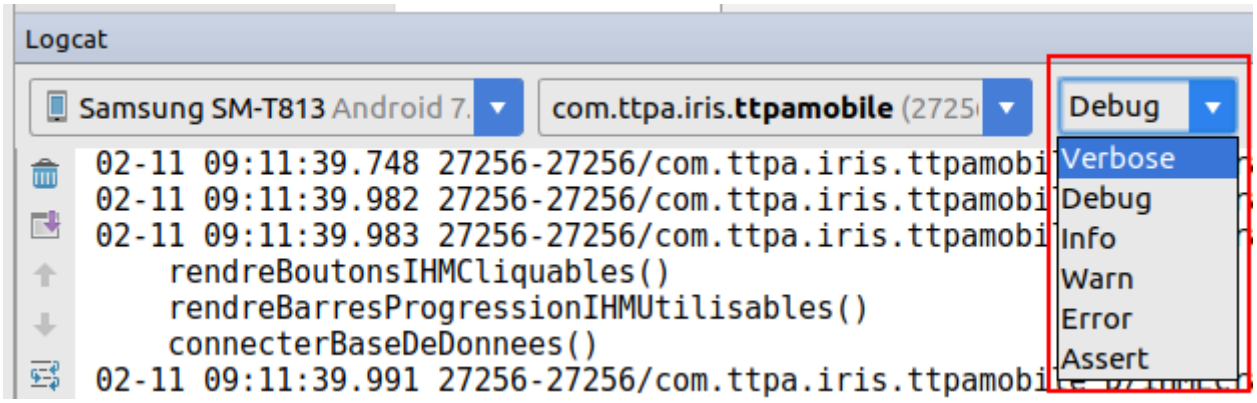
Par exemple :

```
02-11 09:11:39.748 27256-27256/com.ttpa.iris.ttpamobile D/IHMPrincipal: onCreate()
```

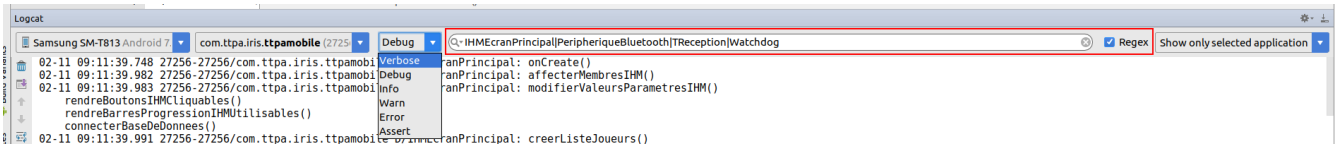
On peut visualiser l'ensemble des messages dans logcat :



On peut filtrer par niveau :

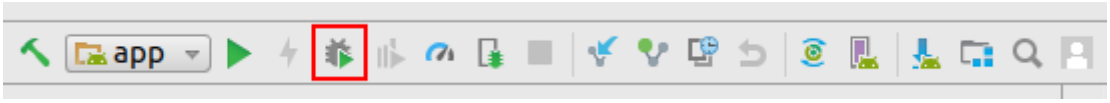


Et en utilisant une expression rationnelle (*regex*) :



