THE INNOVATIVE UNIVERSITY	Cours : Développement Mobile	**Année universitaire :** 2025/2026

TP1 : Introduction et prise en main de l'environnement de développement

Objectif :

Android est la première plate-forme mobile complète, libre et entièrement paramétrable. Android est un système d'exploitation basé sur un noyau Linux pour Smartphones, PDA et terminaux mobiles de nouvelle génération. Il a été créé afin de permettre aux développeurs de bénéficier au maximum de tout ce que peut offrir un appareil mobile. Les développeurs Android bénéficient d'un kit de développement complet, avec des outils suffisamment puissants pour le développement des applications riche en fonctionnalités, ils ont aussi de nombreuses options pour la distribution et la commercialisation de leurs applications.

Plus d'une cinquantaine de compagnies ont contribué au développement Android à travers **Open Handset Alliance** comprenant des sociétés comme google, htc, Intel, Motorola, LG et Samsung, ainsi que les opérateurs T-Mobile, Telecom Italia, NTT, DoCoMo, Qualcomm et China Mobile. Le premier téléphone Android (le T-Mobile G1) a été commercialisé en fin 2008. En avril 2009, la version 1.5 d'Android sort. Cette version baptisée Cupcake (petit gâteau) inaugure les nouveaux noms des versions d'Android. Ce qui donnera pour les futures versions :

- Donut (Beignet) 1.6 (API 4) : septembre 2009
- Eclair (Eclair) 2.0/2.1 (API 7) : octobre 2009
- Froyo (Yaourt glacé) 2.2.x (API 8) : mai 2010
- Gingerbread (Pain d'épice) 2.3.x (API 10) : décembre 2010
- Honeycomb (Ruche d'abeille) 3.x (API 11 - 12 - 13) : février 2011
- Ice Cream Sandwich (Sandwich à la crème glacée) 4.0.x (API 14 - 15) : octobre 2011. Cette version a unifié la version tablette (Honeycomb) et la version smartphone (Gingerbread).
- Jelly Bean (Dragibus) 4.1.x (API 16 et +) : juin 2012.
- KitKat (4.4–4.4.4, 4.4W–4.4W.2) : décembre 2013
- Lollipop (5.0–5.1.1) : janvier 2015
- Marshmallow (6.0–6.0.1) : Octobre 2015
- Nougat (7.0–7.1.1) : Août 2016
- Oreo (8.0- 8.1) : Août 2017
- Pie (9.0) : Août 2018
- Android 10 : 13 mars 2019
- Android 11 : 19 février 2020
- Android 12 : 4 octobre 2021
- Android 13 : 15 août 2022

Développement facile et bon marché : Contrairement aux autres plates-formes mobiles, il n'y a pratiquement aucun cout pour le développement d'une application Android. L'environnement de développement (IDE) Android Studio est devenu le plus populaire (l'environnement officielle) des IDE pour le développement des applications
Android avec un plug-in très puissant. Maintenant, pourquoi le développement Android est facile : Parce que les applications Android sont écrites en **Java**.
Vous pouvez récupérer les sources du SDK Android, obtenir la documentation et voir les forums de développeurs Android sur les sites :

- http://developer.android.com
- http://source.android.com

Architecture du système Android :

Le choix de l'architecture Android est basé sur l'idée de maitriser les ressources et la consommation. Les applications Android s'exécutent sur un système contraint (vitesse de processeur, mémoire disponible, consommation de batterie, différence d'affichage). Vous devez en tant que développeur faire attention à plusieurs points :

- La création de nouveaux objets.
- L'utilisation des ressources (processeur, RAM, stockage, etc.).
- La consommation de la batterie.
- La diversité des tailles des résolutions des écrans et des configurations matérielles.

L'architecture d'Android se compose de cinq parties distinctes (voir Figure 1) :

