

# Cours développement d'application mobile native



Hend Ben Ayed

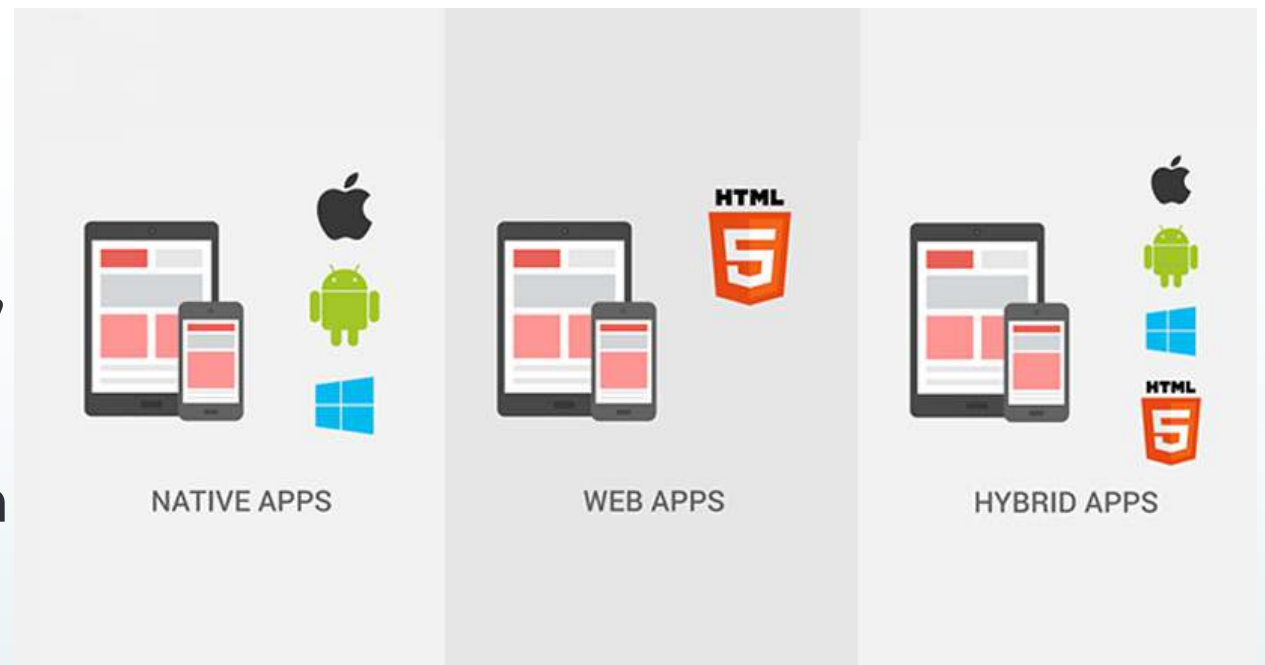
kharrat

2023-2024

L3DSI

## Chapitre 1: Introduction à l'Android

## Application native, hybride, cross-platform ou web, comment faire son choix ?



### ▪ Les applications natives

cela dépend du système d'exploitation qu'ils utilisent.

Société	Appareils	Système d'exploitation	Plateforme de téléchargement	Langages de programmation
Apple	iPhone / iPad	iOS	App Store	Objective C / Swift
Google	Samsung / Google Phone et tablette / Motorola / ...	Android	Google Play	Java
Microsoft	Windows Phone	Windows Phone	Windows Store	C#

## Web-application

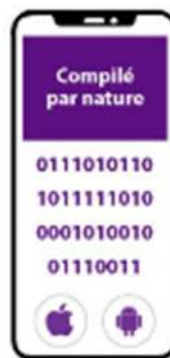
- Les Web-applications sont développées dans un langage Web interprété par le client (le navigateur de votre mobile).
- L'application est développée en HTML, CSS et JavaScript. La plupart du temps, l'utilisateur devra avoir une connexion Internet.

## Application hybride

- L'application hybride est un mix des deux premières solutions. Le développeur va passer par un ensemble de bibliothèques de code pour permettre d'intégrer du HTML, du CSS et du JavaScript dans une application native. Elle est donc téléchargeable et utilisable hors ligne.
- Le développement d'une application hybride repose sur l'utilisation de solutions comme PhoneGap ou Cordova.
- L'application, grâce à ces Frameworks (bibliothèque de code), bénéficie des mêmes fonctionnalités qu'une application native.
- Les deux dernières grandes tendances en matière de Framework pour ce style de développement sont : Ionic et ReactNative (solution facebook).

## Application cross-platform ?

- Contrairement aux appli hybrides dont le code s'exécute dans une « webview » native, les applications cross-platform produisent **un même code source qui, compilé, produit deux applications natives**. Cela induit une meilleure performance et une expérience utilisateur plus proche du natif.
- Les technologies utilisées pour développer une application cross-platform sont Appcelerator, Titanium, Xamarin, React Native et Flutter ou encore Kotlin multi plateforme.



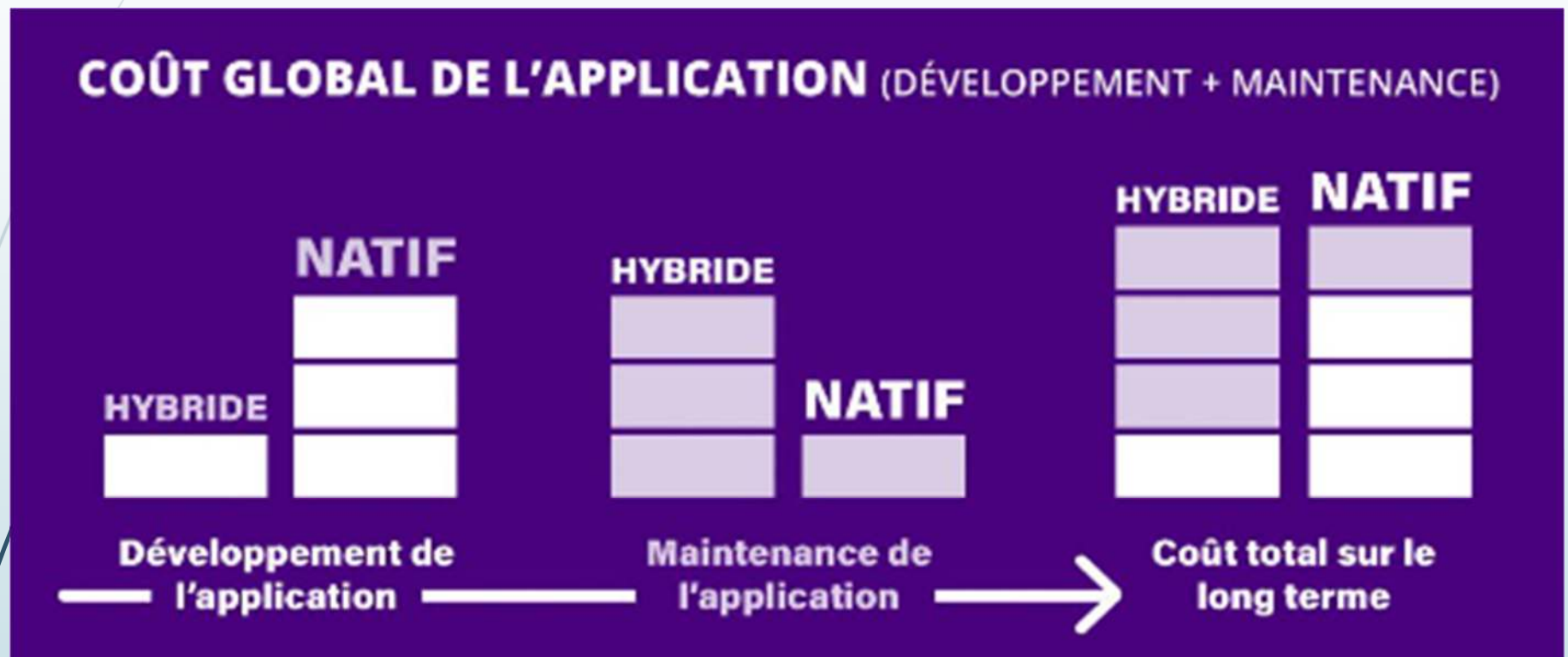
VS



VS

**Critères****NATIVE****HYBRIDE****CROSS  
PLATFORM****Performance  
élevée****Plein usage des  
fonctionnalités du  
terminal mobile****Budget moindre à  
l'achat****Expérience  
utilisateur optimisée  
pour chaque OS****Maintenabilité,  
évolutivité****Planning serré de  
mise à disposition**

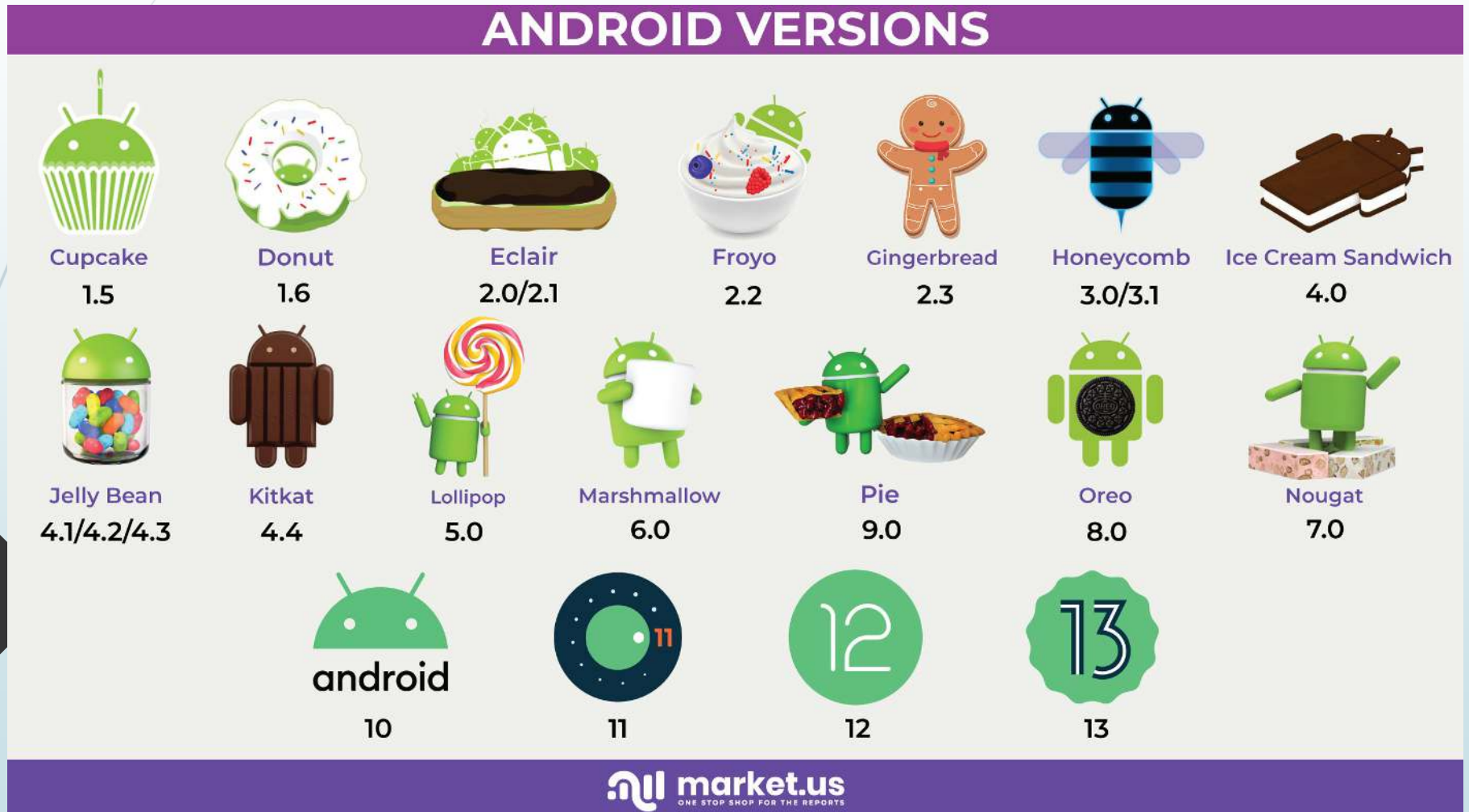
\* Moyennant un effort de développement supplémentaire pour adapter l'appli aux standards de navigation spécifiques à chaque OS  
 \*\* Moyennant des montées de versions plus fréquentes



Référence: <https://www.inflexsys.com/application-native-ou-hybride/>

# Tour d'horizon

- Septembre 2008 : 1<sup>ère</sup> version finale avec le téléphone HTC Dream.