4.7.2. Déboguer

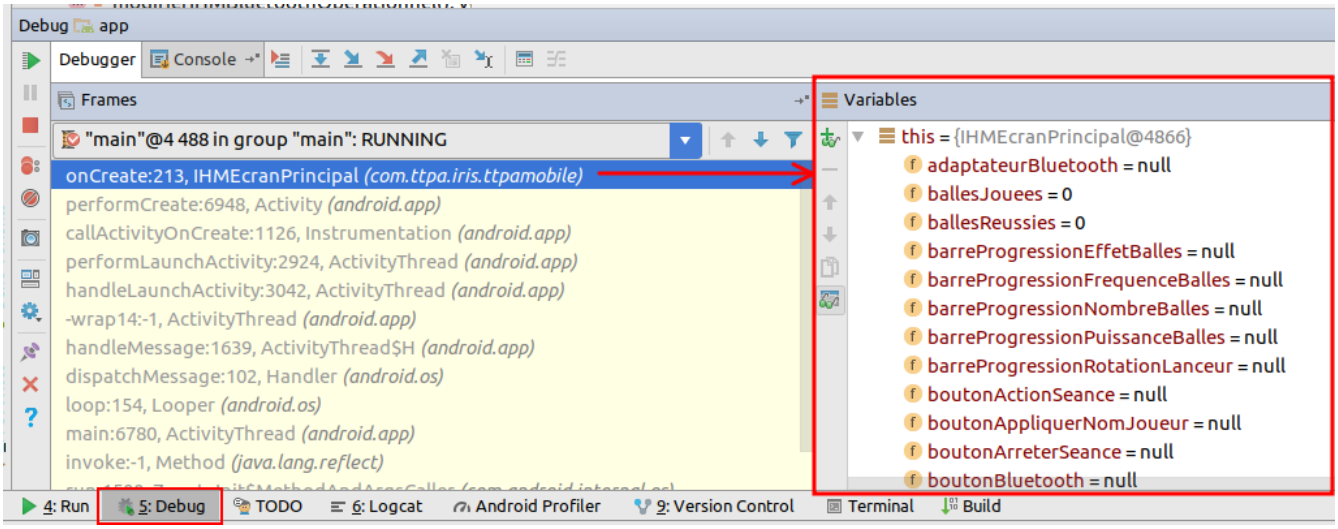
Android Studio propose un mode déboguage :



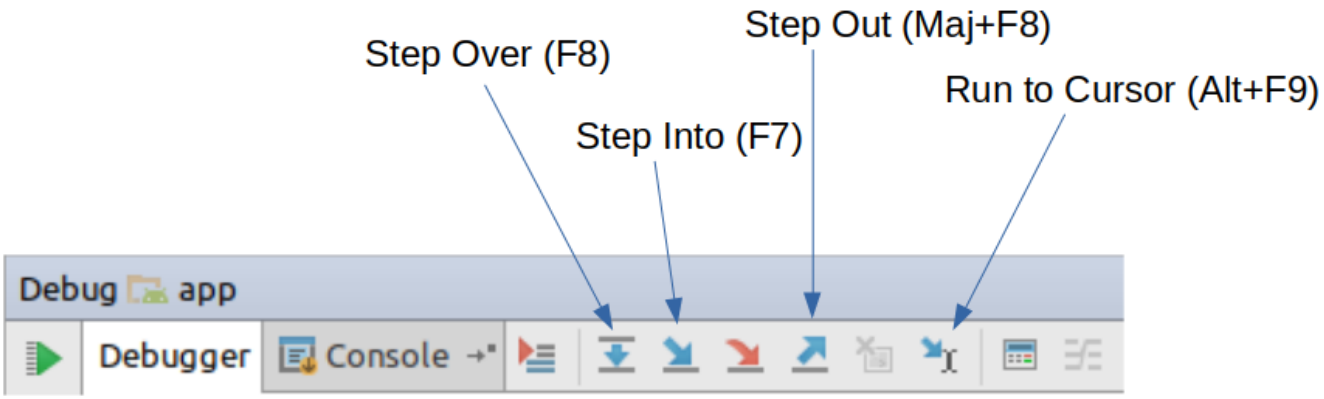
On peut placer des points d'arrêt sur une ligne de code :

```
203     @Override
204     protected void onCreate(Bundle savedInstanceState)
205     {
206         Log.d(TAG,  msg: "onCreate()");
207         super.onCreate(savedInstanceState);
208
209         // Associer le layout
210         setContentView(R.layout.ecran_principal);
211
212         // Affecter des membres de l'IHM
213         affecterMembresIHM();
214
```

Ensuite, on accède à la fenêtre de débogage :