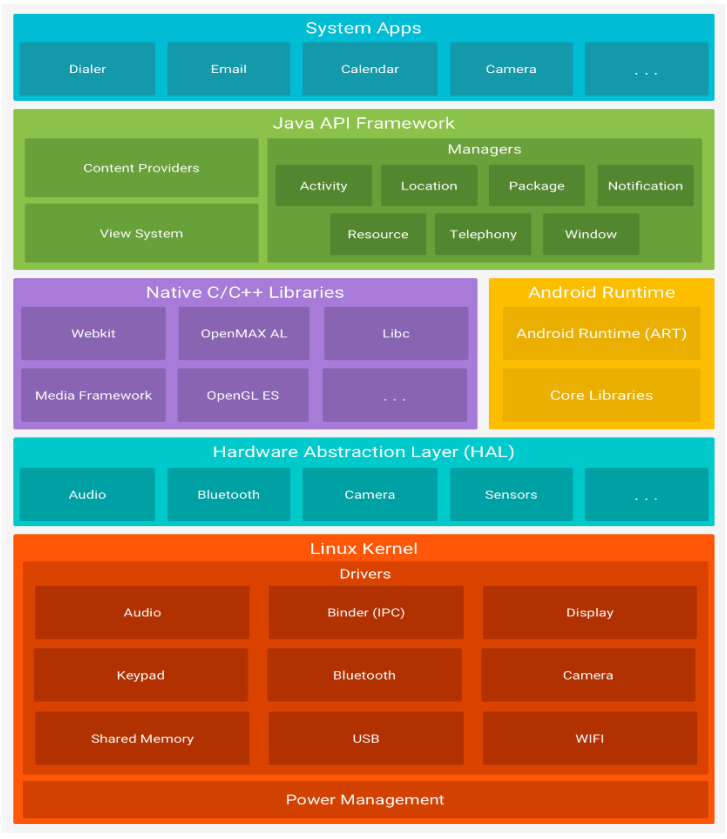

Figure 1 : Architecture d'Android

- Linux Kernel : Android repose sur le noyau Linux qui se charge des services de base tels que la sécurité, gestion de la mémoire, la gestion des processus, de l'énergie, la couche réseau, ou encore des pilotes matériels.
- Hardware Abstraction : Une couche d'abstraction entre le Hardware et le reste de pile logiciel : Propose des interfaces standards qui exposent les capacités matérielles du périphérique au APIs de niveau supérieur.
- Android Runtime : un ensemble de librairies de base du langage Java utile à la conception des applications, et une machine virtuelle ART (Android Runtime remplaçant DALVIK depuis la versions 4.4) qui est le cœur même du système. Elle est chargée d'exécuter toutes les applications du système (y compris les applications de base), ainsi que les librairies de base du runtime. Cette machine virtuelle n'exécute pas des fichiers *.class comme le ferait une JVM, mais des fichiers *.dex qui se trouve être en fait des classes Java transformées spécialement pour Dalvik/ART via l'outil 'dx' de son SDK afin d'optimiser la mémoire utilisée. Cette machine virtuelle possède plusieurs spécifications :
    - L'utilisation de registres et non de piles.
    - 30 % d'instructions en moins qu'une JVM (Java Virtual Machine) classique.
    - Un temps d'exécution deux fois plus rapide qu'une JVM classique.
    - Les applications sont par défaut isolées les unes par rapport aux autres (une application = un processus). Ce qui permet d'empêcher un effet domino lors du crash d'une application.
- Native C/C++ Librairies : regroupent un ensemble de librairies natives couvrant un scope assez large allant des librairies standard C/C++ aux librairies graphiques 2D/3D, en passant par le support multimédia des formats image/audio/vidéo les plus courant, base de données embarquée ou encore un WebKit.
- Java API Framework : représenté dans l'API par les packages android.*. Elle se charge à la fois de proposer un cadre pour le développement d'application, mais également de fournir l'accès aux fonctionnalités avancées de l'appareil en se basant sur les librairies natives de la couche inférieure. C'est un ensemble de services contenant :
    - Un jeu riche et extensible de vues (views), par exemple des listes, des grilles, des zones de texte, des boutons, et même un navigateur web embarqué, qui peuvent être utilisés pour construire une application,
    - Content Providers permettent aux applications d'accéder aux données des autres applications (telles que les Contacts), ou de partager leurs propres données
    - Resource Manager est un gestionnaire de ressources
    - Notification Manager permet à toutes les applications d'afficher des alertes personnalisées dans la barre d'état
    - Activity Manager gère le cycle de vie des applications
- System App : cette couche regroupe toutes les applications de l'appareil mobile, que ce soit les applications de base ou des applications tierces. On trouve par exemple un client de messagerie, SMS, calendrier, cartes, navigateur, contacts.