Android 14 bêta Mars 2023





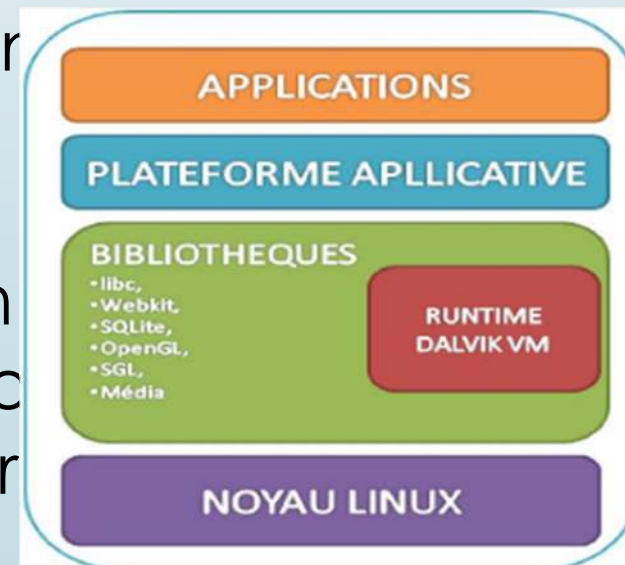
# Définition de l'Android

- C'est un système d'exploitation open source, exclusivement centré sur la téléphonie mobile.
- C'est aussi un ensemble de logiciels destinés à fournir une solution clé en main pour les appareils mobiles, smartphone et tablettes tactiles. Cet ensemble comporte un système d'exploitation (comportant un noyau Kernel Linux et une licence Apache 2.0), des applications clés telles que le navigateur web, le téléphone et le carnet d'adresses,....
- C'est aussi une plateforme de développement open-source pour la création des applications portables.

# Composants d'Android

La plate-forme Android est composée de différentes couches :

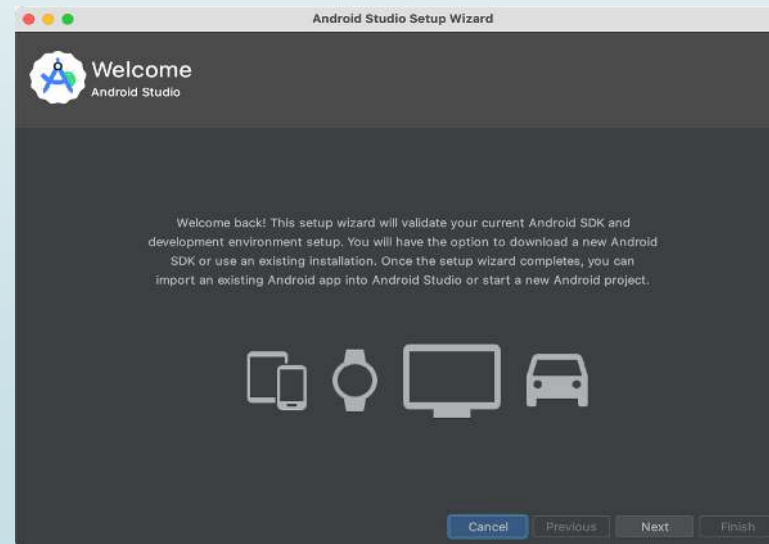
- Un noyau Linux qui lui confère notamment des caractéristiques multitâches ;
- Des bibliothèques graphiques, multimédias ;
- Une machine virtuelle Java adaptée : la « Dalvik Virtual Machine »;
- Un Framework applicatif proposant des fonctionnalités de gestion de téléphonie, de gestion de contenus... ;
- Des applications dont un navigateur Web, une gestion des contacts, un calendrier





# Environnement de développement

- ✓ Le premier prérequis au développement Android est l'installation du JDK (Java Development Kit).
- ✓ Installation de l'Android Studio (Android Studio Giraffe nouvelle UI)  
(<https://developer.android.com/studio>)



Vous pouvez suivre cette [vidéo](#) comme guide d'installation

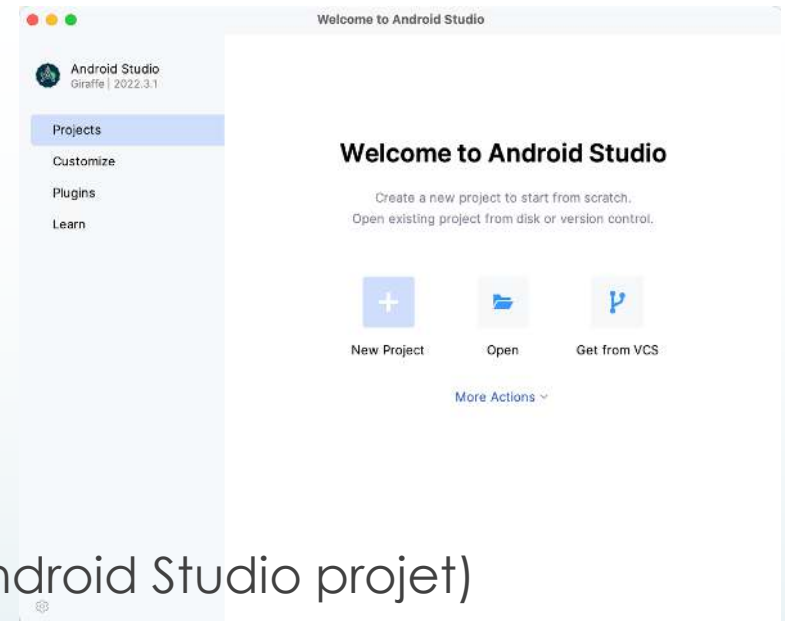
# Prérequis Matériel

- Caractéristiques matérielles nécessaires pour la bonne installation d'Android Studio :
- Minimum 4 GB RAM, recommandé 8 GB RAM, 1 GB pour l'émulateur d'Android.
- Espace disque: 2 GB au minimum, recommandé 4 Gb (500 Mb pour l'IDE et +1.5 Gb pour le SDK d'Android, les images pour l'émulateur et la cache).
- 1280 x 800 la résolution minimale de l'écran.
- Si l'on veut accélérer l'émulateur: il faut un processeur Intel supportant une Technologie de virtualisation avec les extensions VT, VT-x, vmx. Vous pouvez utiliser aussi un processeur AMD supportant une technologie de virtualisation avec les extension AMD-V ou SVM.

# Android Studio

► Les options disponibles sur l'écran d'accueil permettent:

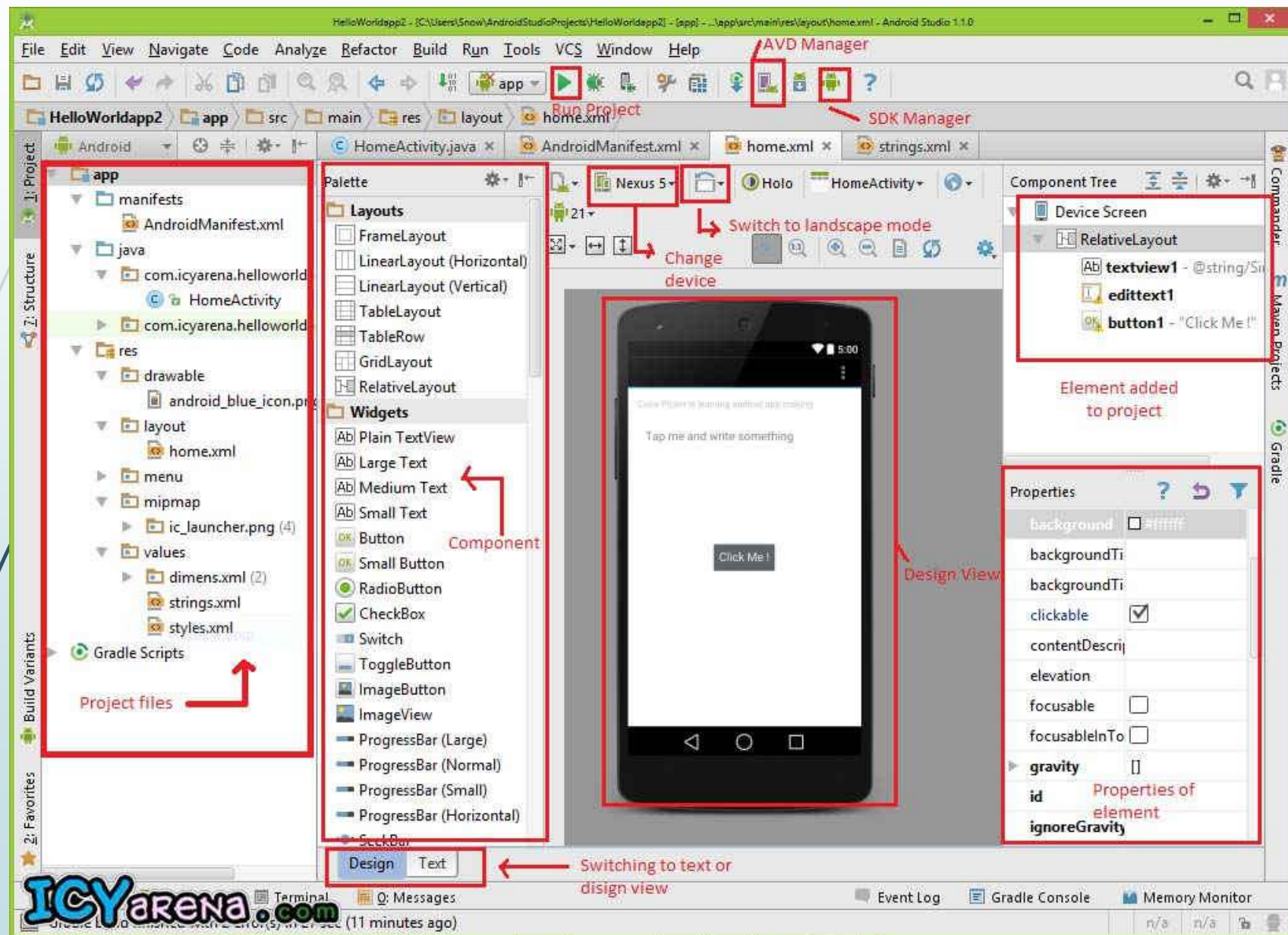
- ✓ Créer un nouveau projet (Start a new Android Studio projet)
- ✓ Ouvrir un projet (open an existing project)
- ✓ Importer un exemple Android (import an Android code sample)
- ✓ Récupérer un projet depuis un gestionnaire de version (check out project from version control)
- ✓ Importer un projet non Android Studio, depuis Eclipse par exemple (import project Eclipse ADT, Gradle, etc))
- ✓ Configurer SDK, plugins, import et export des paramètres ... (Configure)
- ✓ Lire la documentation (Doc and How-Tos)



# Programmation d'application

- Une application Android est composée de :
- Sources Java (ou Kotlin) compilés pour une machine virtuelle « Dalvik » (versions  $\leq 4.4$ ) ou « ART » depuis la version 5
- Fichiers appelés ressources :
  - format XML : interface, textes. . .
  - format PNG : icônes, images. . .
- Manifeste = description du contenu du logiciel
  - fichiers présents dans l'archive
  - demandes d'autorisations
  - signature des fichiers, durée de validité, etc.
- Tout cet ensemble est géré à l'aide d'un IDE (environnement de développement) appelé Android Studio qui s'appuie sur un ensemble logiciel (bibliothèques, outils) appelé SDK Android.