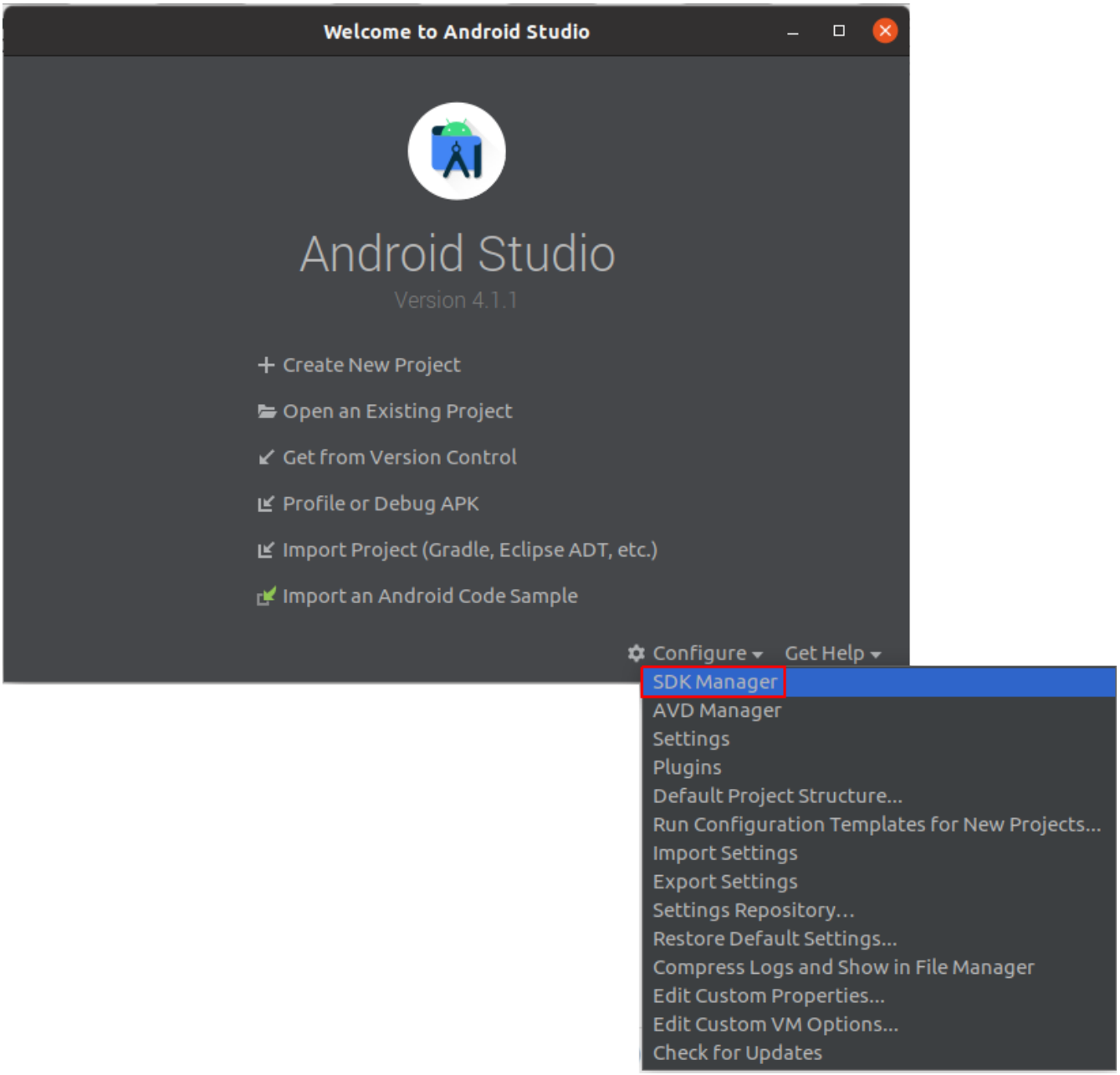
On retrouve les modes habituels de contrôle d'exécution :