Installation de l'environnement de développement (Android Studio)

- Téléchargez le dernier JDK (Java Development Kit) que vous pouvez trouver sur le site d'Oracle: https://www.oracle.com/java/technologies/javase-downloads.html
- Téléchargez Android Studio. Avant, on pouvait utiliser Eclipse, mais son plugin Android n'est pas très développé. Android Studio est l'IDE officiel pour le développement d'applications Android. Il contient l'environnement de développement, le SDK (Software Development Kit) Android avec la dernière version de la plateforme, ainsi qu'un émulateur. Consulter l'Adresse : https://developer.android.com/studio/index.html
- Lancez l'exécutable pour démarrer l'installation et suivez les instructions à l'écran.

NB : Si le répertoire Java n'est pas détecté automatiquement, il faudrait définir une variable d'environnement JAVA_HOME qui indique le répertoire où vous avez installé le JDK (ex : C:\Program Files\Java\jdk1.8.0_281).

Découverte de Android Studio

On va commencer par le principal outil de travail : Il faut démarrer Android Studio pour créer et configurer des applications (Figure 2).

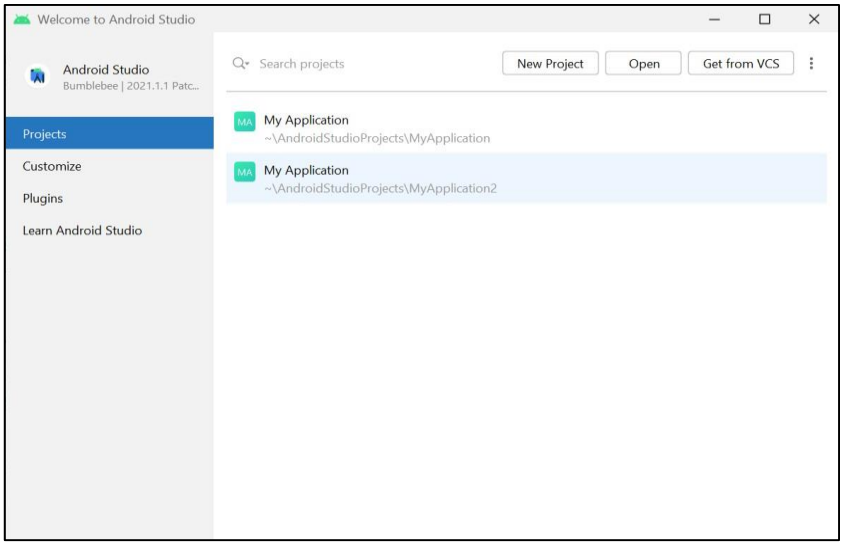

Figure 2 : Page de Démarrage d'Android Studio

Il faut s'assurer que nous possédons tout ce qu'il faut pour développer. En premier temps le "Software developpement Kit" ou kit de développement : C'est un outil utilisé par les programmeurs pour réaliser et tester des applications. Dans la page de démarrage, sélectionnez Configure (les deux points sur la droite) > SDK Manager. Dans le gestionnaire (Figure 3) vous verrez la version du SDK installé (avec les mises jour disponibles) et aussi la version de l'API (Application Programming Interface) installée et la version du OS pour laquelle elle vous permettra de développer. Installez les éventuelles mises à jour. Choisir celles qui correspondent aux types de téléphones qu'on vise, mais tout n'est pas à installer : il faut cocher Show Package Details, puis choisir élément par élément. Seuls ceux-là sont indispensables :

- Android SDK Platform
- Intel x86 Atom_64 System Image

Le reste est facultatif (Google APIs, sources, exemples et docs).
Le gestionnaire installe environ 1 Go de fichiers :

- SDK Tools : indispensable, il contient les outils de gestion du SDK et des AVD (emulator.exe).
- SDK Platform tools : indispensable, contient quelques programmes utiles comme adb.exe (adb sous Linux). Ces programmes sont des commandes qui sont normalement disponibles dans une fenêtre appelée Terminal en bas d'Android Studio. En principe, vous devriez pouvoir taper adb dans ce shell et ça vous affiche l'aide de cette commande. On va s'en servir un peu plus bas.
- SDK Platform : indispensable, contient les librairies.
- System images : pour créer des AVD : contient les images de la mémoire de tablettes et de smartphones virtuelles. Ce sont des images de disques virtuels à lancer avec une machine virtuelle tel que qemu.
- Android Support : Propose divers outils pour créer des applications tel que : des Éléments supplémentaires de mise en page et d'interface utilisateur (tel que RecyclerView : un composant utilisé pour créer les interfaces des applications Android) ou des bibliothèques de Prise en charge de différents facteurs de forme d'appareils.
- Exemples et sources.