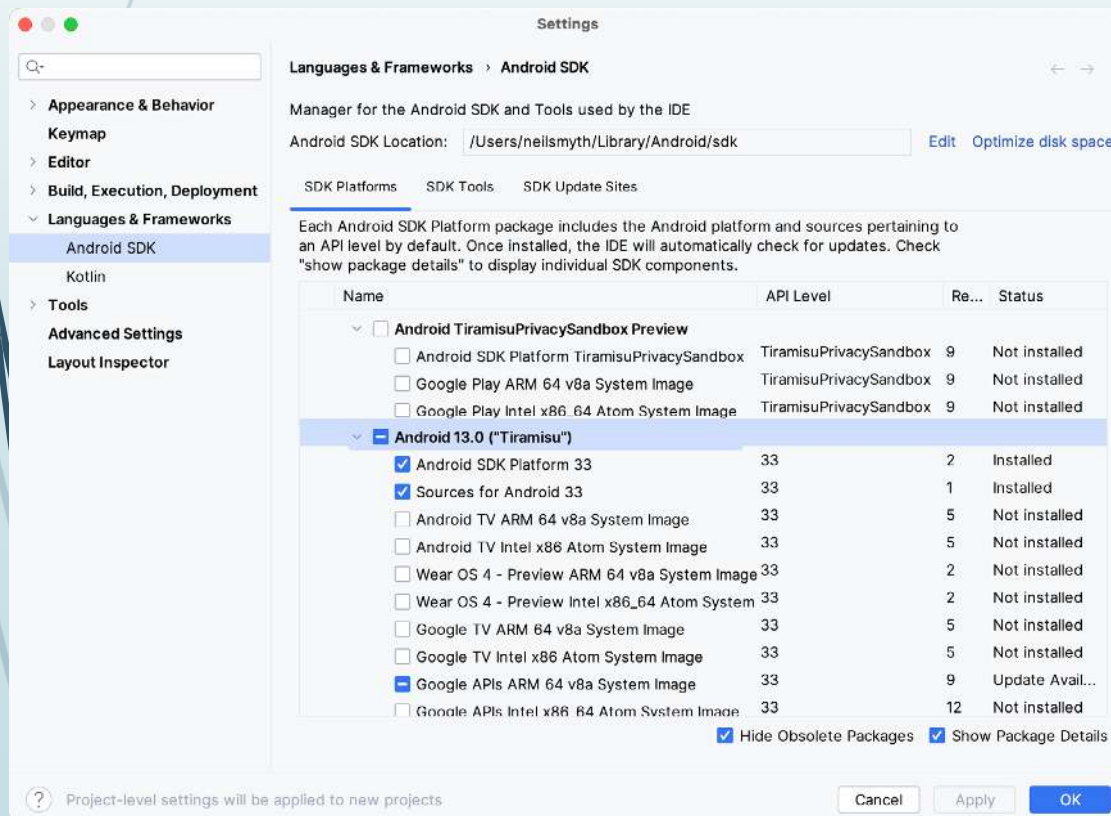
# Interface principale Android Studio





# SDK Manager

IDE Android Studio avec SDK Android par défaut  
Le SDK manager permet de vérifier quels packages sont installés et d'installer tout paquet manquant ou mis à jour



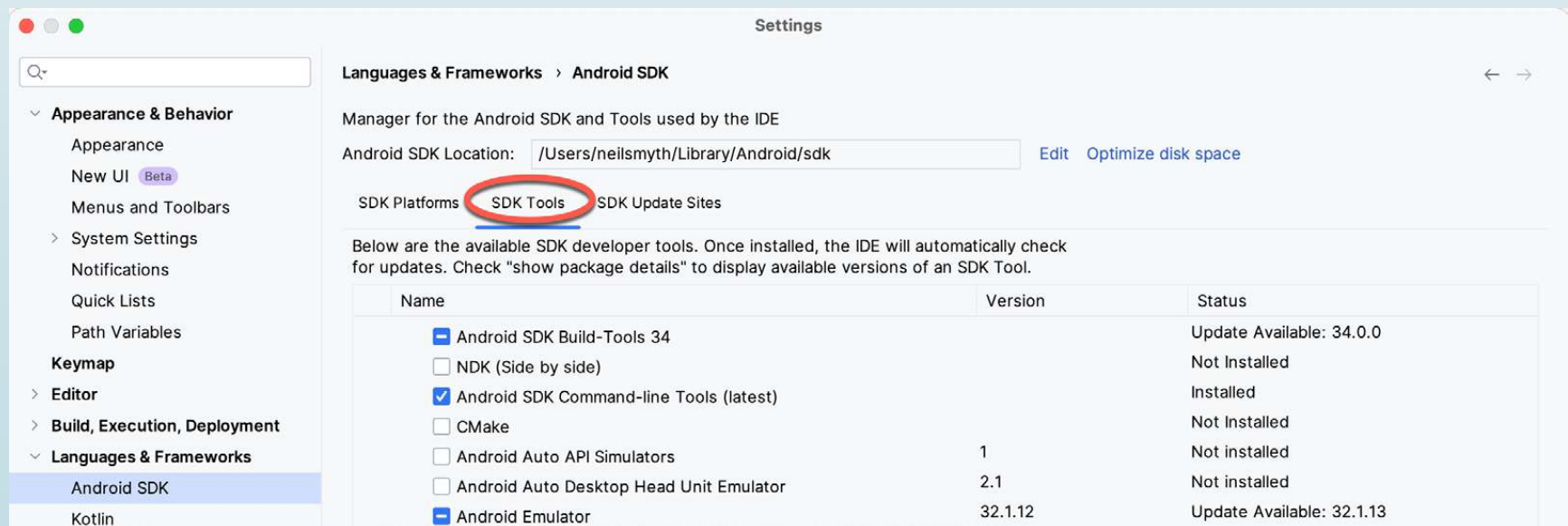
**SDK Platforms:** contient les éléments nécessaires et adaptés à la version d'Android utilisée pour développer l'application. Après avoir installé Android Studio seule la dernière version prise en charge du SDK Android a été installée. Installer les anciennes versions du SDK Android, cela garantit que les applications fonctionnent sur une large gamme d'appareils Android. Dans la liste des versions du SDK, activez cochez la case à côté d'Android 7.0 (Nougat)



**SDK Tools:** contient les éléments nécessaires pour développer, tester et déboguer les applications.

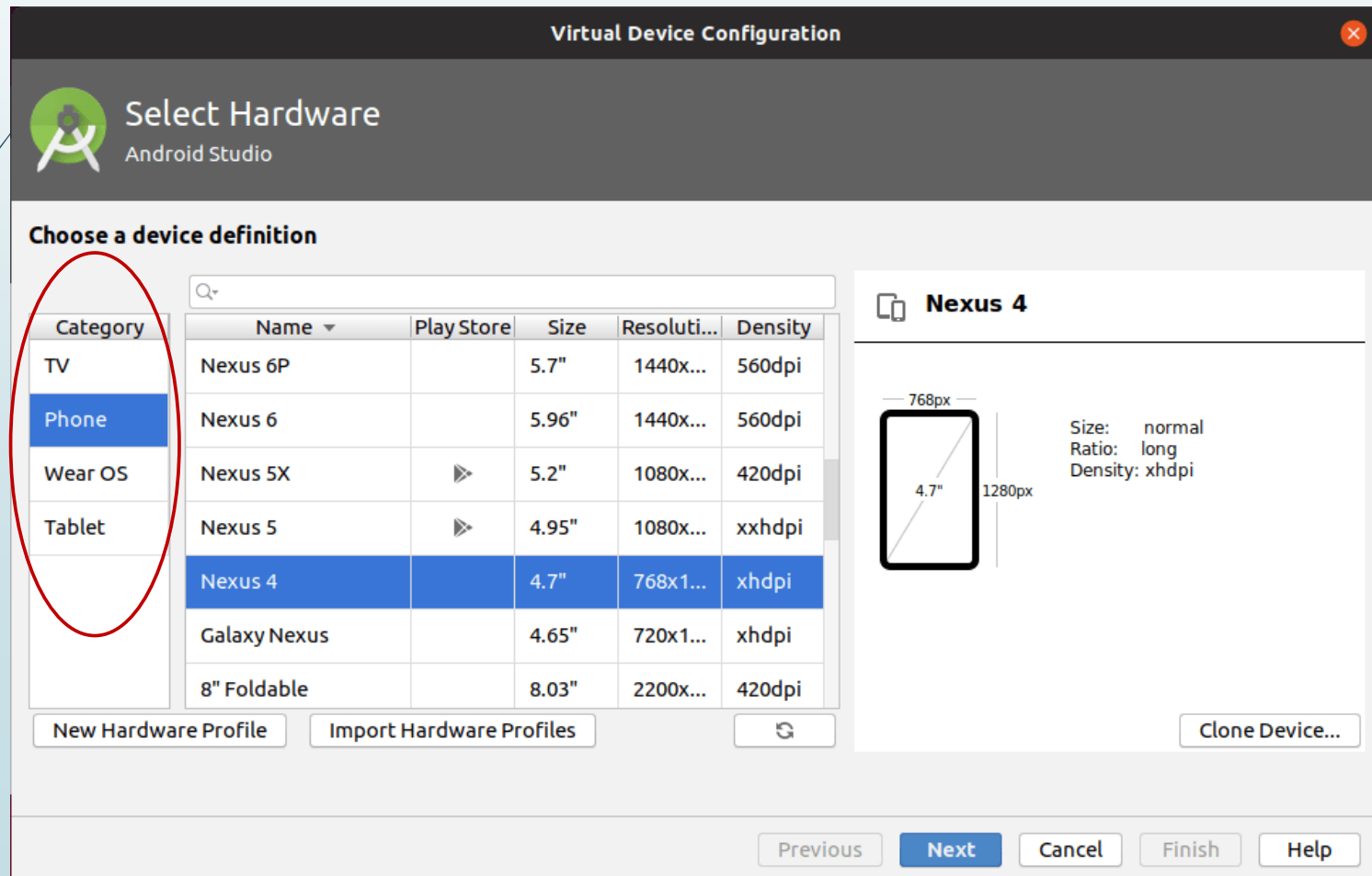
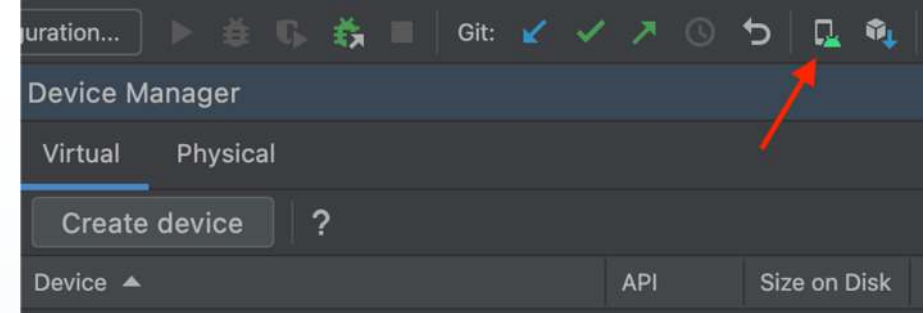
Dans l'écran Outils du SDK Android, assurez-vous que les packages suivants sont installés.