5. Annexes

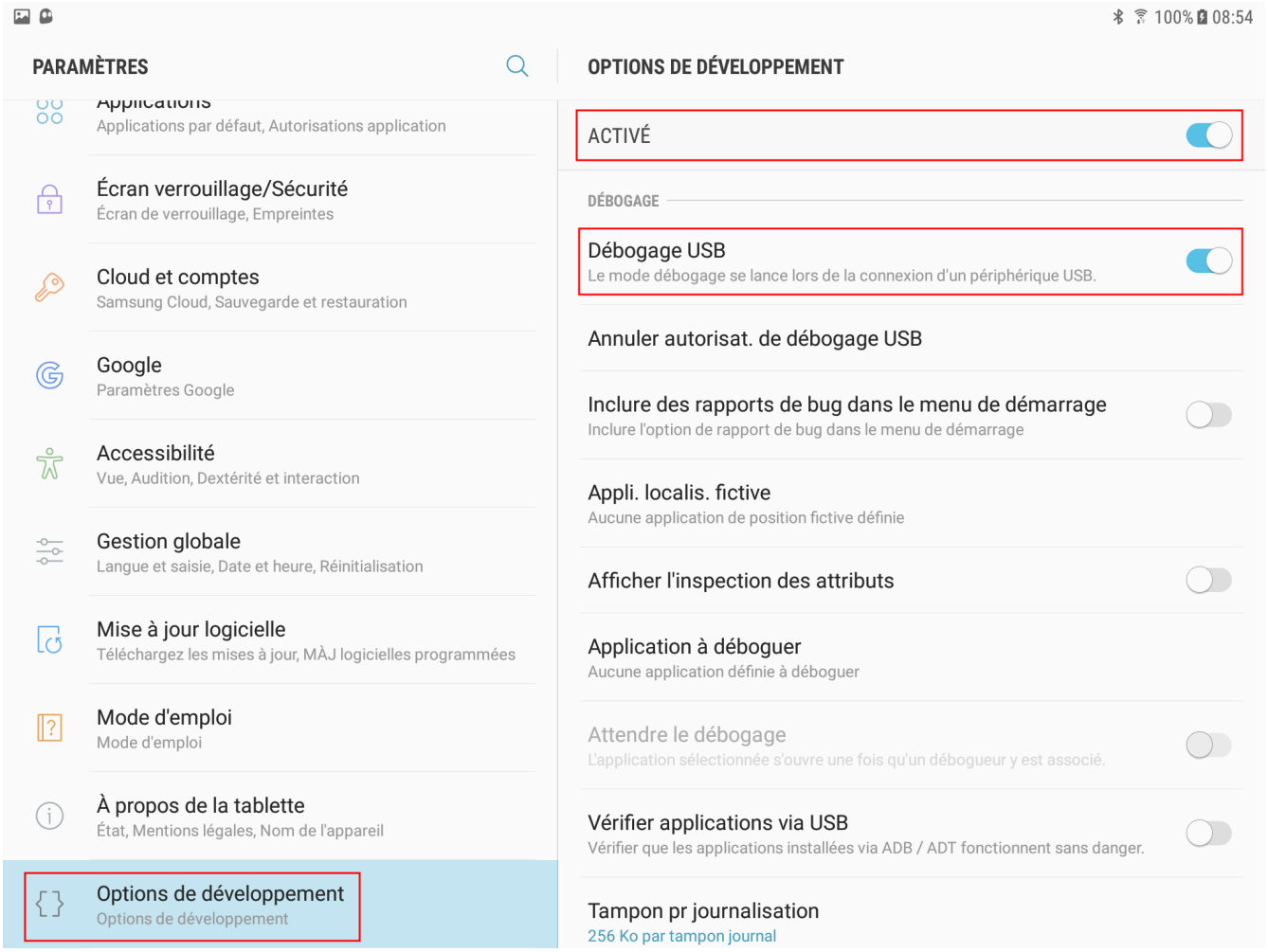
5.1. SDK Manager

Le SDK Android est constitué de paquets modulaires qui peuvent être téléchargés séparément au moyen de l'outil **Android SDK Manager**. Cet outil est particulièrement pratique lorsqu'il est nécessaire de faire une mise à jour liée à une évolution du niveau d'API. Plusieurs niveaux d'API peuvent cohabiter.



5.2. Installation d'un périphérique Android

Il faut préalablement passer le *smartphone* ou la tablette en mode **développeur** (Paramètres → A propos du téléphone → Numéro de build) puis activer le débugage USB (Paramètres → Options pour les développeurs).