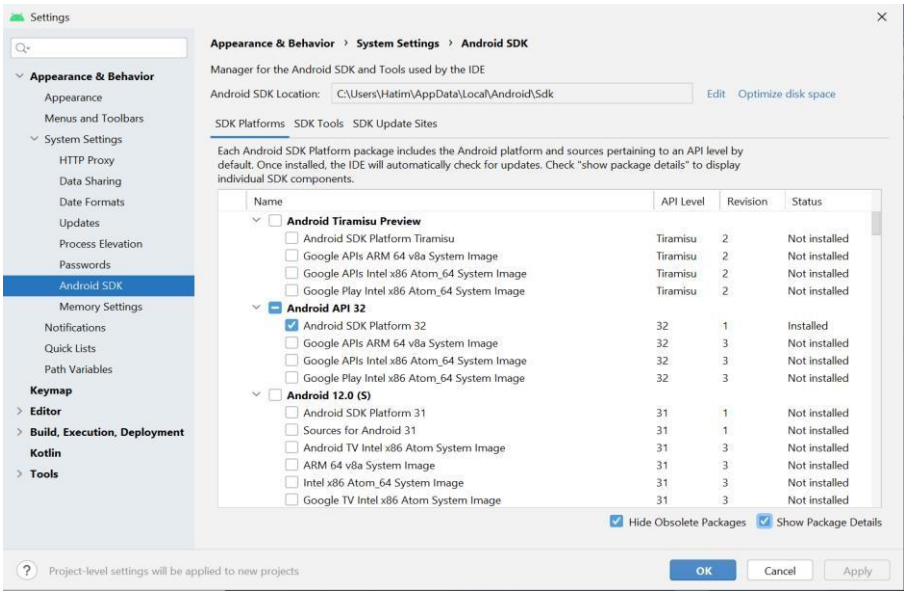

Figure 3 : Configuration du SDK

Afin de tester votre application, Le SDK Android permet de :

- Installer l'application sur une vraie tablette ou un smartphone connecté par USB
- Simuler l'application sur un smartphone ou une tablette virtuelle à l'aide d'un émulateur.

Un émulateur permet de reproduire le comportement d'un appareil réel d'une façon virtuelle. L'utilisation d'un émulateur nous évite d'avoir à charger à chaque fois l'application dans un appareil pour la tester. On pourra ainsi lancer l'application dans l'IDE et elle s'exécutera sur un appareil virtuel appelé Android Virtual Device AVD qui émule le comportement d'un téléphone, une tablette ou autre. Dans la page de démarrage, sélectionnez Configure > AVD Manager.
La fenêtre qui s'ouvre (Figure 4) permet de construire un AVD spécifique avec le bouton **Create Virtual Device.**

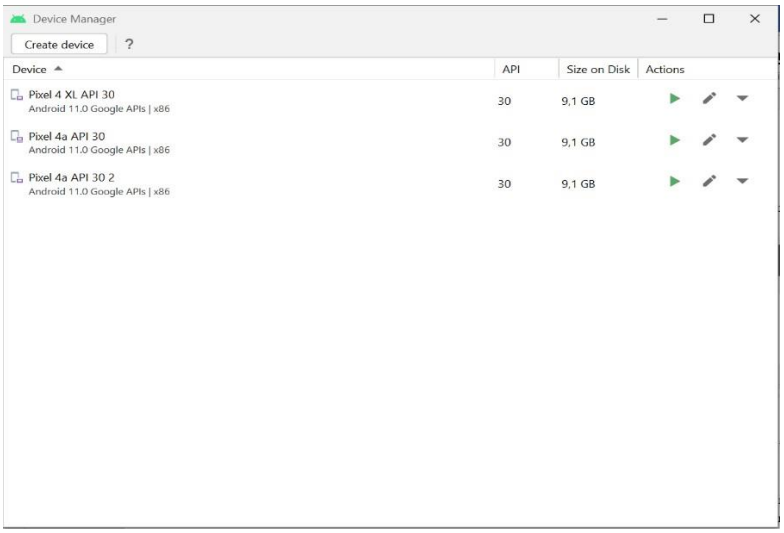

Figure 4: Gestionnaire d'AVD

La Figure 5 correspond aux valeurs utilisées pour la création de cette AVD. Pour configurer un émulateur, vous avez beaucoup de choix possibles.