- Android SDK Build-tools
- Android Emulator
- Android SDK Platform-tools
- Google Play Services (contient des outils de google, tel que google map, géolocalisation des utilisateurs.)
- Intel x86 Emulator Accelerator (HAXM installer)
- Google USB Driver (Windows only, test des téléphone google(Galaxy Nexus, Nexus 5/6,...). )
- Layout Inspector image server for API 31 and 34



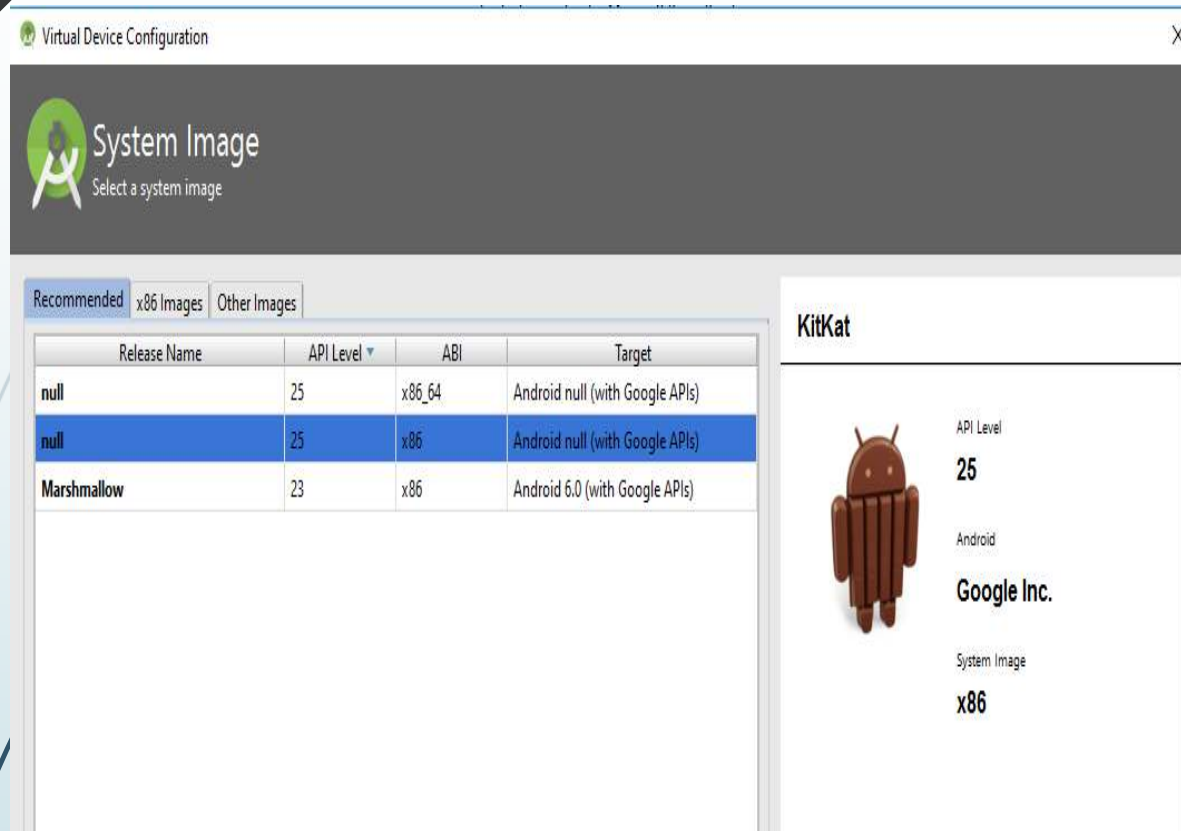
# Emulateur Android

Il permet de simuler(virtualiser) un téléphone, une tablette et une large gamme d'émulateur afin de tester les applications développées.



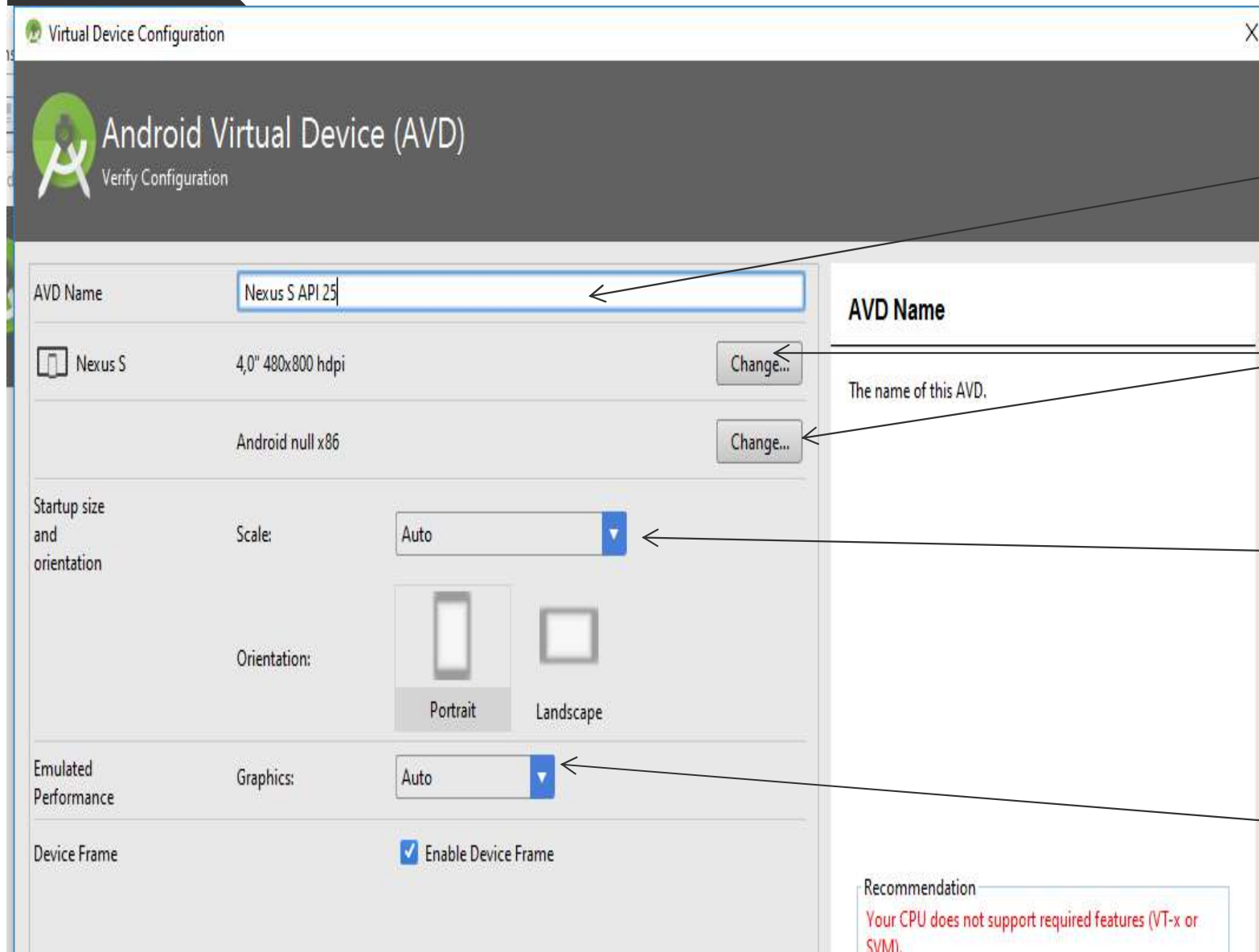
# Emulateur Android

17



Sélectionner la version d'Android qui va être installée sur le device et cliquer sur Next.