Ensuite, il faut relier le *smartphone* ou la tablette sur un port USB de la machine de développement et vérifier qu'il est détecté par le système :

Exemple pour un *smartphone* Acer :

```
$ dmesg
...
[72893.663574] usb 3-9.4.2: New USB device found, idVendor=0502, idProduct=3472
[72893.663581] usb 3-9.4.2: New USB device strings: Mfr=2, Product=3, SerialNumber=4
[72893.663584] usb 3-9.4.2: Product: MT65xx Android Phone
[72893.663587] usb 3-9.4.2: Manufacturer: MediaTek
[72893.663589] usb 3-9.4.2: SerialNumber: AAEEQCBQAYRCRWDQ
[72893.664509] scsi14 : usb-storage 3-9.4.2:1.0
[72894.660646] scsi 14:0:0:0: Direct-Access     Linux    File-CD Gadget   0000 PQ: 0
ANSI: 2
[72894.660945] scsi 14:0:0:1: Direct-Access     Linux    File-CD Gadget   0000 PQ: 0
ANSI: 2
[72894.662178] sd 14:0:0:0: Attached scsi generic sg10 type 0
[72894.662552] sd 14:0:0:1: Attached scsi generic sg11 type 0
[72894.663606] sd 14:0:0:0: [sdj] Attached SCSI removable disk
[72894.663818] sd 14:0:0:1: [sdk] Attached SCSI removable disk

$ lsusb
...
Bus 003 Device 016: ID 0502:3472 Acer, Inc.
```

Sous Linux, il est probablement nécessaire de créer un fichier de règles *udev* qui contient une configuration USB pour chaque type de périphérique réel. En tant que **root**, créer le fchier `/etc/udev/rules.d/51-android.rules` et y inscrire la ligne suivante (en précisant en hexadécimal votre *idVendor*) :

```
$ sudo vim /etc/udev/rules.d/51-android.rules
SUBSYSTEM=="usb", ATTR{idVendor}=="0502", MODE="0666"
```

Brancher le téléphone et vérifer qu'il est reconnu :

```
$ ~/Android/Sdk/platform-tools/adb devices
List of devices attached
AAEEQCBQAYRCRWDQ    device
```

Exemple pour une tablette Samsung :

```
$ dmesg
...
[603597.410078] usb 1-5: new high-speed USB device number 51 using xhci_hcd
[603597.559403] usb 1-5: New USB device found, idVendor=04e8, idProduct=6860
[603597.559409] usb 1-5: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[603597.559413] usb 1-5: Product: SAMSUNG_Android
[603597.559415] usb 1-5: Manufacturer: SAMSUNG
[603597.559418] usb 1-5: SerialNumber: d95f698401ba0ee8
[603597.561135] cdc_acm 1-5:1.1: ttyACM0: USB ACM device

$ lsusb
...
Bus 001 Device 051: ID 04e8:6860 Samsung Electronics Co., Ltd Galaxy (MTP)
```

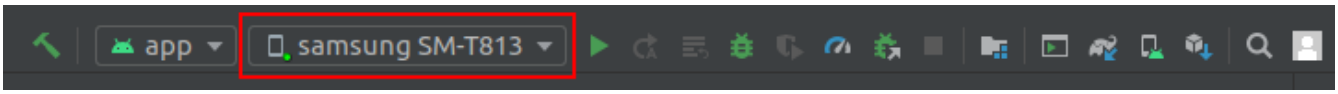
Sous Linux, il est probablement nécessaire de créer un fichier de règles *udev* qui contient une configuration USB pour chaque type de périphérique réel. En tant que **root**, créer le fchier `/etc/udev/rules.d/51-android.rules` et y inscrire la ligne suivante (en précisant en hexadécimal votre *idVendor*) :

```
$ sudo vim /etc/udev/rules.d/51-android.rules
SUBSYSTEM=="usb", ATTR{idVendor}=="04e8", MODE="0666"
```