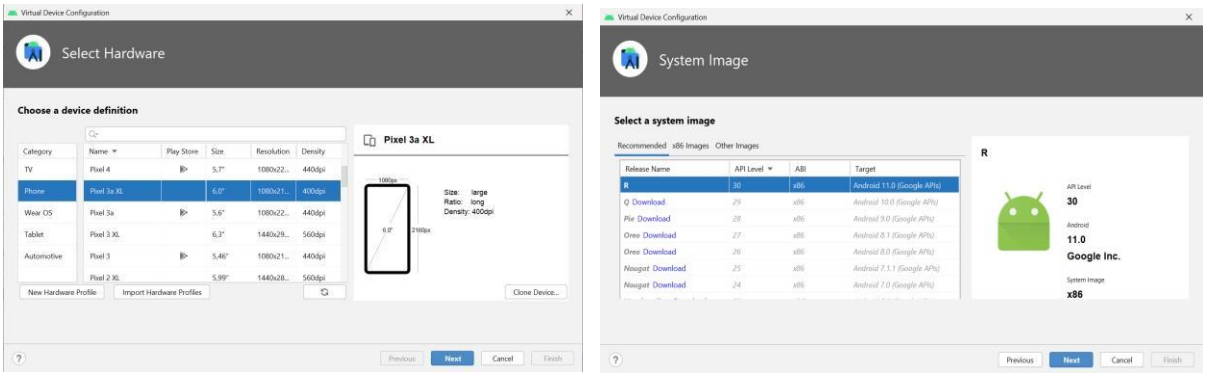

Figure 5 : Création d'un AVD

Le principe est de ne pas créer un smartphone virtuel trop riche. Si le smartphone virtuel est trop puissant, trop gros (RAM et taille d'écran), il va occuper énormément de place en mémoire et sur le disque.

Un autre point important est le CPU du smartphone virtuel. Les vrais smartphones fonctionnent en général avec un processeur ARM. Ça désigne un modèle de programmation (registres et instructions) spécifiques qui doit être simulé sur votre ordinateur. Le mieux est de choisir un AVD avec un processeur de type amd64 qui ne devra pas être simulé, mais simplement lié, comme avec Docker ou VirtualBox.

Une solution prudente consiste donc à choisir l'AVD le moins rutilant. Par exemple, choisissez le type Phone, modèle 5.1 480x800 en mdpi ; puis dans l'écran suivant, changez d'onglet pour x86 images et choisissez une des API installées, la 27 Android 8.1 par exemple, puis dans le dernier écran, activez Show Advanced Settings et changez la taille de la RAM : au lieu de 343 Mo, mettez 512 Mo, et décochez Enable Device Frame. Validez la création de ce smartphone virtuel.

La création d'un AVD prend du temps. Il faut générer l'image de la mémoire virtuelle, entre 1 et 4 Go. L'AVD apparaît dans la liste et vous pouvez le lancer en cliquant sur le triangle vert à droite. Notez qu'elle met également beaucoup de temps à devenir disponible. Pour ne pas perdre de temps

plus tard, vous devrez faire attention à ne pas la fermer.

Première application « Bienvenue l'UIR ! »

On va commencer par créer un nouveau projet, qu'on appellera Bonjour.

1. Lancer Android Studio, puis Create New Project. Cette étape vous permet de sélectionner l'activité principale de votre application.
   Définition : Une activité est une composante essentielle de la plateforme Android. Chaque activité représente une tâche de l'application, souvent lié à une interface utilisateur de l'application. Ici, l'application Bonjour a une seule activité, appelée main, qui a une tâche unique : afficher sur l'écran "Bienvenue dans le monde Android !".
   Pour débuter nous allons choisir « Empty Activity », c'est une activité vierge. Cliquer sur Next.

2. Une fenêtre s'ouvre, choisissez un nom pour votre application (Application name = Bonjour), puis un « Company domain », vous devez le former comme une URL, voici un exemple : monnom.monentreprise.com, cela formera avec le nom de votre application, le nom du package de votre application (Package name). Après, l'emplacement du dossier à créer pour le projet. Ensuite, Le langage de programmation que vous voulez utiliser (On va choisir JAVA) et enfin vous pourrez choisir la version minimum nécessaire au terminal utilisant votre application (Minimum SDK) (vous pouvez cliquer sur le bouton d'aide pour savoir les détails sur les différentes distributions). Et Cliquer sur Finish.