# Emulateur Android



Nom du device

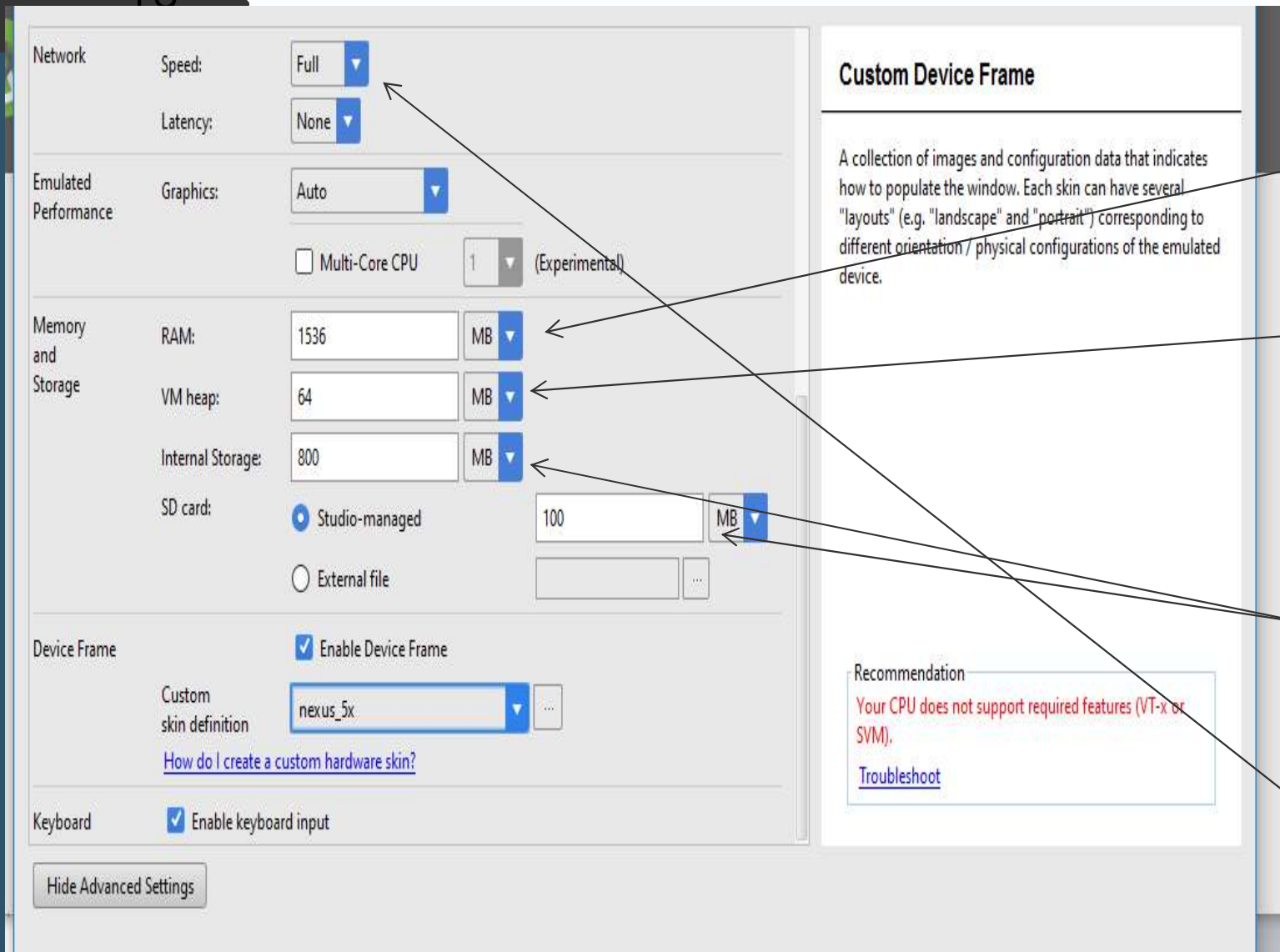
Change son  
type et sa  
version

Android  
Échelle de  
conversion  
dpi/pixel

Emulation de la  
carte graphique

Cliquer sur advanced parameters pour avoir plus  
de paramètres de l'émulateur

# Emulateur Android



Taille du RAM  
du device

Taille à partir  
de laquelle  
Android  
démarre

Garbage  
Collector  
Taille de DD et  
de la mémoire  
externe du  
device

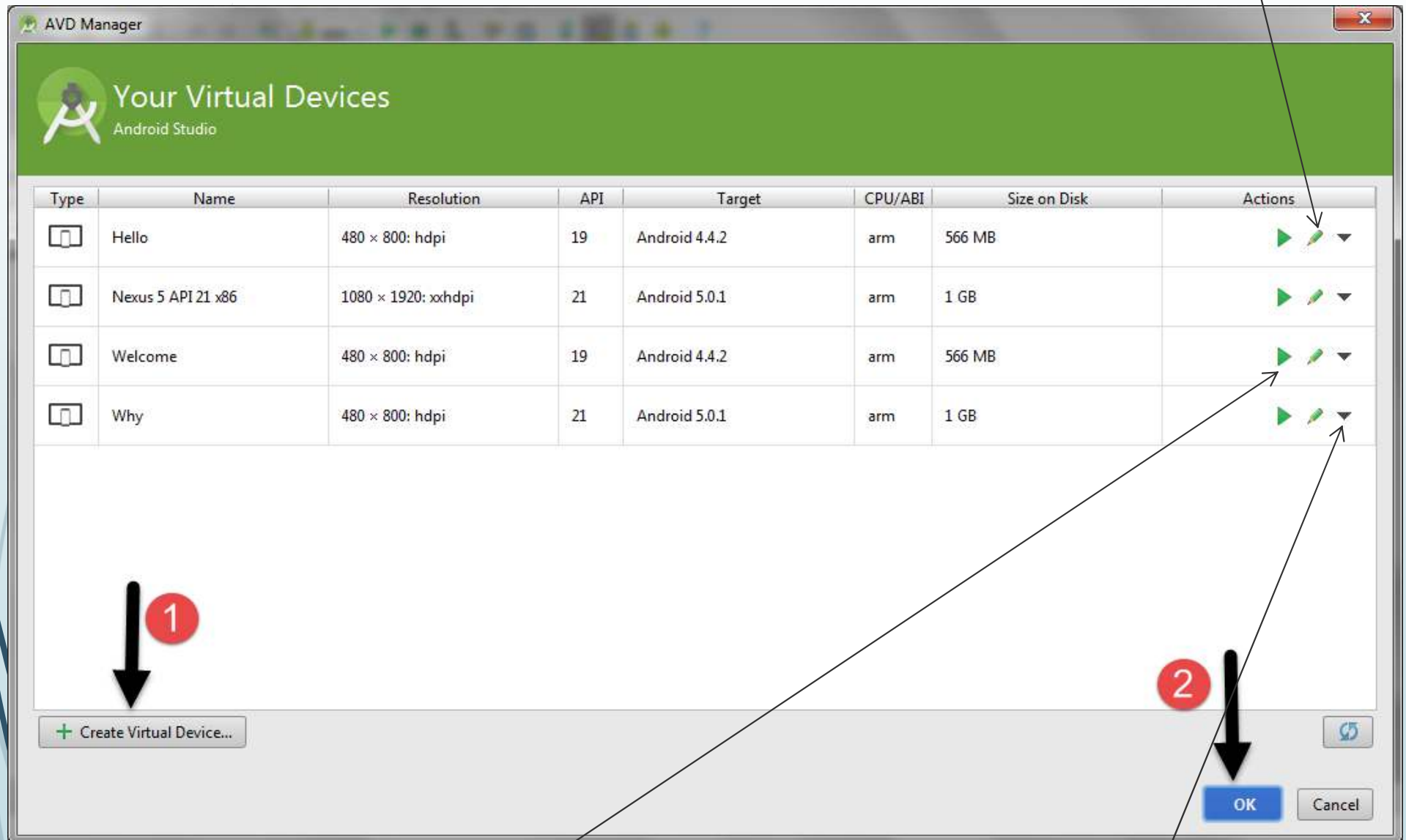
Emulation de la  
connexion réseau

Cliquer sur Finish pour sauvegarder les paramètres  
de l'émulateur

# Emulateur Android

Modifier l'émulateur

20



Démarrer l'émulateur

Supprimer l'émulateur, plus de détails...

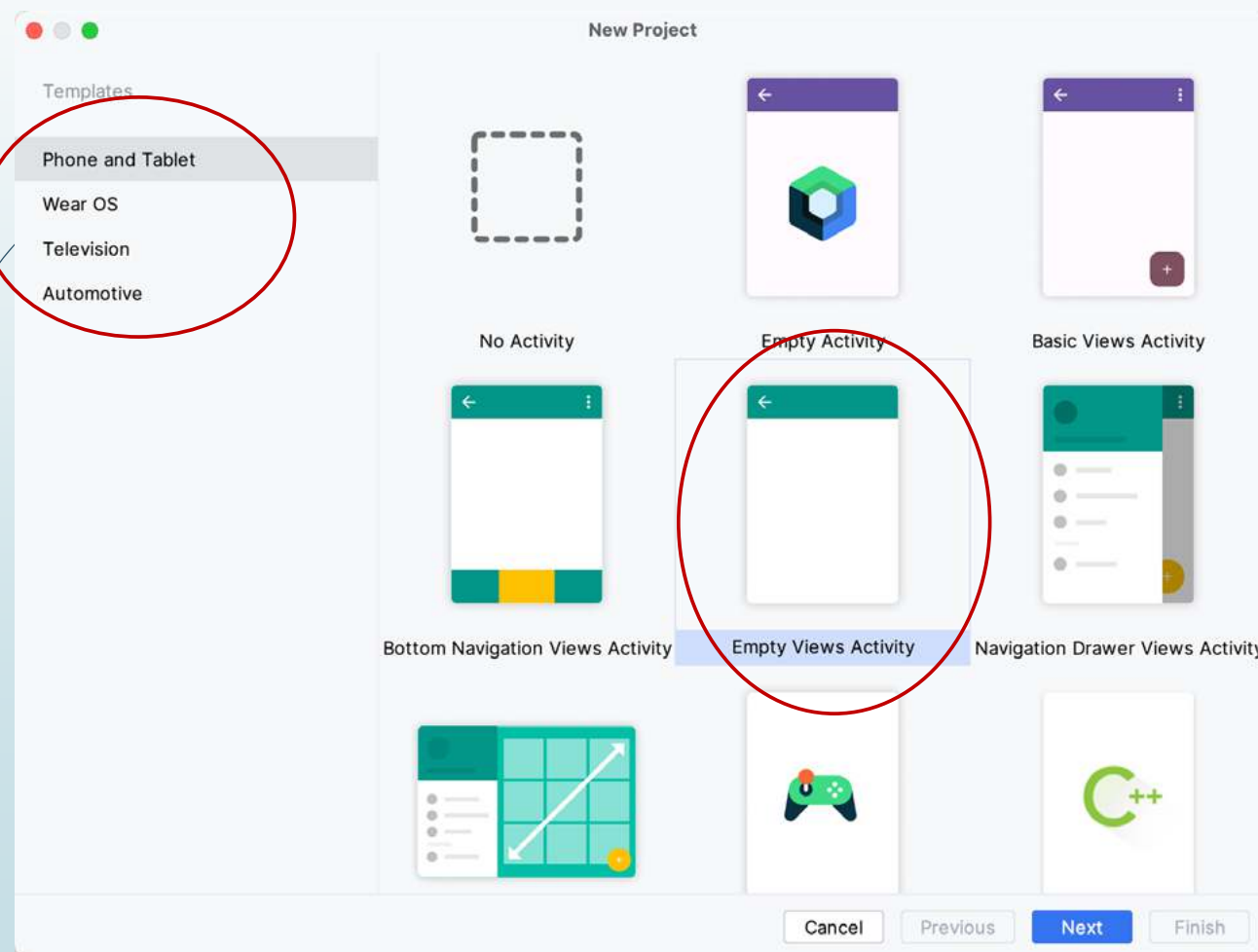


# Première Application



# Assistant de création d'application

Android Studio contient un assistant de creation d'applications : on choisit le type de projet. Pour un premier essai, on se limite au plus simple, *Blank Activity* :