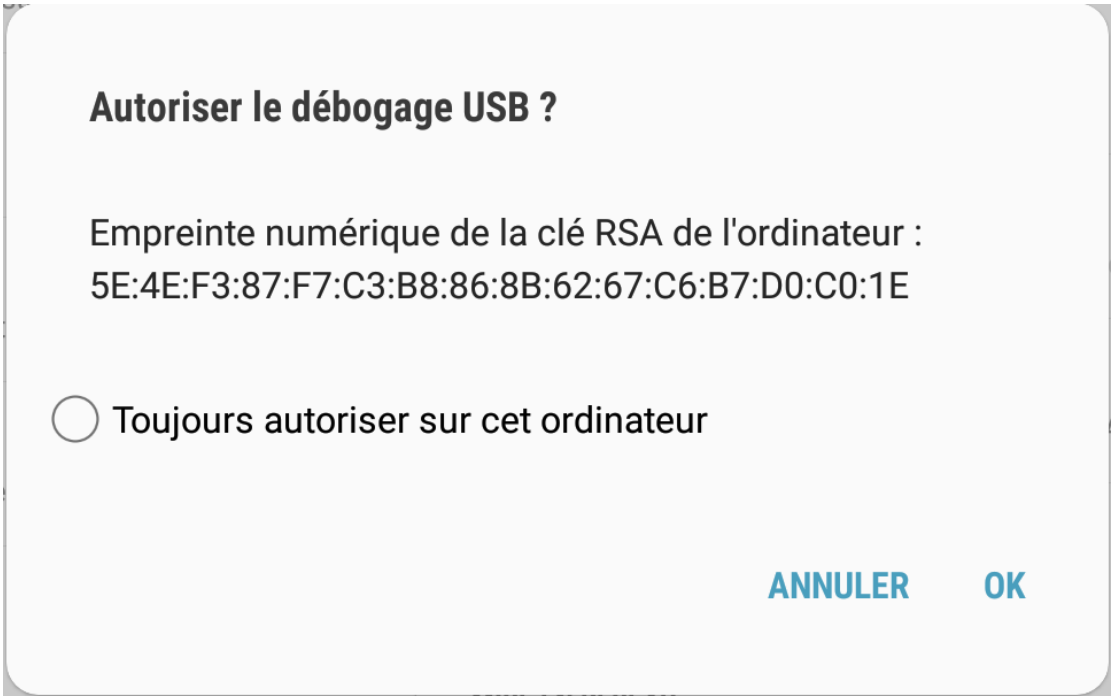
Brancher la tablette et vérifer qu'il est reconnu :

```
$ ~/Android/Sdk/platform-tools/adb devices
* daemon not running; starting now at tcp:5037
* daemon started successfully
List of devices attached
d95f698401ba0ee8    device
```

L'appareil apparaîtra alors dans Android Studio :



La première fois que l'appareil sera détecté, une autorisation de débogage USB sera demandée :



6. Documentations

Liens de référence :

- Introduction à Android Studio
- Guide de référence d'Android SDK

Et aussi :

- Cours et tutoriels pour Android
- FAQ Android

7. Ressources

Pour le développement d'applications Android : http://tvaira.free.fr/dev/android/index.html

Quelques activités de base :

- Installation d'Android Studio et Installation d'Android Studio 4.1
- Activité n°1 : IHM
- Activité n°2 : Communication réseau
- Activité n°3 : Base de données SQLite
- Activité n°4 : Bluetooth
- Activité n°7 : Base de données MySQL

- Activité n°8 : MQTT

Et aussi :

- GUI
- RecyclerView
- Débogage
- Communication UDP
- Requête HTTP
- JSON
- XML
- Capteurs
- NFC
- Camera
- Base de données
- Tâches d'arrière-plan
- Géolocalisation
- Cartographie
- Dessin et Graphique
- SMS
- Firebase et Android

Pour les applications hybrides :

- Activité n°5 : jQuery Mobile
- TP jQuery Mobile
- Activité n°6 : Cordova et Installation et tests d'Apache Cordova
- Application hybride

Pour finir :

Qt pour Android

Site : tvaira.free.fr

Thierry Vaira - tvaira@free.fr - version v1.0 - 05/01/2021