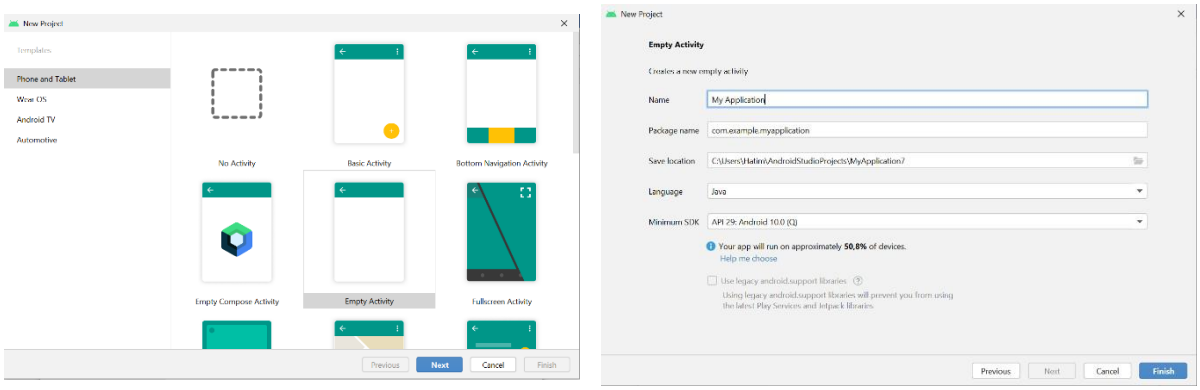

Figure 6 : création d'un nouveau Projet

Découverte de la fenêtre de travail :

L'interface d'Android Studio est un peu déroutante au début. Vous verrez rapidement qu'on ne se sert que d'une toute petite partie, vite apprise. Il y a des boutons tout autour de la fenêtre : Project, Structure, Captures, Favorites, Build Variants, Terminal, Android Monitor, Messages, TODO, etc. Ce sont des boutons à bascule, exclusifs entre eux sur un même côté, qui affichent des onglets ; celui qui est actif est grisé. Par exemple dans l'image ci-dessous, c'est l'onglet Project qui est ouvert. La structure du projet est visible à gauche, comme dans Eclipse. Les fichiers sont arrangés de manière pratique : manifeste, sources java et ressources, grâce au choix déroulant Android (figure 7).

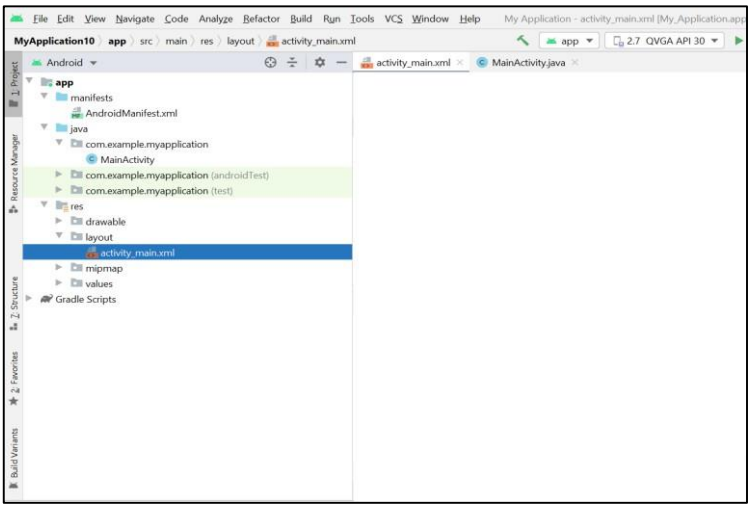

Figure 7: Inventaire du projet

Prenez quelques instants pour déplier les dossiers manifests, java et res et regarder ce qu'il y a dedans (ça a été décrit en cours).

Structure du projet

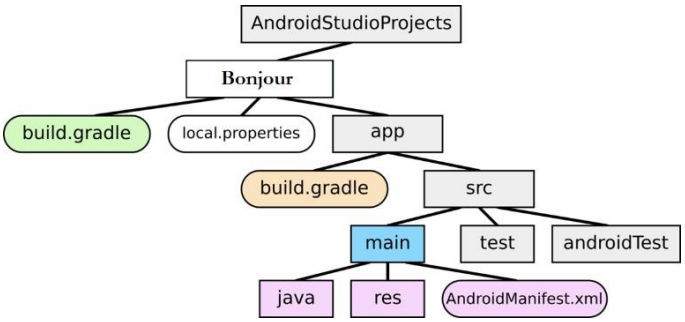

Figure 8: Structure des fichiers d'un projet Android

- Le dossier du projet contient le premier build.gradle, celui en vert qui indique les dépôts à utiliser pour la compilation. Le fichier local.properties contient le chemin du SDK.
- Le dossier app contient le second build.gradle, celui en orange qui décrit les caractéristiques du projet : API et dépendances.
- Le dossier app/src/main en bleu contient le cœur du projet : manifeste, sources et ressources.
- app/src/main/java contient les dossiers correspondant au package que vous avez déclaré, avec les sources Java tout au bout du package.
- app/src/main/res/drawable contiendra des images vectorielles utilisées dans votre projet.
- app/src/main/res/layout contient les dispositions d'écran des activités de votre projet.
- app/src/main/res/menu contient les menus contextuels et d'application du projet.
- app/src/main/res/mipmap* contient les icônes bitmap dans diverses résolutions.
- app/src/main/res/values contient les constantes du projet, dont les chaînes de caractères avec toutes leurs traductions.

Tester votre application :

Le lancement de l'application est simple. Il suffit de cliquer sur le bouton triangulaire vert dans la barre d'outils.

Figure 9: Bouton de lancement

Il faut s'assurer qu'un AVD en cours d'utilisation sinon il faut le créer. Il en faut un qui soit au moins du niveau d'API de votre application.

Affichage d'un message :

Ouvrez le source MainActivity.java et trouvez la méthode onCreate. Cela doit ressembler au source suivant. Complétez-le avec les instructions qui permettent de faire afficher un message dans la fenêtre LogCat.

```
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.util.Log;

public class MainActivity extends AppCompatActivity {

    private static final String TAG = "Test" ;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.i(TAG, "Salut !");
    }
}
```

Ouvrir l'onglet LogCat en bas de de l'IDE.

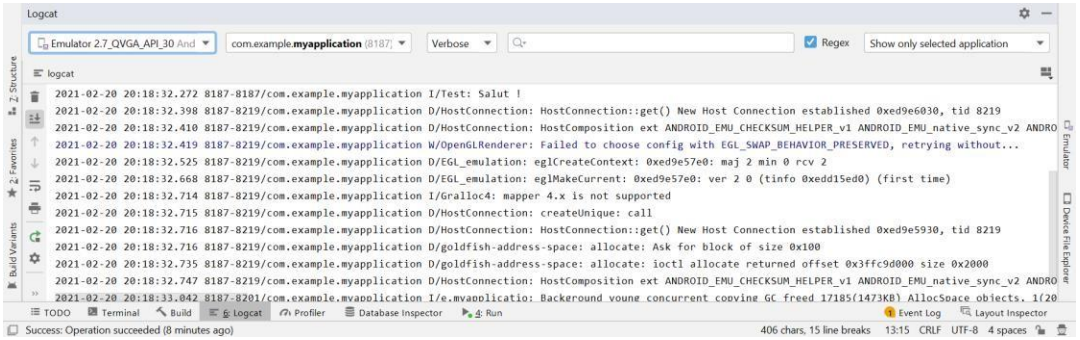

Figure 10: LogCat

Cette fenêtre permet de voir les messages émis par l'émulateur. Le bouton en forme de poubelle vide ces messages. Vous pouvez changer le niveau d'affichage : verbose, debug, info. . . ou créer un filtre basé sur le TAG pour ne voir que les messages utiles.

Lancez l'exécution du projet. Vous devriez voir une ligne similaire à celle-ci :

```
2021-02-20 20:18:32.272 8187-8187/com.example.myapplication I/Test: Salut !
```

Mise au point : débogueur

Nous allons expérimenter quelques aspects du débogueur intégré dans Studio. On va être un peu limités par la taille de notre programme. On va seulement pouvoir découvrir quelques techniques très utiles : points d'arrêt et exécution pas à pas.

Lancement en mode debug

Normalement, il suffit de cliquer sur le bouton encadré par un cercle dans la figure 9 pour lancer la mise au point.
Mais s'il y a une erreur de connexion (message Error running app: Unable to open debugger port dans la barre d'état tout en bas), il faut d'abord « attacher » Studio à l'AVD. Il faut cliquer sur le bouton Attach debugger to Android process entouré par un carré (figure 9). Vous serez peut-être obligé(e) de relancer l'AVD.
La figure 11 présente les boutons utiles pour mettre le programme au point. Le bouton, entouré d'un carré en bas, affiche les points d'arrêt. Nous n'emploierons pas les autres boutons.
Le bouton vert en haut permet de faire repartir le programme jusqu'au prochain point d'arrêt. Le bouton, entouré d'un carré, permet d'exécuter la prochaine ligne sans rentrer dans les méthodes imbriquées s'il y en a. Le bouton, entouré d'un cercle, entre dans les appels imbriqués de méthodes. Le bouton entouré d'un triangle, exécute à pleine vitesse jusqu'à la sortie de la fonction puis s'arrête.

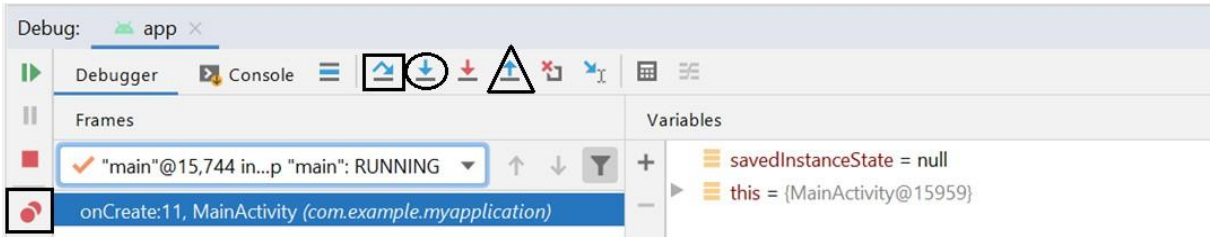

Figure 11: Bouton d'exécution pas à pas

La mise au point repose sur ces deux techniques :

- Lancer en mode normal et attendre que ça se plante. L'exécution s'arrête alors exactement sur l'instruction qui est mauvaise, par exemple celle qui déclenche une NullPointerException.
- Mettre un point d'arrêt avant l'endroit qu'on pense mauvais puis exécuter ligne par ligne pour voir ce qu'il y a dans les variables et guetter l'erreur.

Placement d'un point d'arrêt

Cliquez dans la marge gauche au début de la méthode onCreate(). Ça place un gros point rouge dans la marge.

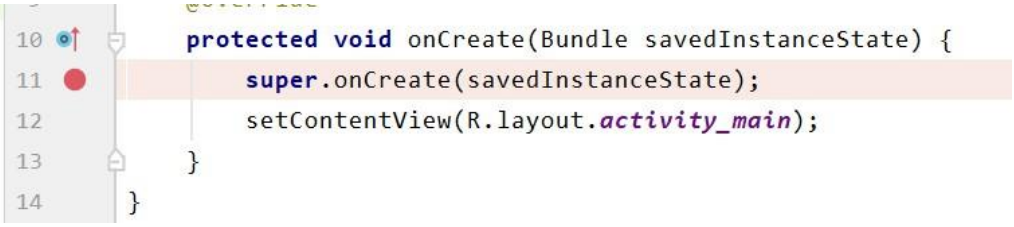

Figure 12: Point d'arrêt dans le source

La fenêtre Breakpoints, qu'on obtient en cliquant sur le bouton entouré en bleu foncé dans la figure précédente, liste tous les points d'arrêts que vous avez placé. N'en mettez pas trop dans un programme, vous aurez du mal à suivre tout ce qui peut se passer. Mettez vos méthodes au point les unes après les autres, pas tout en même temps.

Android Debug Bridge : ADB

L'ADB est un outil pour nous aider à communiquer avec une tablette ou smartphone virtuelle. Vous devez configurer votre variable d'environnement Path sous Windows en ajoutant : C:\Users\User\AppData\Local\Android\Sdk\platform-tools

Redémarrer votre IDE et ouvrer après la fenêtre Terminal en bas d'Android Studio, puis essayez :

- adb devices Affiche la liste des smartphones connectés : les réels par USB et les virtuels actifs.
- adb shell. Dans cette connexion à un smartphone, tapez :
  - ls : c'est une arborescence Unix, mais on n'a pas beaucoup de droits, on n'est pas root : sinon il y a des erreurs dues aux protections.
  - cd /mnt/sdcard : c'est l'un des rares dossiers où on peut aller.
  - mkdir essais : crée un dossier sur la carte sd virtuelle (vérifiez la création avec ls).
  - exit pour sortir du shell et revenir dans Windows.
  - adb push : copie une image sur un smartphone, dans le dossier correspondant au stockage interne sdcard : L'image sera visible dans l'application Google Photos. adb push D:\images\img.jpg /sdcard/Pictures

Il y a une vraie ressemblance entre l'adb avec des outils comme telnet, ssh, ftp.