# Choix de la Version

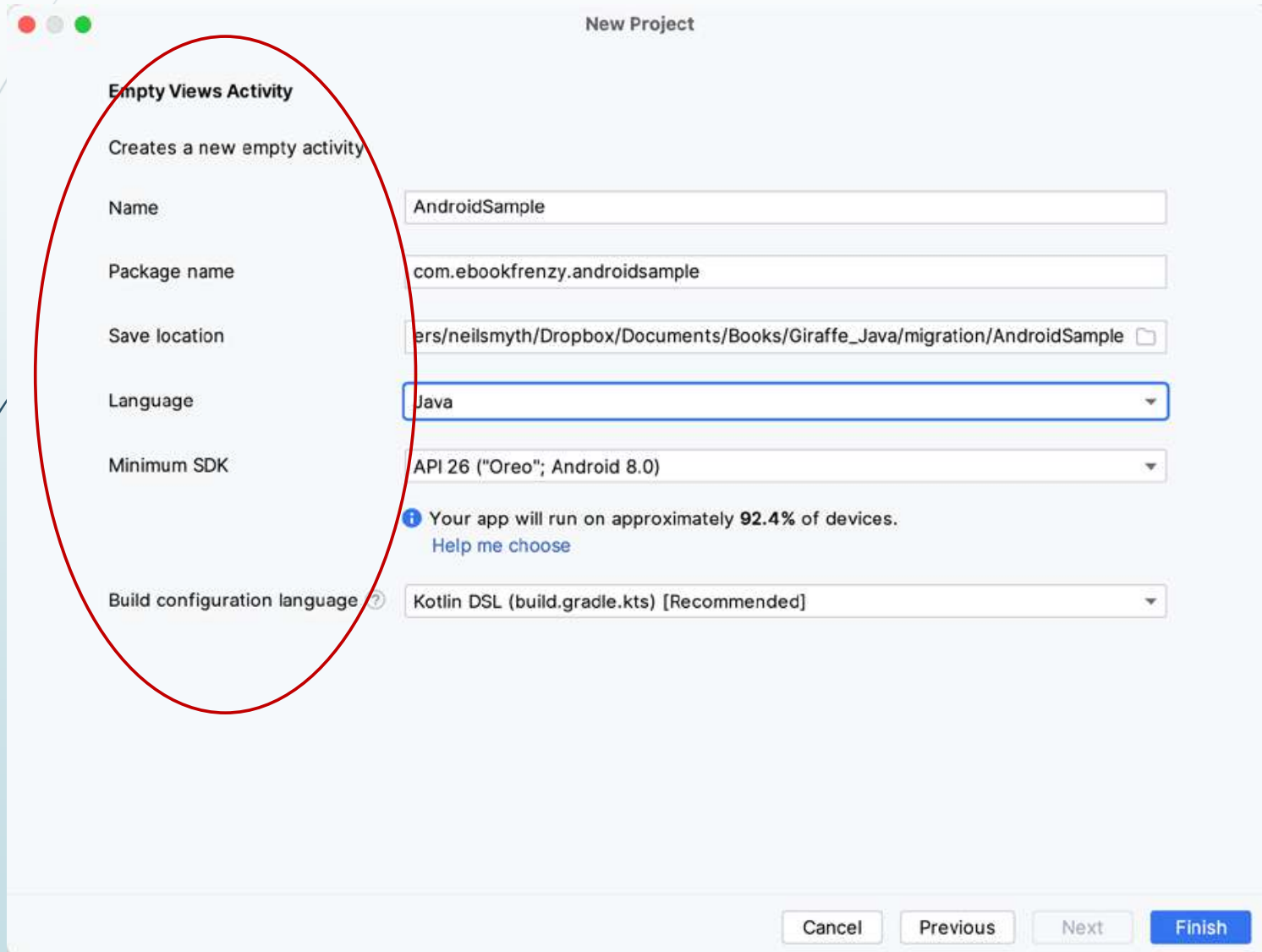
Chaque version d'Android, dénotée par son *API level*, ex: 25, apporte des améliorations et supprime des dispositifs obsolètes.

Toute application exige un certain niveau d'API :

- **minSdkVersion** doit être inférieur pour cibler la couverture maximale des appareils Android sur lesquels l'application sera installée
- **targetSdkVersion**: est la dernière version du système d'exploitation Android sur laquelle vous souhaitez exécuter votre application pour obtenir une optimisation complète des ressources Android. l'application sera testée et marchera correctement jusqu'à ce niveau d'API,
- **compileSdkVersion** : c'est le niveau maximal de fonctionnalités API qu'on se limite à employer. Si on fait appel à quelque chose de plus récent que ce niveau, le logiciel ne se compilera pas.

# Choix de Configuration

Voici comment choisir le Minimum SDK, nom application,...



New Project

**Empty Views Activity**  
Creates a new empty activity

Name: AndroidSample

Package name: com.ebookfrenzy.androidsample

Save location: ers/neilsmyth/Dropbox/Documents/Books/Giraffe\_Java/migration/AndroidSample

Language: Java

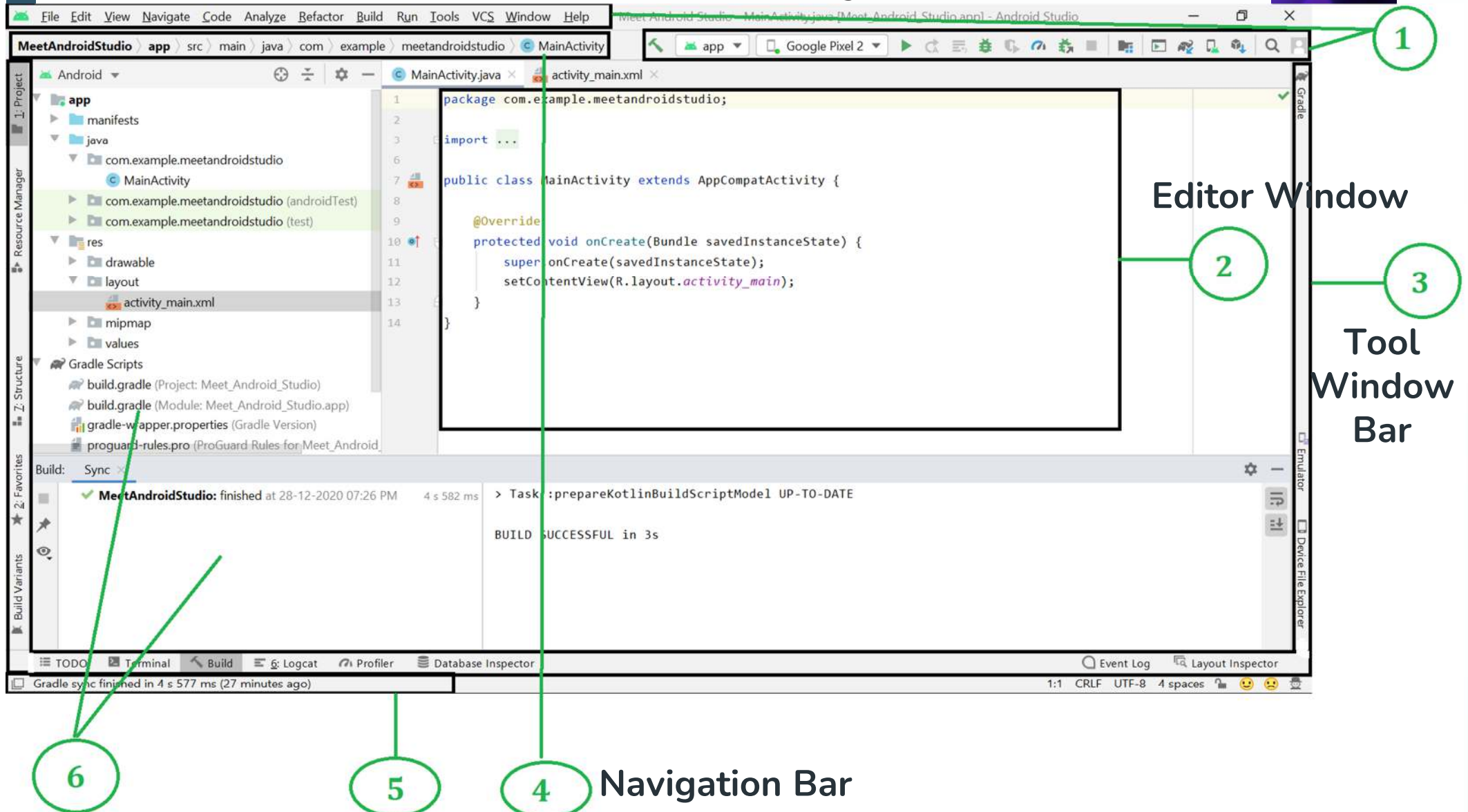
Minimum SDK: API 26 ("Oreo"; Android 8.0)

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

Information: Your app will run on approximately 92.4% of devices.  
[Help me choose](#)

Buttons: Cancel, Previous, Next, Finish

# Fenêtre du projet



Tool Windows    Status Bar  
project management, search, version  
control...

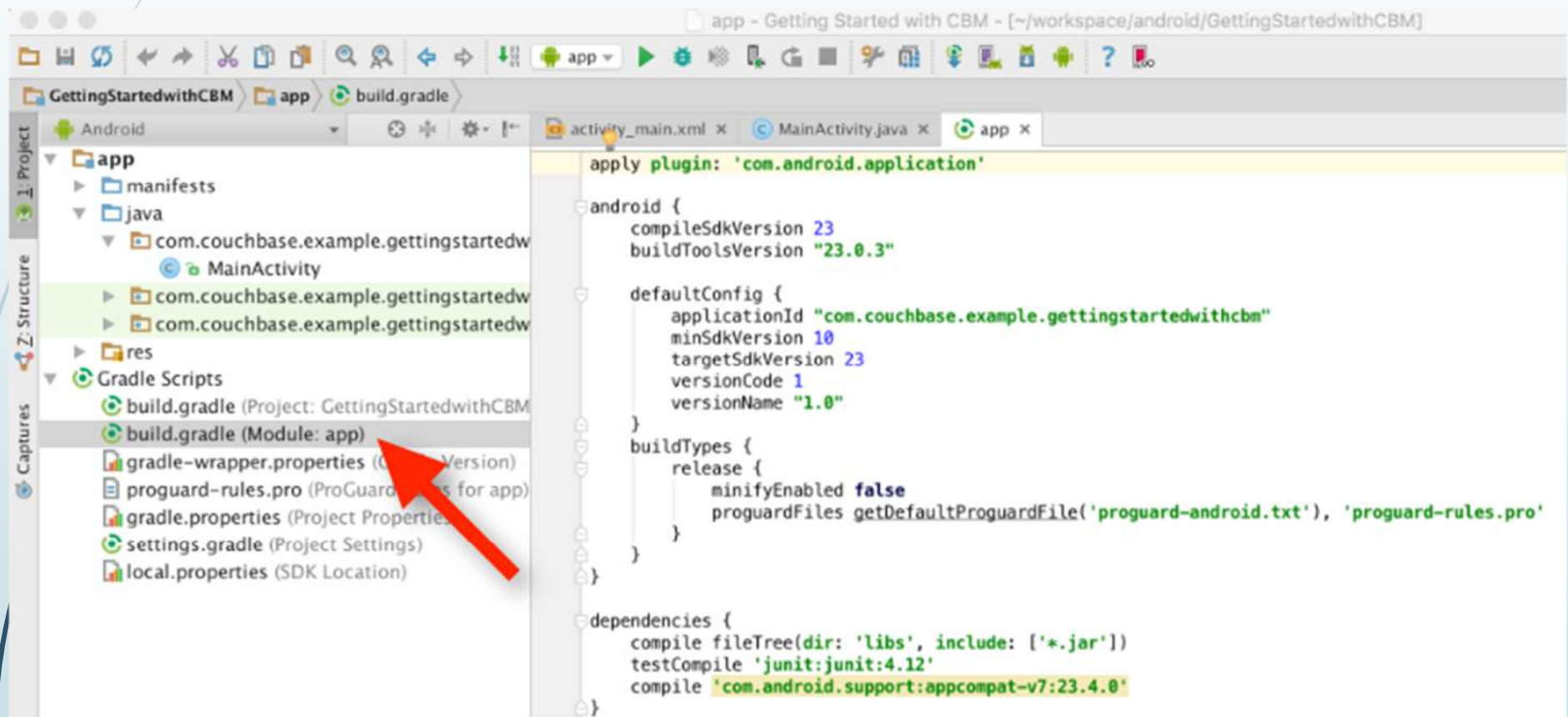
# Structure d'un projet Android

- **AndroidManifest.xml** : Le fichier qui décrit l'application
- **java/** : Répertoire qui contient code source Java de l'application
- **res/** : Répertoire qui contient les ressources (icônes, layouts...)
  - **res/drawable/** : Répertoire qui contient les images (JPG, PNG...)
  - **res/layout/** : Répertoire qui contient les descriptions XML de la composition de l'IHM (les layouts)
  - **res/menu/** : Répertoire qui contient les descriptions XML des menus
  - **res/mipmap/** : dossier est réservé spécifiquement aux différentes icônes utilisées dans l'application (mipmap-hdpi, mipmap-mdpi, etc..)
  - **res/values/** : Répertoire qui contient les messages, les dimensions, chaînes de caractères...(internationalisation)



- **Gradle Scripts**

- **build.gradle** : Le script Gradle qui permet de compiler l'application et de l'installer sur le terminal
- **gradle.properties** et **local.properties** : Deux fichiers de propriétés utilisés par le script Gradle



# STRUCTURE D'UN PROJET ANDROID : LE MANIFEST

- Fichier XML
- Précise l'**architecture** de l'application
- Chaque application doit en avoir un
- **AndroidManifest.xml** est dans la racine du projet
- Précise le nom du package java utilisant l'application. Cela sert d'identifiant unique !
- Décrit les composants de l'application
  - Liste des **activités**...
  - Précise les classes qui les implémentent
  - ...
- Définit les permissions de l'application
  - Droit de passer des appels
  - Droit d'accéder à Internet
  - Droit d'accéder au GPS

# STRUCTURE D'UN PROJET ANDROID : LE MANIFEST

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.mohamed.authentication" >
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme"
  >
    <activity
      android:name=".Login"
      android:label="@string/app_name"
      android:theme="@style/AppTheme.NoActionBar"
    >
      ...
    </activity>
    <activity android:name=".Admin"
      android:label="@string/title_activity_admin"
      android:theme="@style/AppTheme.NoActionBar"
    >
    </activity>
  </application>
</manifest>
```

# Les Ressources

- Les ressources de l'application sont utilisées dans le code au travers de la classe statique **R**
- La classe R s'agit d'une classe statique régénérée à chaque fois que des ressources sont ajoutées au projet.
- Les ressources sont utilisées de la manière suivante:

```
android.R.type_ressource.nom_ressource
```

- Cette méthode permet de retourner l'identifiant de la ressource
- Par exemple, pour récupérer une ressource de type String d'identifiant « hello »

```
Resources res = getResources();  
String chaine = res.getString(R.string.hello);
```

- Une méthode spécifique pour les objets graphiques permet de les récupérer à partir de leur id, c'est la méthode **findViewById**.
- Cette méthode permet d'agir sur des instances d'objets graphiques créés via leur définition XML

```
TextView texte = (TextView)findViewById(R.id.le_texte);  
texte.setText("Bonjour!");
```

# Les Ressources: les chaines

- Les chaines constantes de l'application se situent dans le fichier **res/values/string.xml**.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">Bonjour</string>
  <string name="app_name">BonjourM2I</string>
</resources>
```

- La récupération de la chaine se fait via le code:

```
Resources res = getResources();
String hw = res.getString(R.string.hello);
```

# Les Ressources: internationalisation

- L'exploitation du fichier strings.xml pour la déclaration des chaînes de caractères est très utile pour **l'internationalisation** de l'application.
- Pour ce faire, il suffit de créer des répertoires **values-XX** ou **XX** est le code de la langue qu'on souhaite implanter.
- On obtient par exemple pour les langages anglais et français l'arborescence suivante:

```
MyProject/  
  res/  
    values/  
      strings.xml  
    values-en/  
      strings.xml  
    values-fr/  
      strings.xml
```



# Editeur spécifique

Les ressources (disposition des vues dans les interfaces, menus, images vectorielles, textes. . . ) sont définies à l'aide de fichiersXML.

Studio fournit des éditeurs spécialisés pour ces fichiers, par exemple :

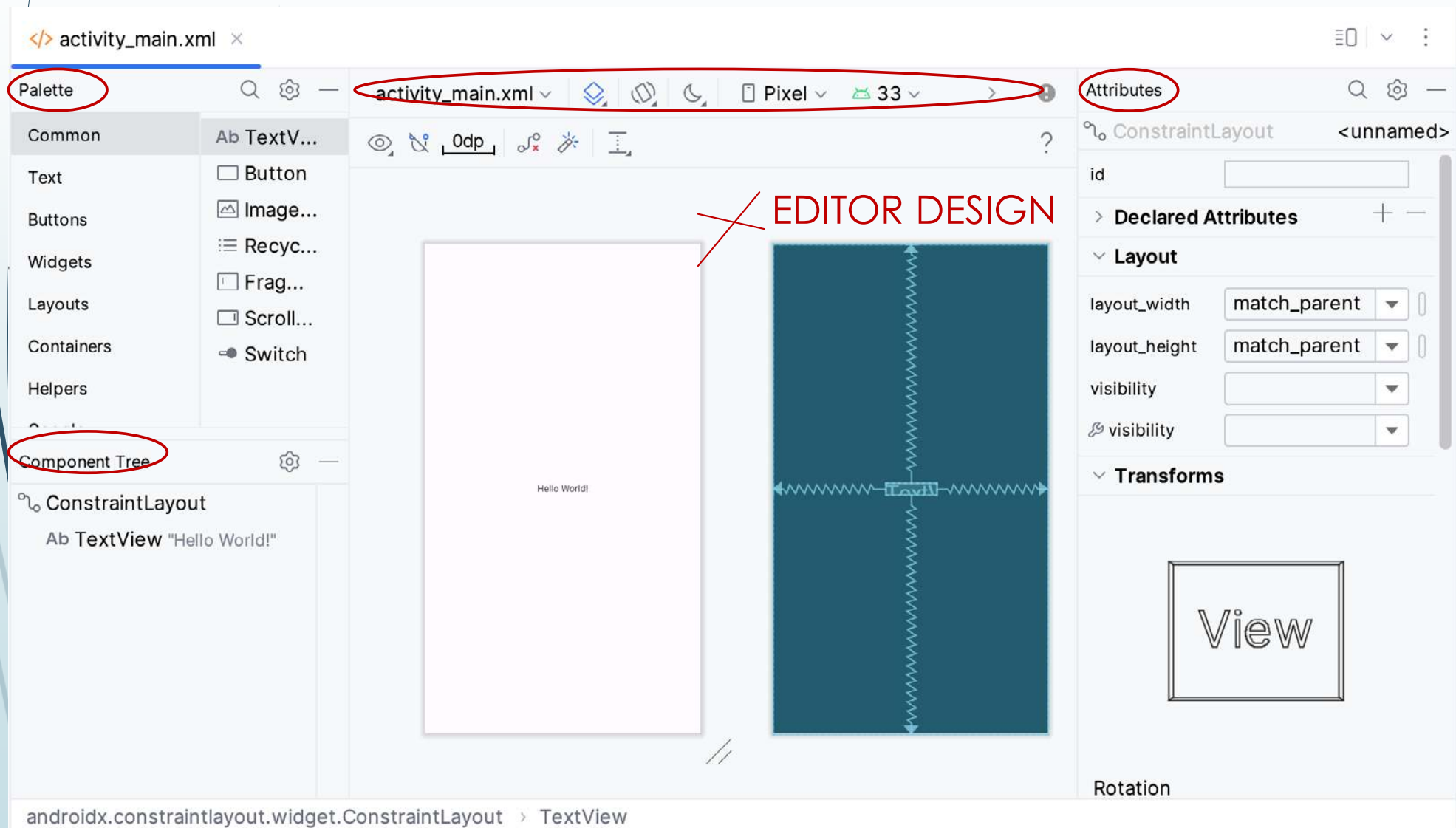
Formulaires pour :

- `res/values/strings.xml` : textes de l'interface.

Éditeurs graphiques pour :

- `res/layout/*.xml` : disposition des contrôles sur l'interface.

# Exemple res/layout/main.xml



# Reconstruction du Projet

Chaque modification d'un source ou d'une ressource fait **reconstruire** le projet. C'est automatique.

Dans certains cas (travail avec un gestionnaire de sources comme Subversion ou Git), il peut être nécessaire de reconstruire manuellement. Il suffit de sélectionner le menu Build/Rebuild Project.

En cas de confusion d'Android Studio (compilation directe en ligne de commande), ou de mauvaise mise à jour des sources (partages réseau), il faut parfois nettoyer le projet. Sélectionner le menu Build/Clean Project.

**Ces actions lancent l'exécution de *Gradle*.**

# Gradle

Gradle est un outil de construction de projets comme Make(projets C++ sur Unix), Ant(projets Java dans Eclipse) et Maven.

De même que make se sert d'un fichier Makefile, Gradle se sert d'un fichier nommé build.gradle pour construire le projet.

C'est assez compliqué car AndroidStudio fait une distinction entre le projet global et l'application. Donc il y a deux build.gradle :

- un **script build.gradle** dans le dossier **racine** du projet. Il indique quelles sont les dépendances générales (noms des dépôts Maven contenant les librairies utilisées).
- un dossier **app** contenant l'application du projet.
- un **script build.gradle** dans le dossier **app** pour compiler l'application.

# Mise A jour

Le SDK ainsi que Gradle sont régulièrement mis à jour, automatiquement avec AndroidStudio. Cependant, vous devrez parfois éditer les build.gradle à la main pour en tenir compte.

Par exemple, ce build.gradle de projet fait appel à Gradle 3.0.1 et Realm 4.3.2.

```
buildscript {  
    repositories { ... }  
    dependencies {  
        classpath 'com.android.tools.build:gradle:3.0.1'  
        classpath 'io.realm:realm-gradle-plugin:4.3.2'  
    }  
}
```

Il faudra changer les numéros de version manuellement en cas de mise à jour, puis reconstruire le projet (*sync now* ou *try\_again*).

# Utilisation de Bibliothèques

Certains projets font appel à des bibliothèques externes. Cela fait généralement rajouter quelques lignes dans les deux build.gradle.

Par exemple, Realm (une base de données distribuée)

dans le build.gradle du dossier app:

```
apply plugin: 'realm-android'
```

dans le build.gradle à la racine du projet:

```
dependencies {  
    classpath "io.realm:realm-gradle-plugin:5.8.0"  
}
```

La reconstruction du projet fait automatiquement télécharger la bibliothèque.



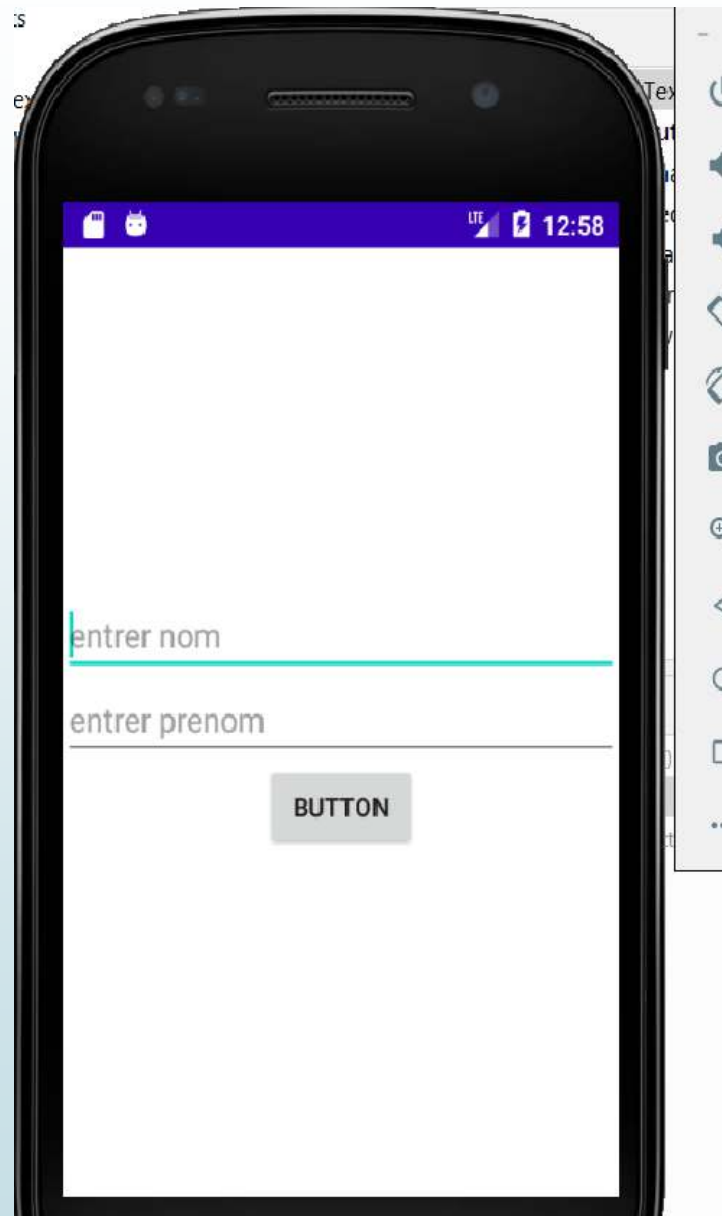
# Exécution de l'Application

L'application est prévue pour tourner sur un appareil (smartphone ou tablette) **réel** ou **simulé** (virtuel).

Le SDK Android permet de :

- Installer l'application sur une vraie tablette connectée par USB
- Simuler l'application sur une tablette virtuelle *AVD*

# L'application sur l'AVD



L'apparence change d'une version à l'autre du SDK.



## Ccommunication AVD - Android Studio

# Fenêtre Android

Android Studio affiche plusieurs fenêtres utiles indiquées dans l'onglet tout en bas :

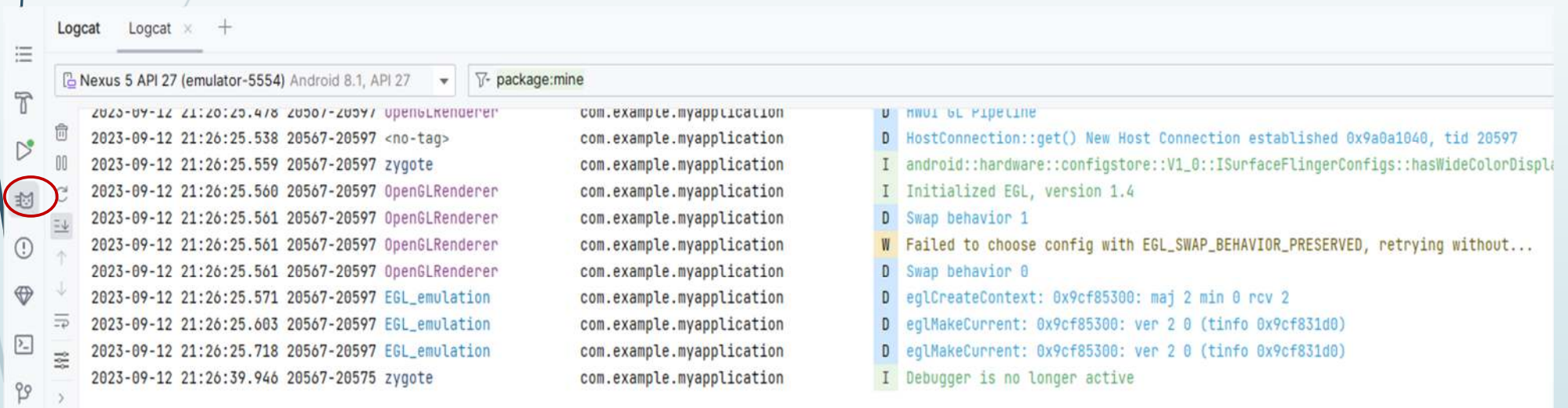
**Logcat** Affiche tous les messages émis par la tablette courante

**Messages** Messages du compilateur et du studio

**Terminal** Shell unix permettant de lancer des commandes dans le dossier du projet.

# Fenêtre LogCat

Des messages détaillés sont affichés dans la fenêtre LogCat:



Ils sont émis par les applications : debug, infos, erreurs. . . comme syslog sur Unix : date, heure, gravité, source (code de l'émetteur) et message.

# Filtrage des messages

Il est commode de définir des *filtres* pour ne pas voir la totalité des messages de toutes les applications de la tablette :

- sur le niveau de gravité : verbose, debug, info, warn, error et assert,
- sur l'étiquette *TAG* associée à chaque message,
- sur le *package* de l'application qui émet le message.



# Emission d'un message vers LogCat

Une application émet un message par ces instructions :

```
Import android.util.Log;

Public class MainActivity extends Activity {
    public static final String TAG ="monappli";

    void maMethode() {
        Log.i(TAG,"appel de maMethode()");
    }
}
```

Fonctions Log.\*:

Log.i(String tag, String message) affiche une info,

Log.w(String tag, String message) affiche une alerte,

Log.e(String tag, String message) affiche une erreur.

Log.d(String tag, String message) affiche debug.



## Création d'un paquet installable

Sources Java

Bytecode Java

Bytecode Dalvik  
(optimisé)



/data/app

Ressources + Manifest



Application  
empaquetée  
+ signée

# Paquet

Un paquet Android est un fichier **.apk**.

C'est une archive signée (authenticifiée) contenant les binaires, ressources compressées et autres fichiers de données.

La création est relativement simple avec Studio :

1. Menu contextuel du projet Build..., choisir Generate Signed APK,
2. Signer le paquet à l'aide d'une *clé privée*,
3. Définir l'emplacement du fichier .apk.

Le résultat est un fichier .apk dans le dossier spécifié.

# Signature d'une Application

Lors de la mise au point, Studio génère une clé qui ne permet pas d'installer l'application ailleurs. Pour distribuer une application, il faut une *clé privée*.

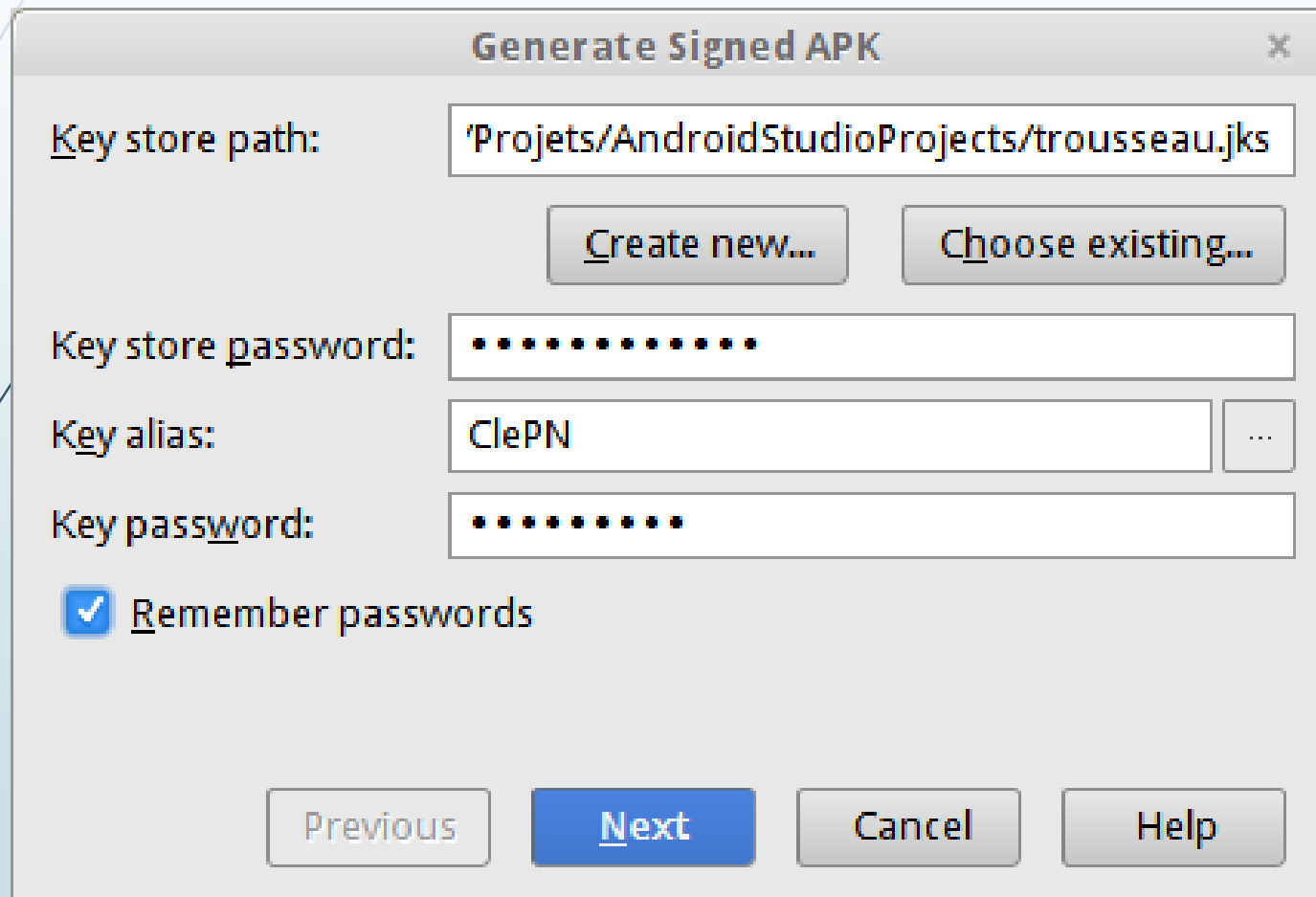
Les clés sont stockées dans un *keystore* = trousseau de clés. Il faut le créer la première fois. C'est un fichier crypté, protégé par un mot de passe, à ranger soigneusement.

Ensuite créer une *clé privée* :

alias = nom de la clé, mot de passe de la clé  
informations personnelles complètes : prénom, nom,  
organisation, adresse, etc.

Les mots de passe du trousseau et de la clé seront demandés à chaque création d'un .apk. **Ne les perdez pas.**

# Création du *keystore*



**Generate Signed APK** [X]

Key store path: Projets/AndroidStudioProjects/trousseau.jks

[Create new...] [Choose existing...]

Key store password: [Masked]

Key alias: ClePN [...]

Key password: [Masked]

☒ Remember passwords

[Previous] [Next] [Cancel] [Help]



# Création d'une clé

**New Key Store**

Key store path: /home/pierre/Projets/AndroidStudioProjects/trousseau.jks

Password: ..... Confirm: .....

Key

Alias: ClePN

Password: ..... Confirm: .....

Validity (years): 25

Certificate

First and Last Name: Pierre Nerzic

Organizational Unit: Département Informatique

Organization: IUT de Lannion

City or Locality: Lannion

State or Province: Côtes d'Armor 22

Country Code (XX): FR

OK Cancel

# Création du paquet

Ensuite, Studio demande où placer le .apk:

