

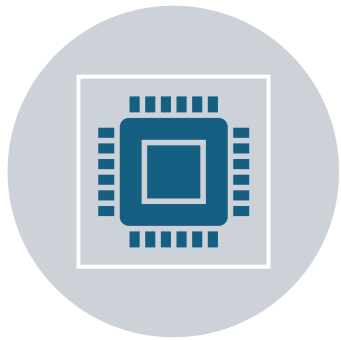
Cours de Programmation Mobile Android

Par: Dr ANIBOU Chaimae

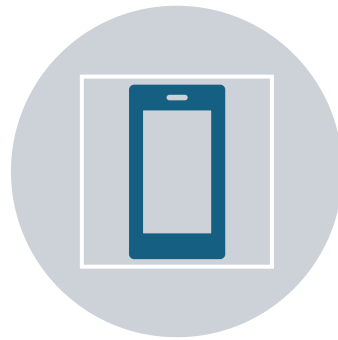


- Copyright:ANIBOU Chaimae

Qu'est-ce qu'une application mobile ?



UNE APPLICATION MOBILE EST UN LOGICIEL DÉVELOPPÉ SPÉCIFIQUEMENT POUR ÊTRE UTILISÉ SUR UN APPAREIL MOBILE, TEL QU'UN SMARTPHONE OU UNE TABLETTE.



CES APPLICATIONS SONT CONÇUES POUR FOURNIR DES FONCTIONNALITÉS SPÉCIFIQUES AUX UTILISATEURS, TELLES QUE LA COMMUNICATION, LE DIVERTISSEMENT, LA PRODUCTIVITÉ, OU TOUT AUTRE SERVICE ADAPTÉ À UN USAGE MOBILE.



ELLES SONT GÉNÉRALEMENT TÉLÉCHARGÉES ET INSTALLÉES À PARTIR D'UNE BOUTIQUE D'APPLICATIONS, COMME L'APP STORE POUR IOS OU GOOGLE PLAY STORE POUR ANDROID.

Application mobile (Principaux OS)



Les principaux systèmes d'exploitation pour les applications mobiles sont :



- Android : Développé par Google, utilisé sur une grande variété d'appareils mobiles.



- iOS : Développé par Apple, utilisé exclusivement sur les appareils iPhone et iPad.



- Windows Phone : Développé par Microsoft, utilisé principalement sur les appareils Windows Phone.



- BlackBerry OS : Développé par BlackBerry, utilisé sur les appareils BlackBerry.



- HarmonyOS : Développé par Huawei, visant à être utilisé sur divers appareils intelligents.

Qu'est-ce qu'Android OS ?

Android OS est un système d'exploitation mobile développé par Google.

Il est basé sur le noyau Linux et est principalement utilisé dans les smartphones et les tablettes.

Android OS est connu pour sa flexibilité, sa personnalisation et son large éventail d'applications disponibles sur Google Play Store.

Composants de l'écosystème Android



L'écosystème Android comprend plusieurs composants clés :



- Système d'exploitation Android : Le noyau Linux et la couche logicielle développée par Google.



- Appareils Android : Smartphones, tablettes, téléviseurs, montres intelligentes, etc.



- Google Play Store : La principale plateforme de distribution d'applications pour Android.



- Développeurs Android : La communauté de développeurs qui créent des applications pour Android.

Android pour qui ?



Android est conçu pour une large gamme d'utilisateurs, notamment :



- Utilisateurs finaux : Les particuliers qui utilisent des smartphones et des tablettes pour diverses activités telles que la communication, le divertissement, et la productivité.

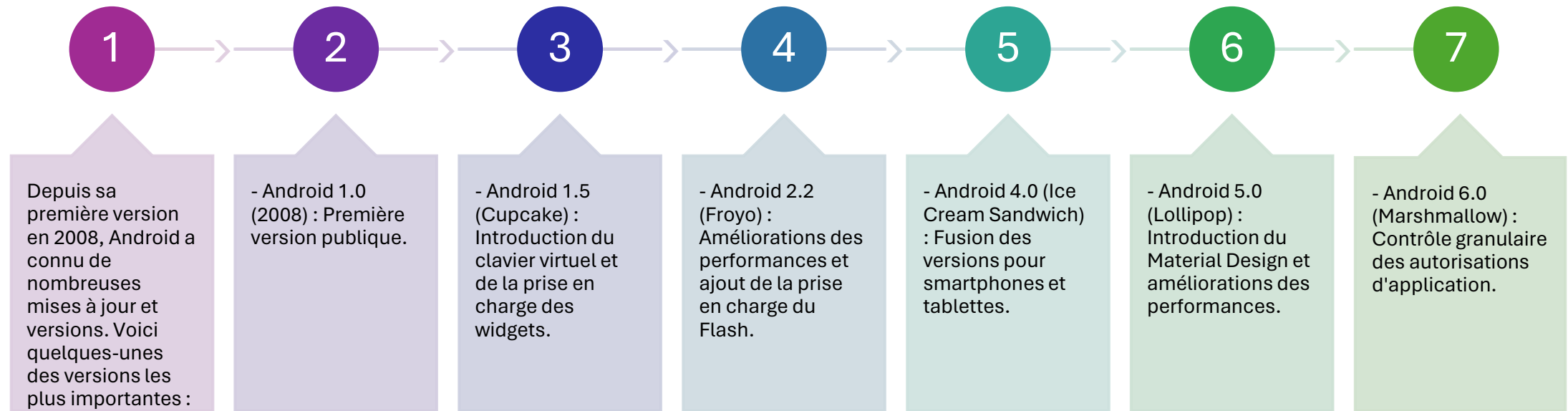


- Développeurs d'applications : Les professionnels de développement qui créent des applications pour la plateforme Android, profitant de sa large base d'utilisateurs et de son écosystème riche.

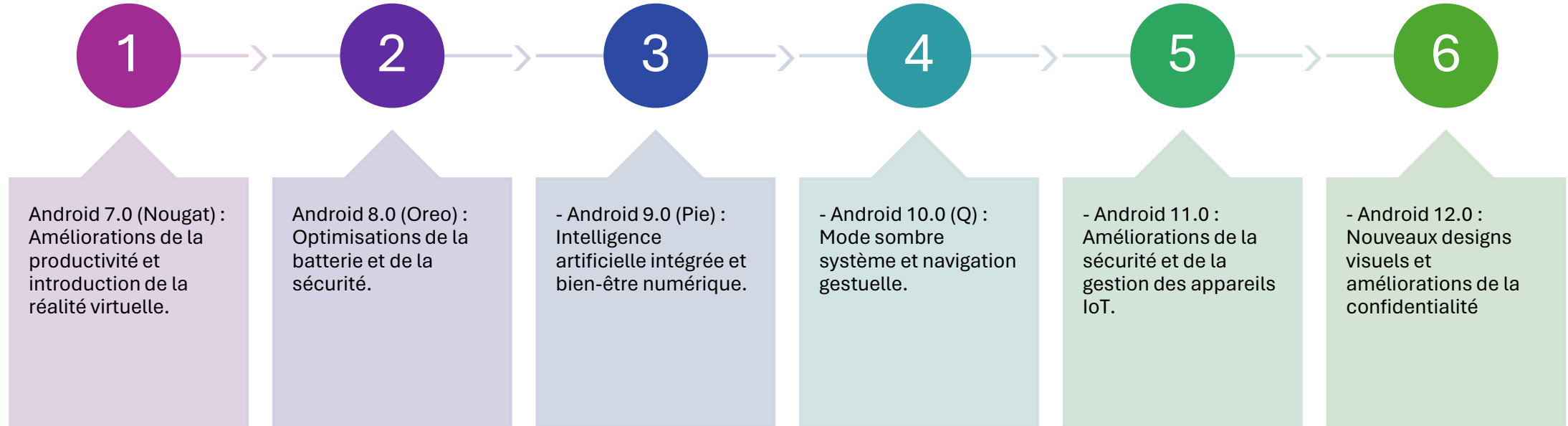


- Fabricants de dispositifs : Les entreprises qui produisent des appareils mobiles utilisant Android comme système d'exploitation pour leurs produits.

Historique des versions d'Android



Historique des versions d'Android



Architecture d'Android

L'architecture d'Android est basée sur une pile de logiciels en couches, comprenant les éléments suivants :

- Noyau Linux : Fournit des fonctions de base du système d'exploitation.

- Bibliothèques Android : Ensemble de bibliothèques écrites en langage C/C++ utilisées par différentes composantes d'Android.

- Couche d'abstraction matérielle (HAL) : Permet à la plateforme Android de communiquer avec le matériel sous-jacent.

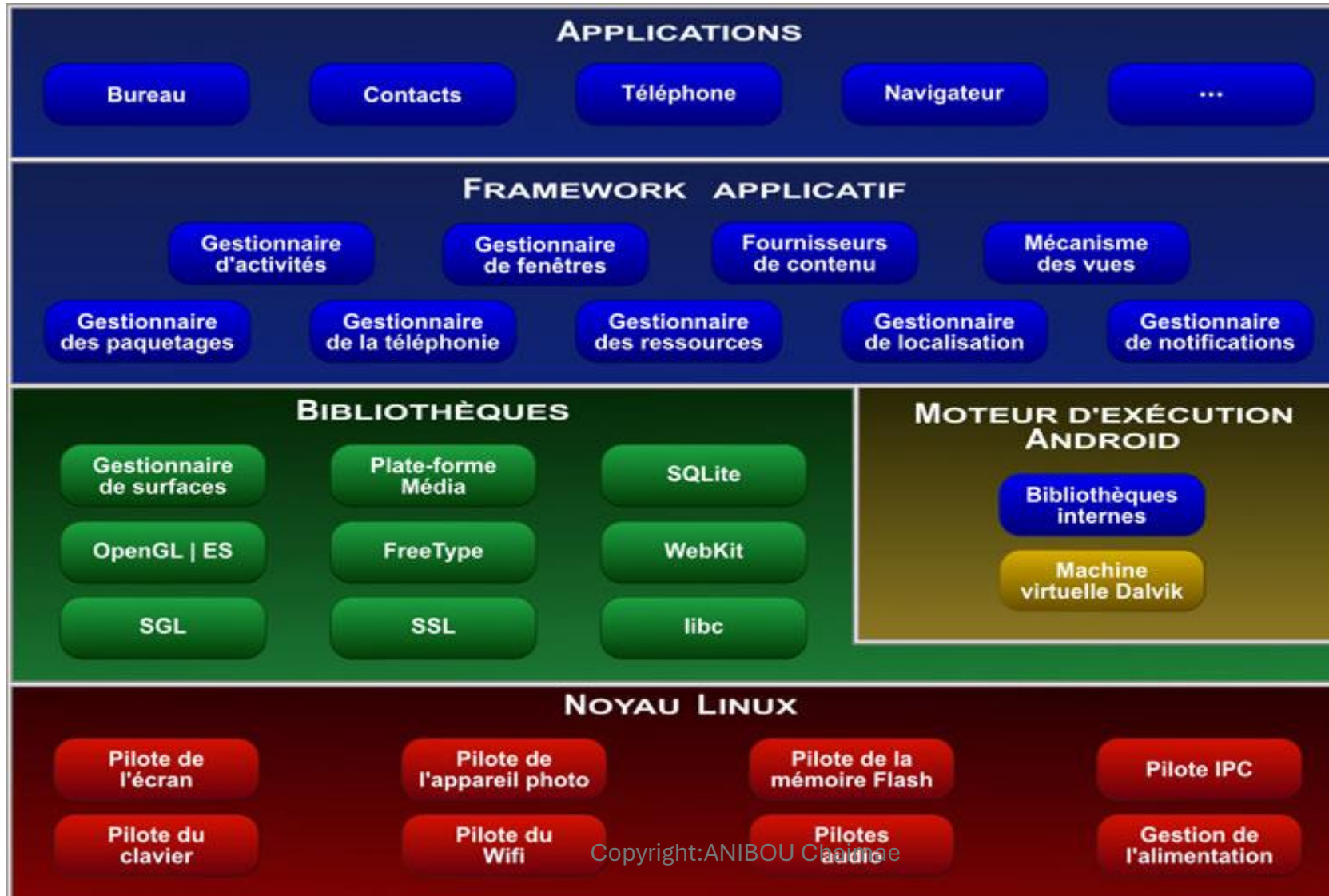
- Android Runtime (ART) : Environnement d'exécution pour les applications Android, exécutant le bytecode Dalvik.

- Cadre d'application Android : Fournit une infrastructure pour le développement d'applications Android, y compris les composants d'interface utilisateur et les services système.

- Applications système : Applications intégrées fournies par défaut avec Android, telles que le navigateur, le calendrier et les contacts.

- Applications tierces : Applications développées par des tiers et disponibles sur le Google Play Store ou d'autres sources.

Composants d'Android



Composant : Noyau Linux



Couche basse de la plateforme pour la gestion des services du système

Gestion de la mémoire
Gestion des processus
Pilotes matériels
Gestion réseau
Sécurité



Première abstraction entre matériel et couche applicative



Gestion du Power Management



Binder pour faciliter la communication inter processus

Composant : Librairies natives

Librairie système C : libc

- Dérivée de BSD (UNIX - Berkeley Software Distribution) optimisée pour les systèmes embarqués
- rapides

Stockage, Rendu graphique, multimédia

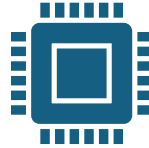
- SQLite : un SGBD simple et puissant accessible pour toutes les
- API Média basée sur OpenCore de PacketVidéo (mpeg4,
- SGL/OpenGL ES, bibliothèques 2D et 3D
- LibWebCore (moteur de navigation web utilisé)

Gestionnaire de surfaces (Surface Manager) : Pour créer des interfaces visuelles

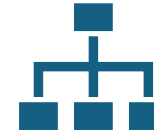
Composant : Framework Applicatif



Architecture conçue pour faciliter la réutilisation



Il fournit des API permettant aux développeurs de créer des applications riches.



Chaque application est basée sur des services et systèmes :

Gestionnaire d'activités (Activity Manager) : gère le cycle de vie des applis et l'enchaînement des vues

Gestionnaire des ressources (Resources Manager) : accès aux ressources statiques (images, fichiers, ...)

Gestionnaire de fenêtres (Window Manager) : gère les fenêtres des applications; quelle fenêtre doit être affichée devant une autre à l'écran.

Mécanisme des vues (View System) : fournit tous les composants graphiques.

...

Composant : Android Runtime



DALVIK Virtual Machine (avant Lollipop)

Dalvik : ville en islande

VM optimisée pour les systèmes embarqués

VM à registres avec un jeu important d'instructions

Conversion du bytecode : `.class` → `.dex`

Exécute un fichier `.dex` généré par dx tool

Une instance par processus



Un compilateur (ART) de bytecode vers le natif Android Runtime (à partir de Lollipop).

Architecture Android

Android est conçu par des développeurs pour des développeurs.

Android nécessite aucune certification pour devenir développeur.

Avec **Android**, on peut distribuer et monétiser des applications.

Puissant et intuitif, **Android** facilite le développement mobile.

Ouvert, un SDK simple et puissant. L'absence de coût de licence attire plus de développeur.

Facilité d'accès au matériel de bas niveau avec une série d'API.

Android fonctionne sur plusieurs marques de Smartphones.

Intégration de Google Maps et utilisation de services d'arrière plan.

Android et JAVA

Android ressemble à JAVA mais l'en est pas. Les applications sont écrites par JAVA mais ne sont pas interprétés par la machine virtuelle Java ME.

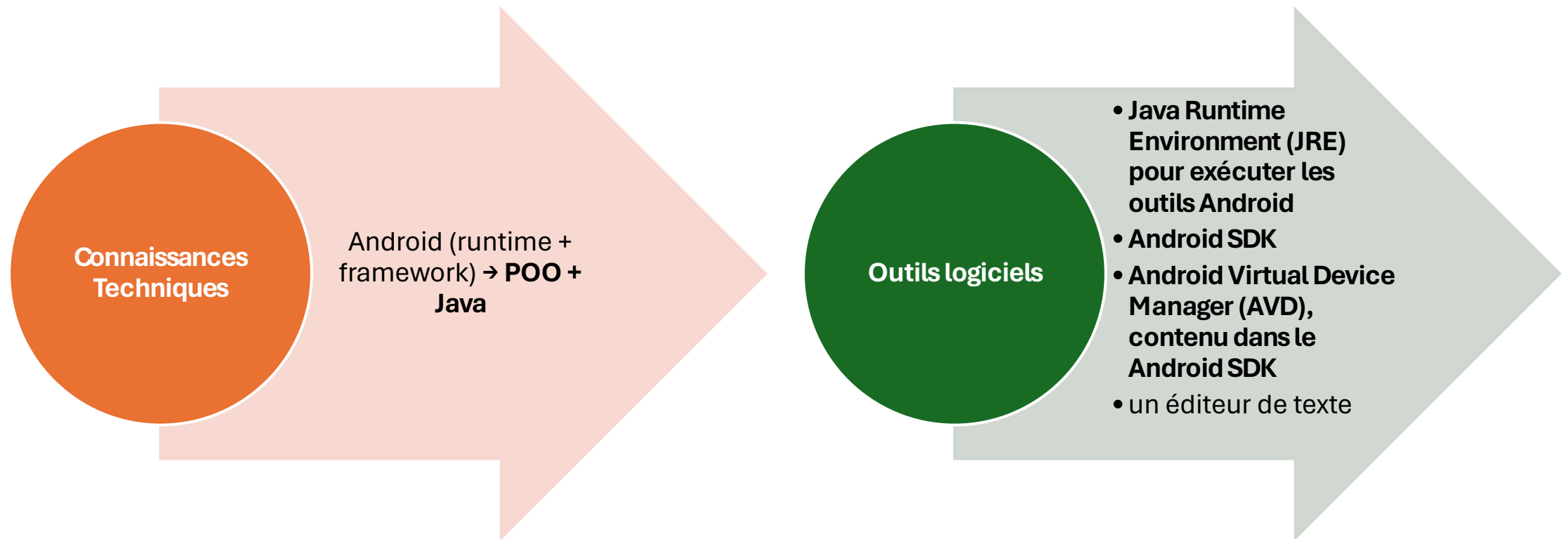
Les applications Java ne fonctionnent pas nativement sous **Android**.

Les applications **Android** sont exécutées par une machine virtuelle spécifique Dalvik (ART) et non par une JVM classique.

Android inclus une implémentation Open source spécifique de java basée non pas sur **OpenJdk**, mais sur le **projet Harmony** (implémentation java 5 de la fondation apache).

Google a adapté **Harmony** et supprimé certains packages non nécessaires en environnement mobile.

Créer une application Android



Android SDK

Le kit de développement Android (**SDK**) fournit l'environnement de travail pour développer, tester et déboguer des applications Android.



Dans le SDK on trouve:

Composés de bibliothèques d'API Android, les API Android qui sont le cœur du SDK.

Des outils de développement qui permettent de compiler et déboguer vos applications.

Le Virtual Device Manager et l'Emulateur qui fournit un meilleur environnement de test.

Des exemples de code et un support en ligne.

Développer sous Android


- **Application Android**, en pratique on utilise deux solutions :
 - ~~Android SDK + Eclipse~~
 - **Android Studio**



<https://developer.android.com/studio/index.html?authuser=2>

Création Projet

Create New Project

 **New Project**
Android Studio

Configure your new project

Application name:

Company Domain:

Package name: [Edit](#)

Project location: ...

Annotations:

- Type your Android Studio application name here.
- Type your company domain here.
- The package name for the app will be automatically generated.

Buttons: Cancel Previous Next Finish

Choix version cible

Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

☒ Phone and Tablet

Minimum SDK: API 15: Android 4.0.3 (IceCreamSandwich)

Lower API levels target more devices, but have fewer features available. By targeting API 15 and later, your app will run on approximately **94,0%** of the devices that are active on the Google Play Store.

[Help me choose](#)

☐ Wear

Minimum SDK: API 21: Android 5.0 (Lollipop)

☐ TV

Minimum SDK: API 21: Android 5.0 (Lollipop)

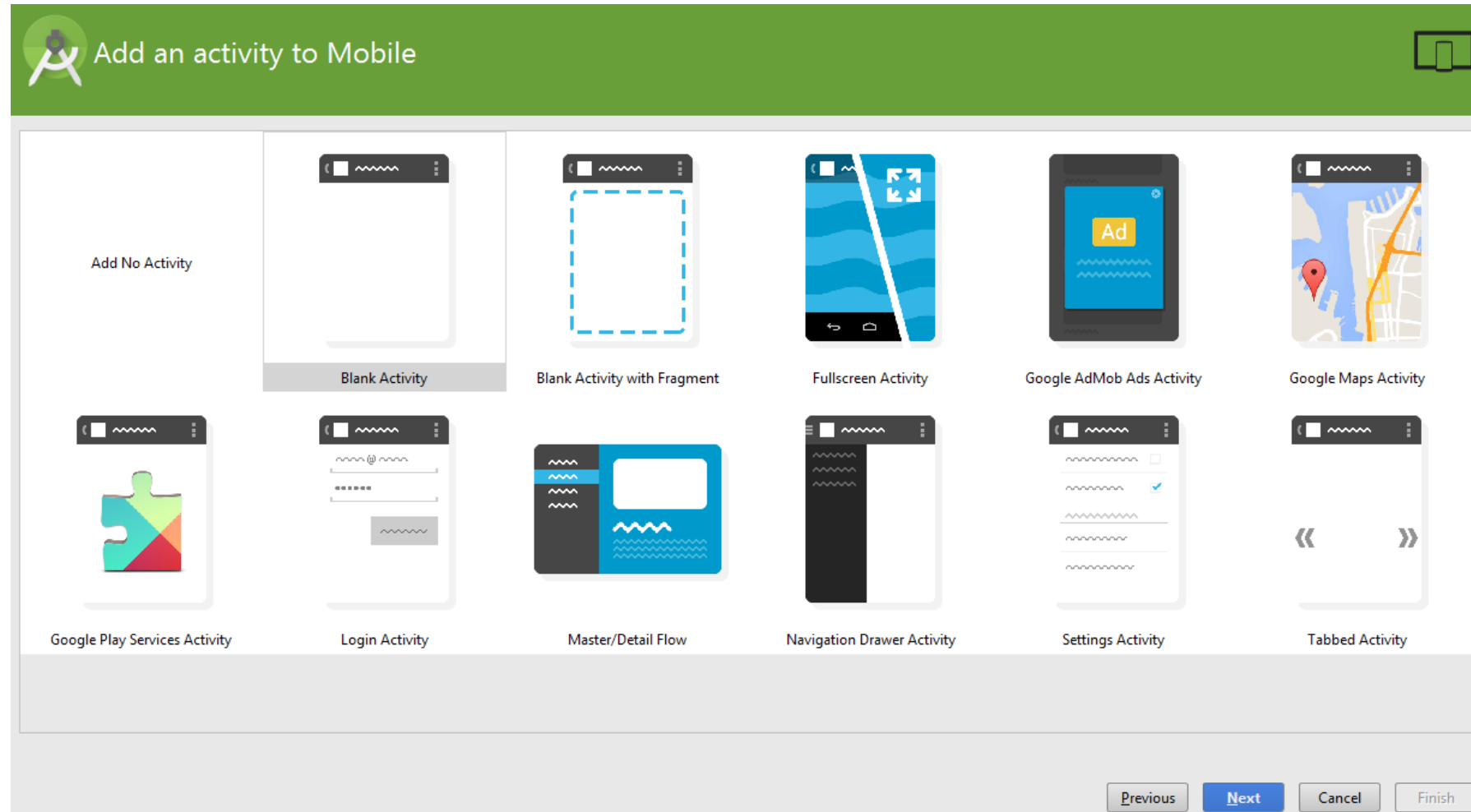
☐ Android Auto

☐ Glass (Not Installed) [Download](#)



Minimum SDK:

[Previous](#) [Next](#) [Cancel](#) [Finish](#)

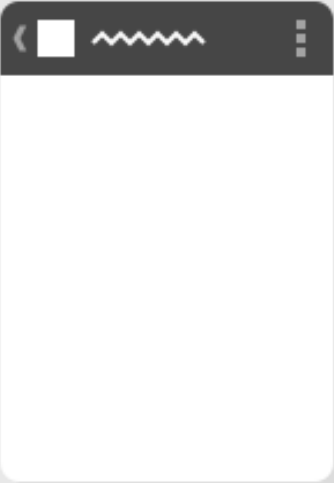
Ajout d'une *Activity*



Configuration de l'*Activity*

 Customize the Activity 

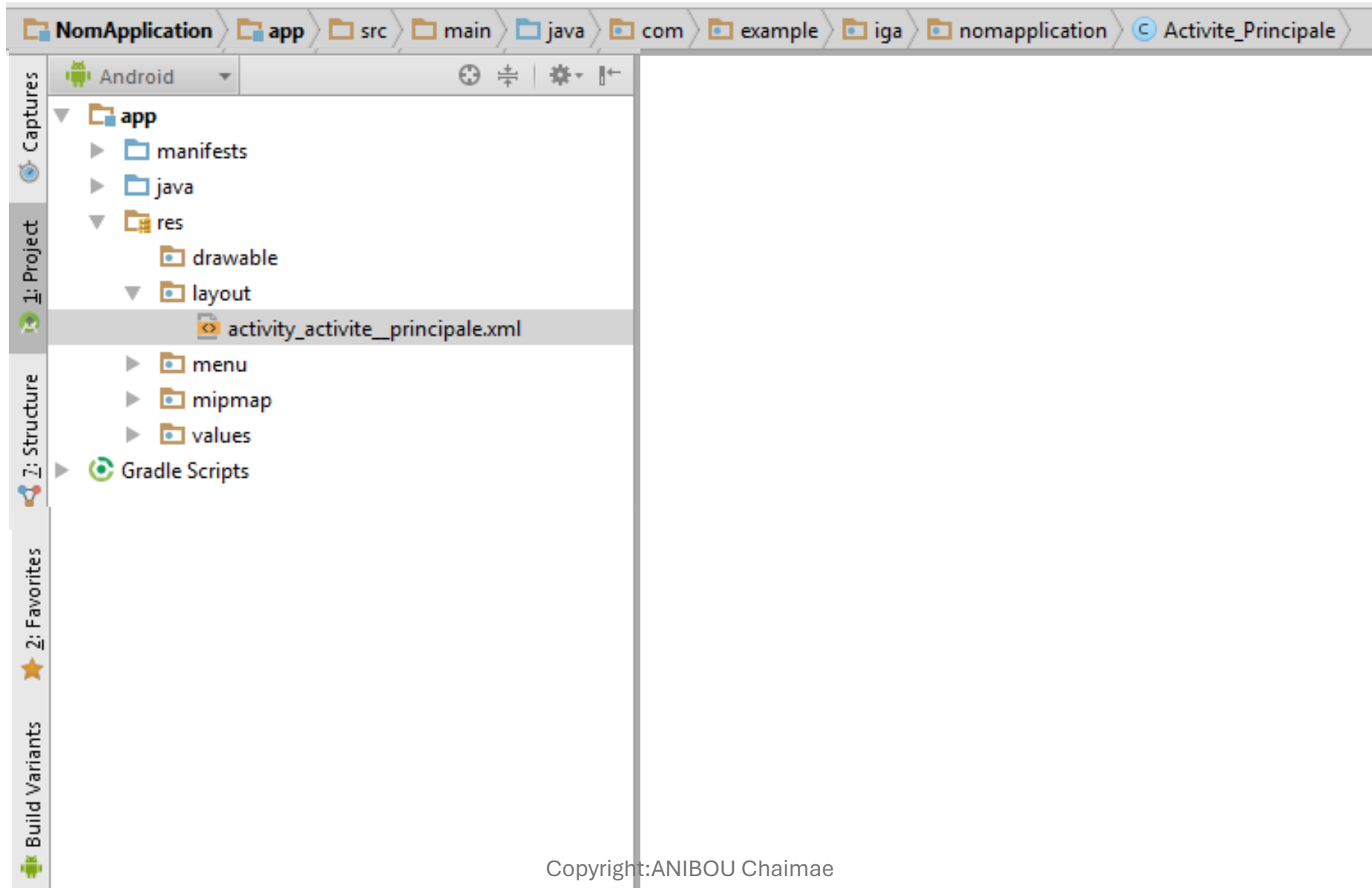
Creates a new blank activity with an action bar.



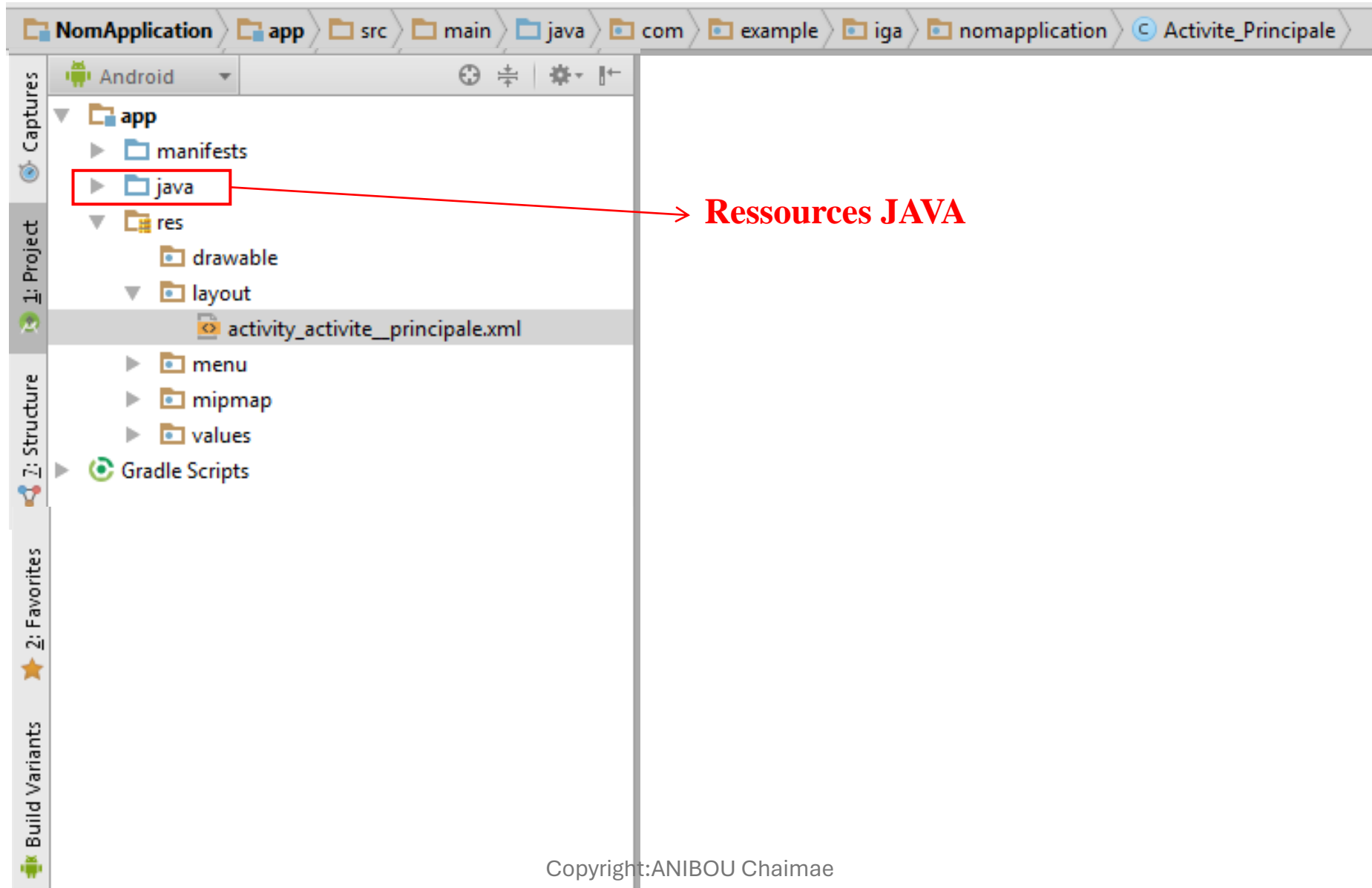
Activity Name:	<input type="text" value="Activite_Principale"/>
Layout Name:	<input type="text" value="activity_activite_principale"/>
Title:	<input type="text" value="L'Activite Principale de l'application"/>
Menu Resource Name:	<input type="text" value="menu_activite_principale"/>

The name of the activity. For launcher activities, the application title.

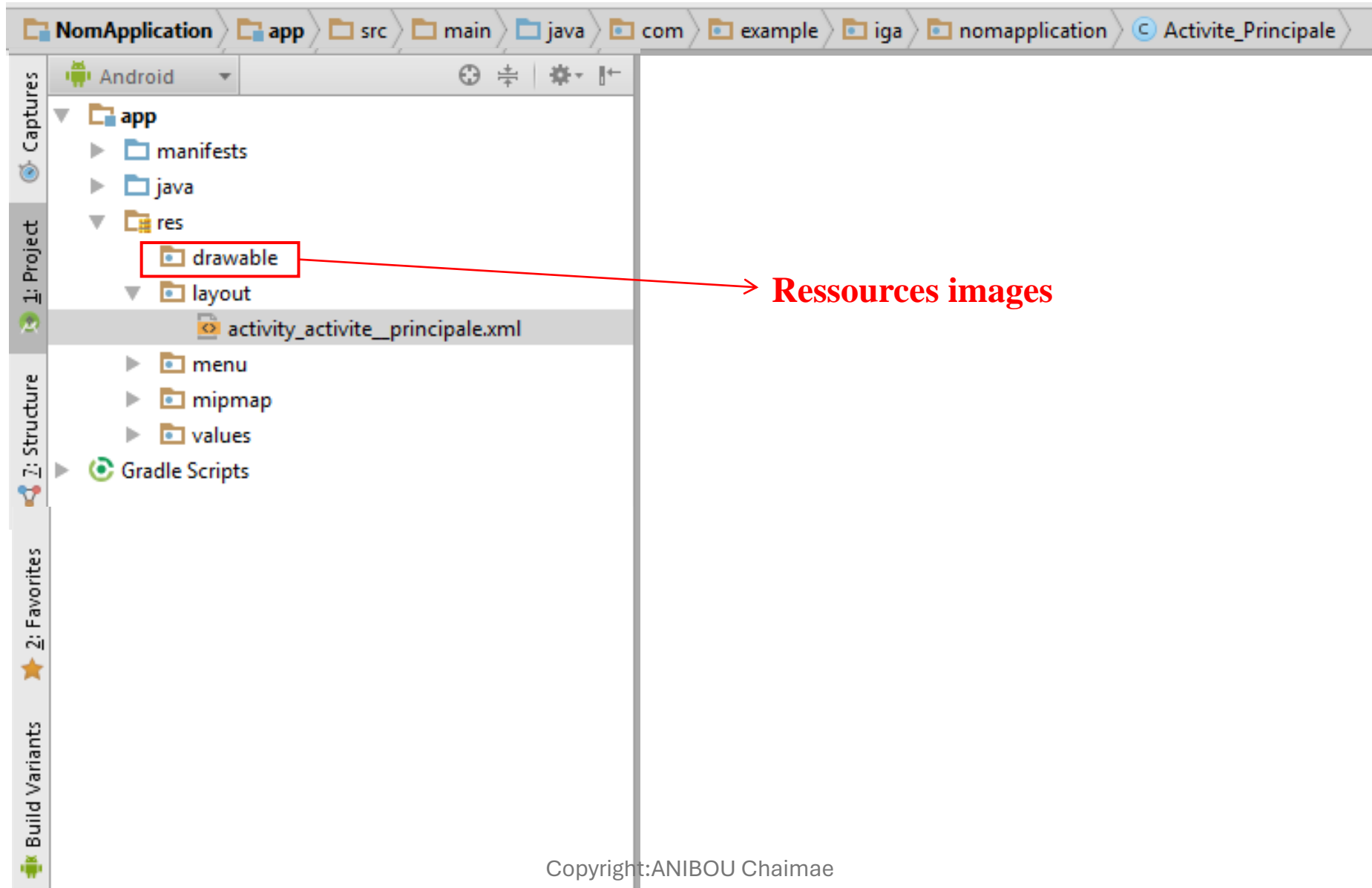
Arborescence du projet



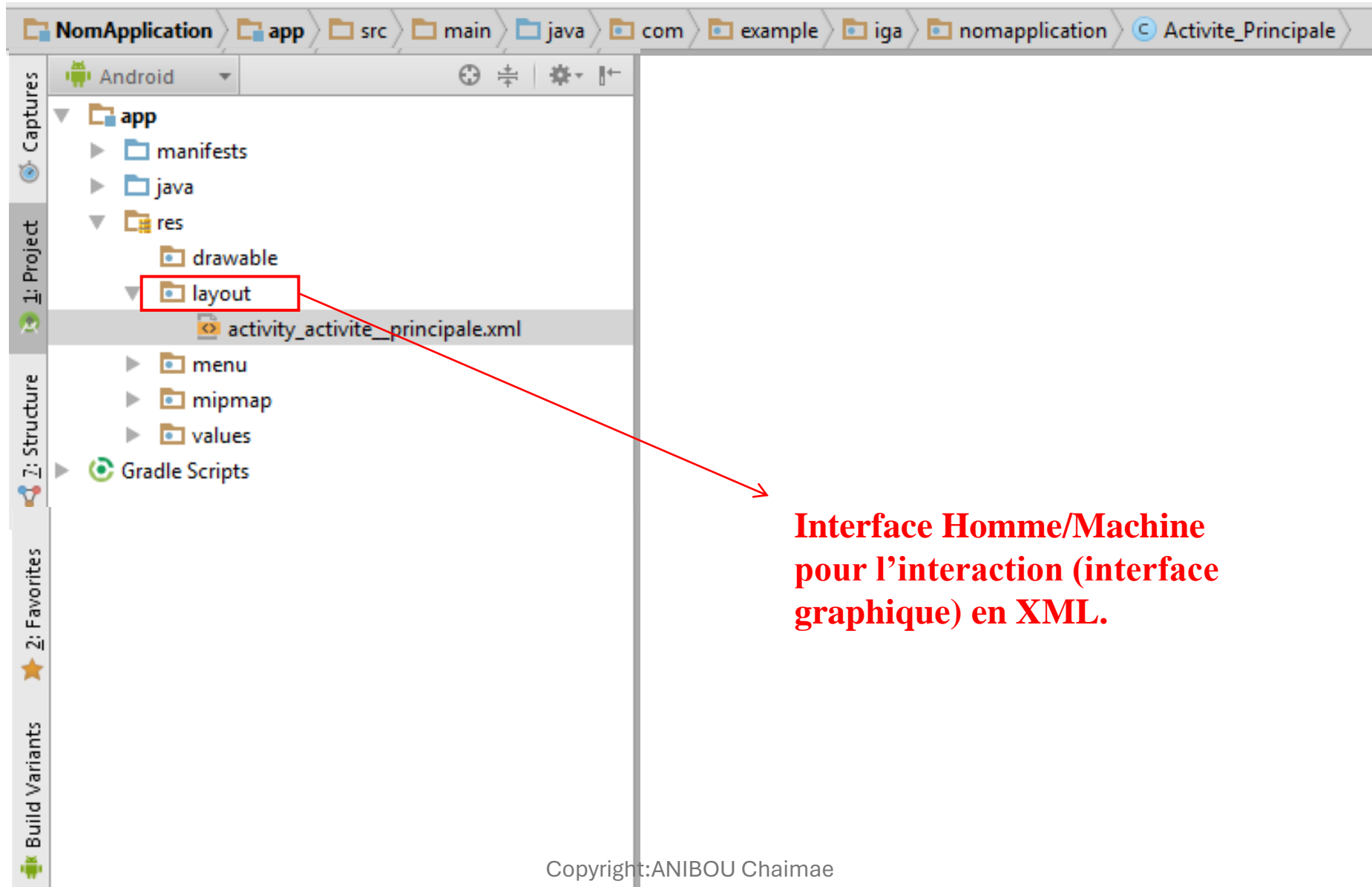
Arborescence du projet



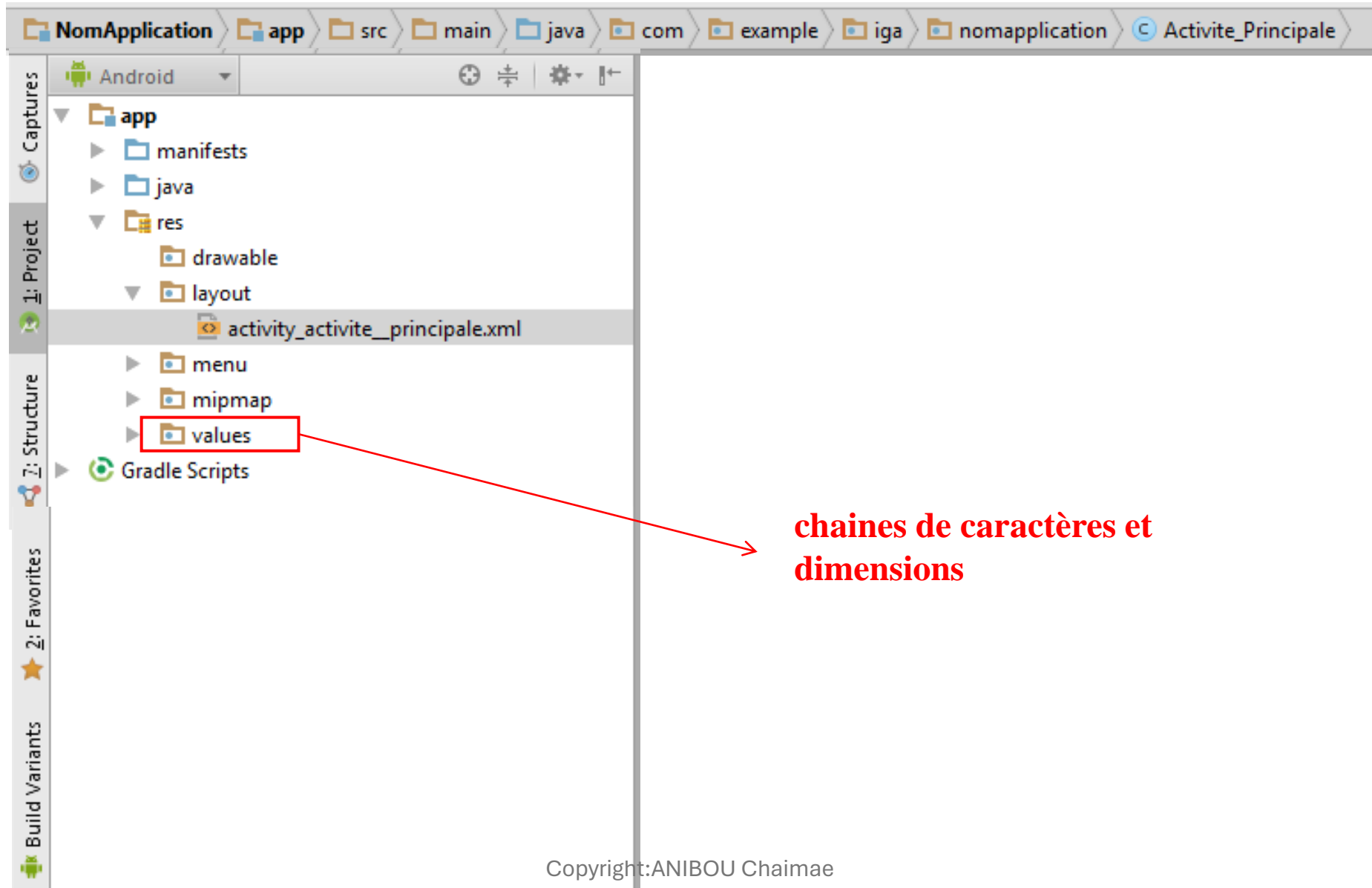
Arborescence du projet



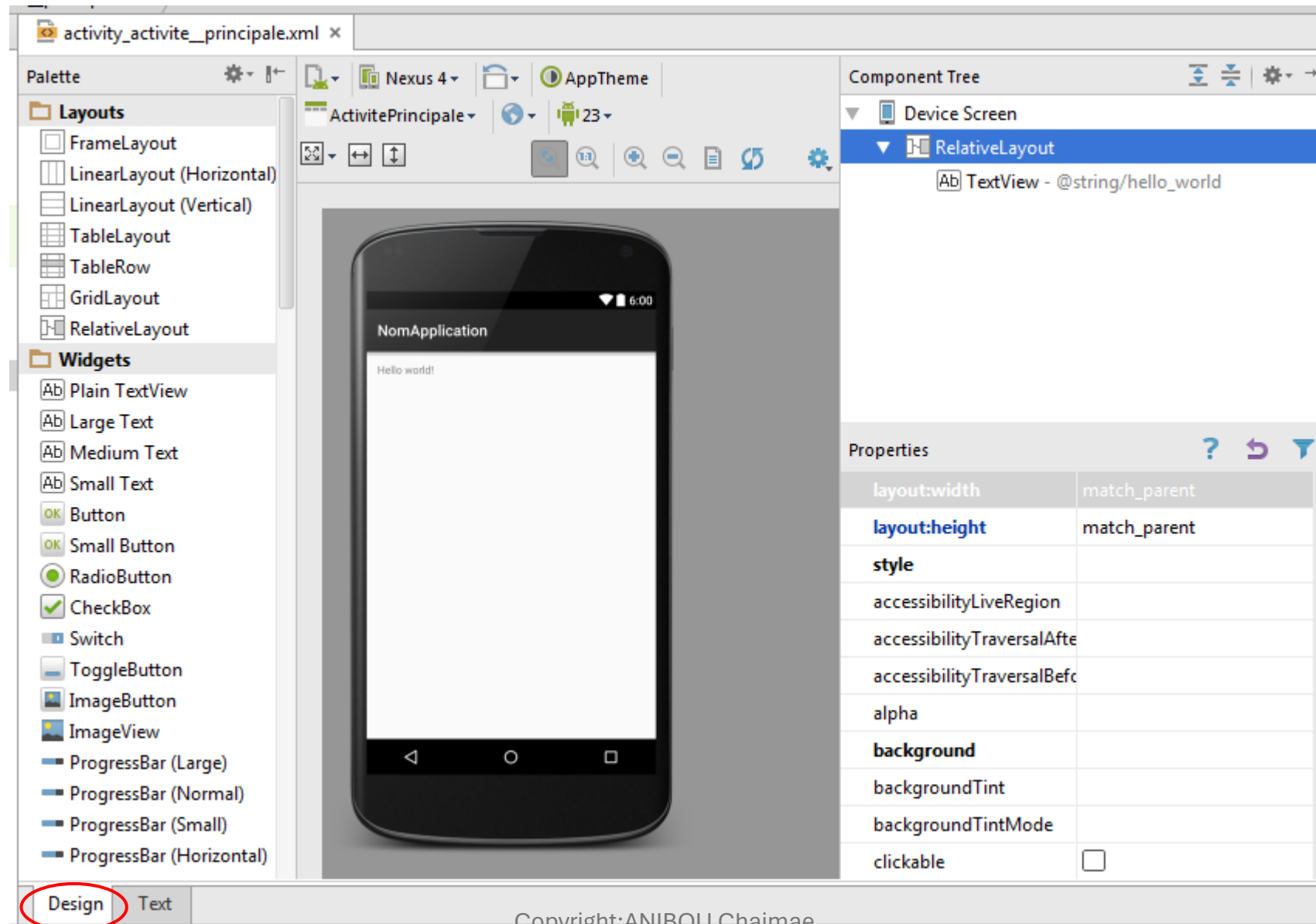
Arborescence du projet



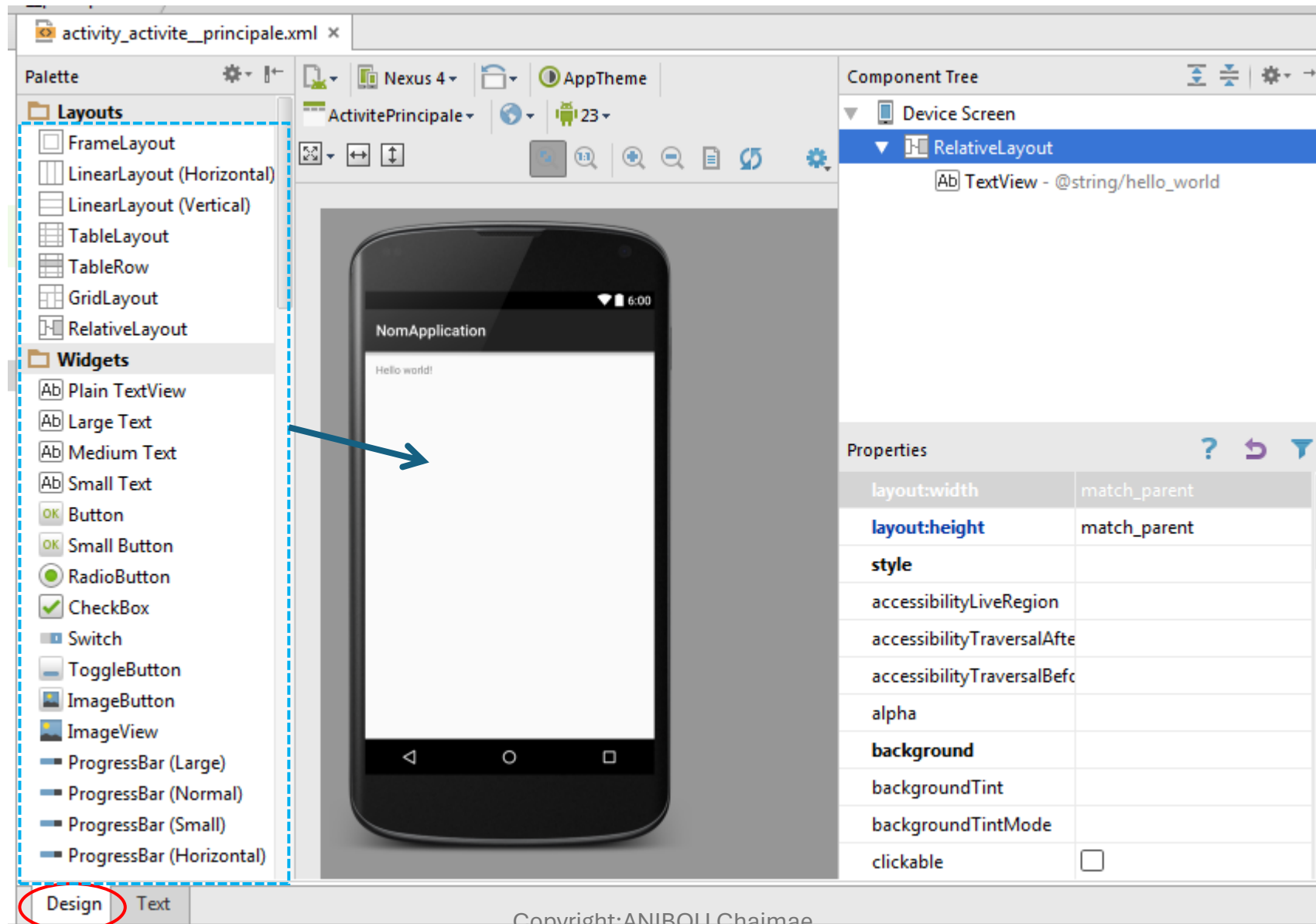
Arborescence du projet



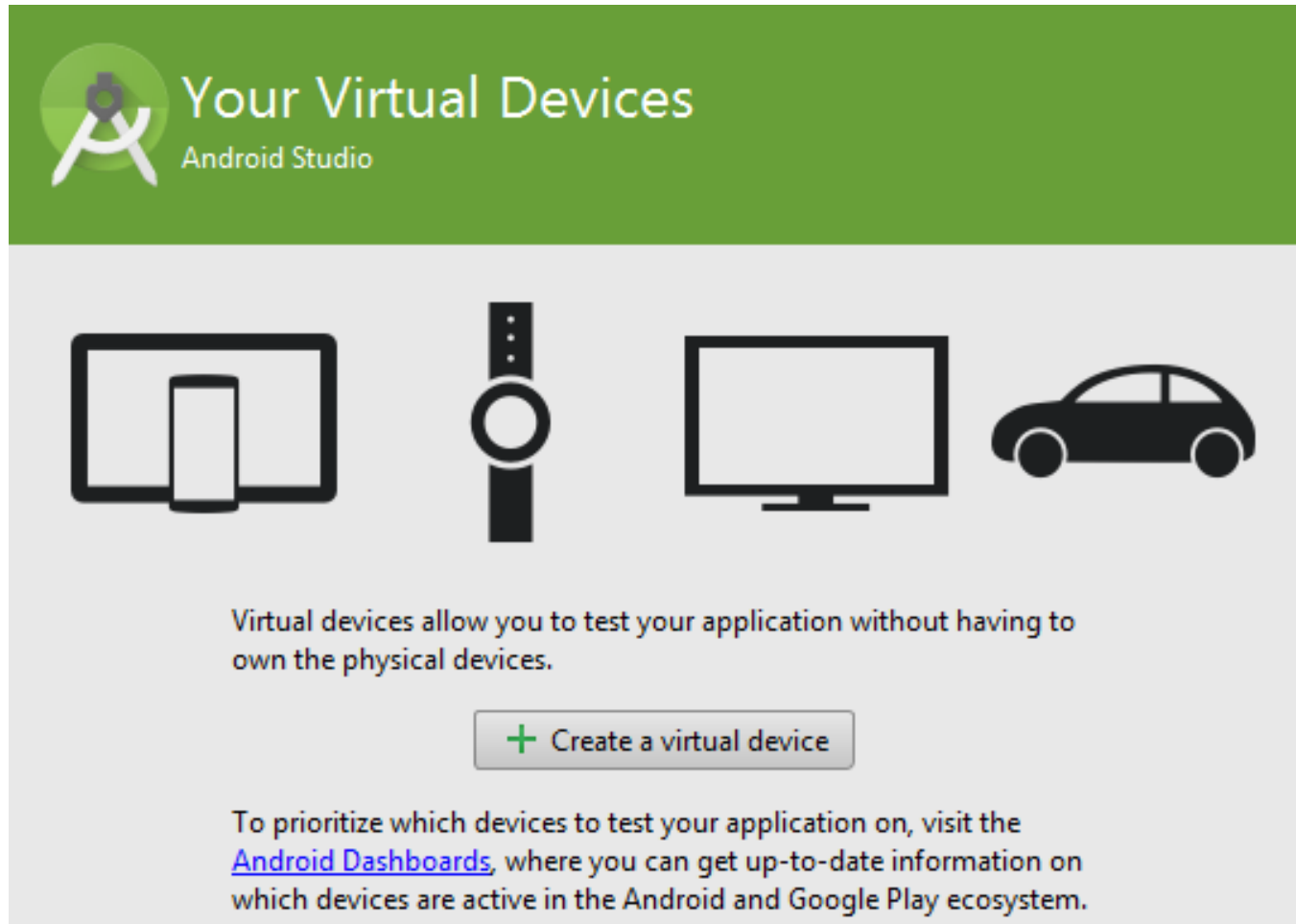
Palette de création



Palette de création

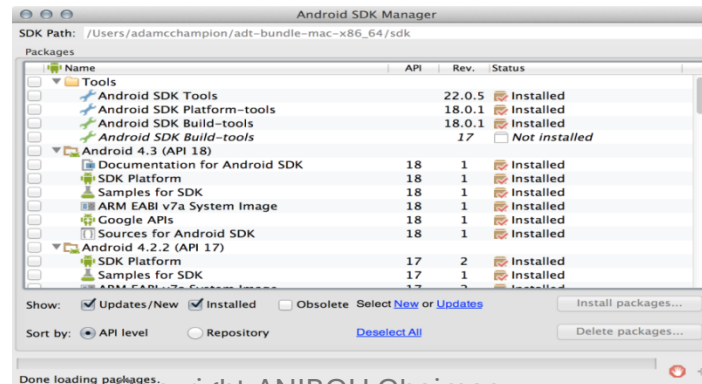


Android Virtual Devices

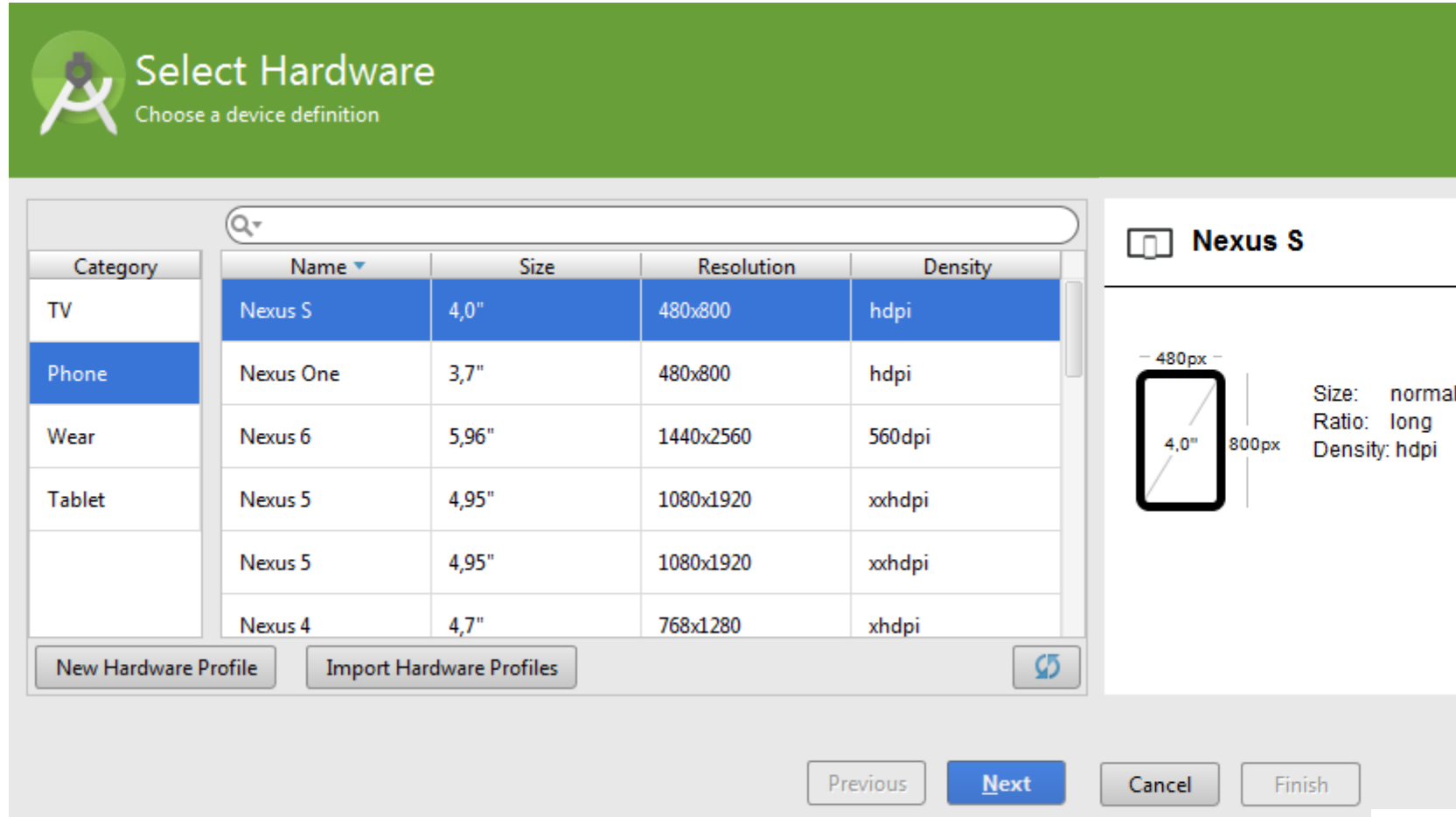


SDK Manager (Tools > Android)

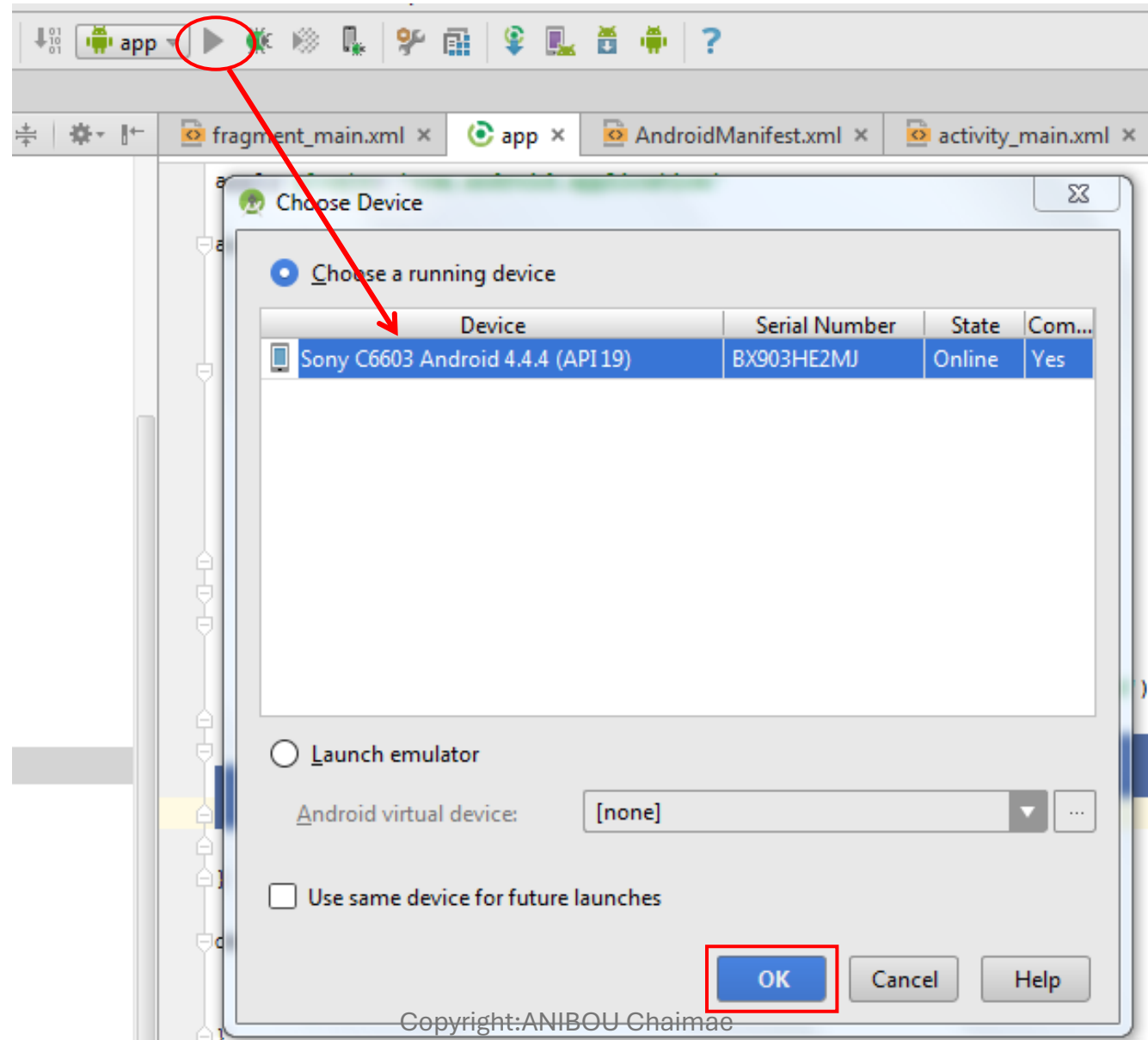
- Gère les versions du SDK d'Android installés
- Gère les images des virtuals devices
 - Pour ARM, pour INTEL, pour chaque version d'Android
- Gère les sources des différents SDK
- Gère les sources des exemples



Android Virtual Devices



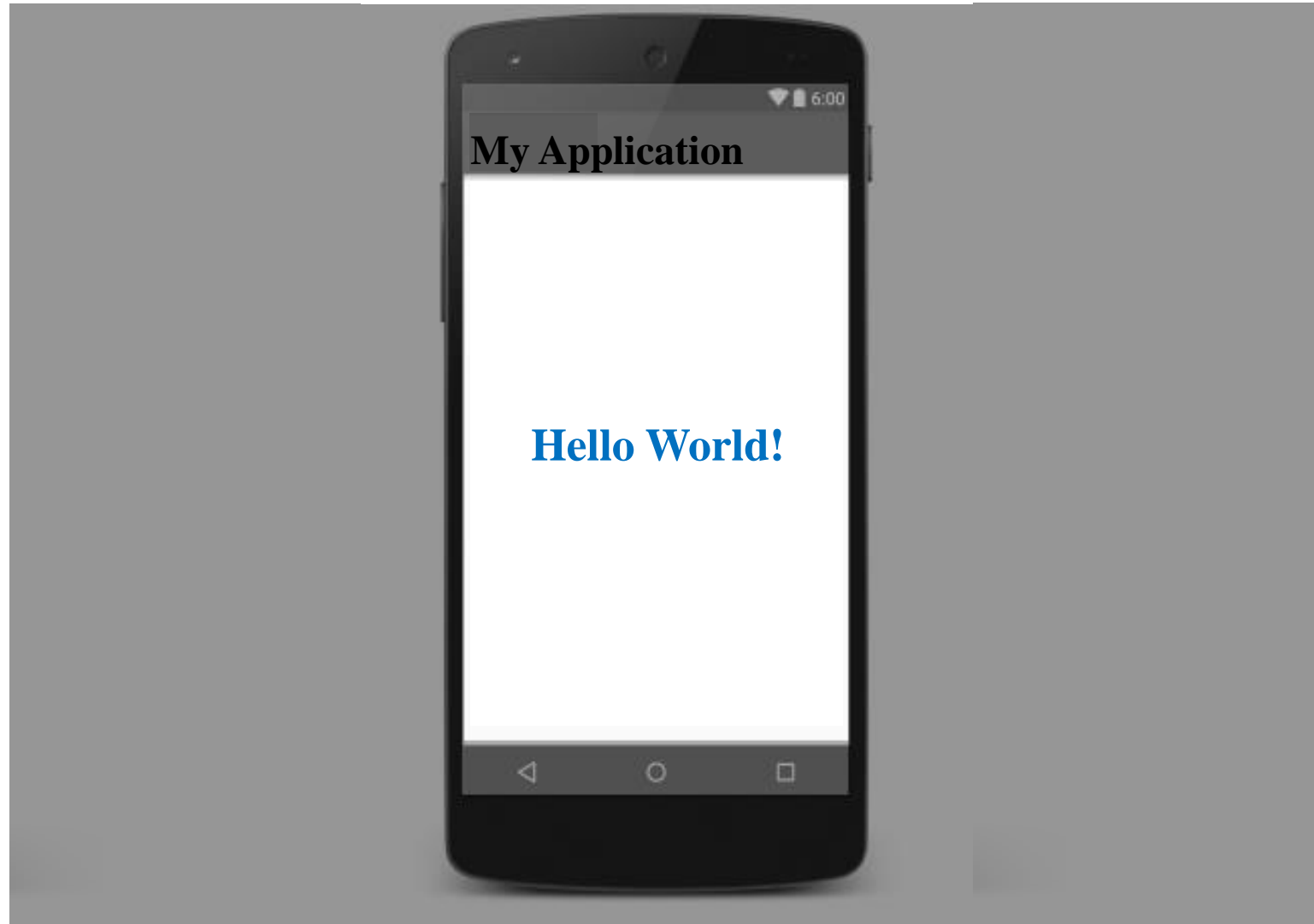
Android Virtual Devices



Émulateur Android



Première Application



Structure d'une application Android

Une application Android est composée de deux parties :

Une série de fichier XML décrivant

l'application (version de l'API utilisée, activité principale, etc.)

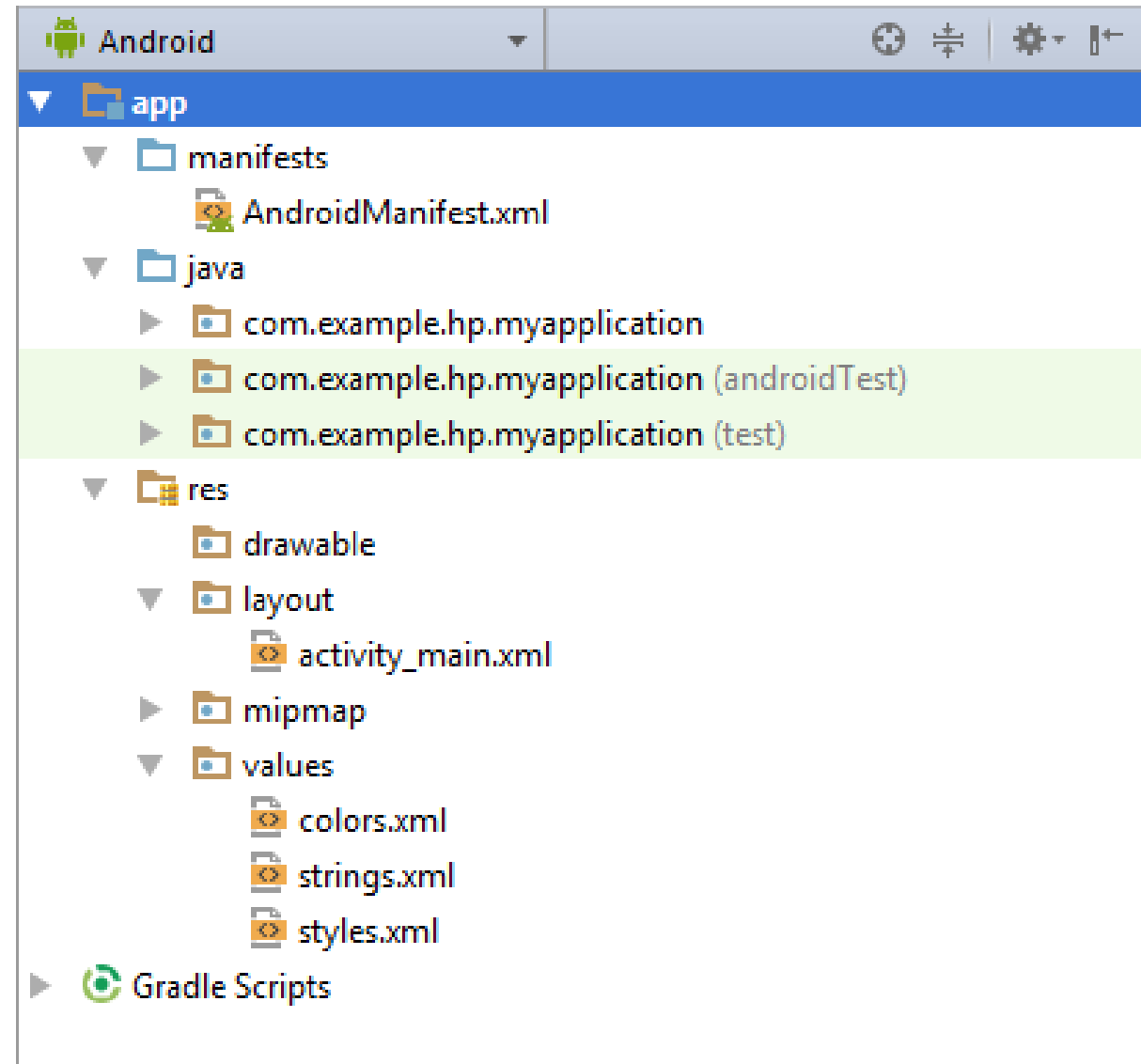
les ressources (texte, image, **layout**, etc.)

du code Java pour spécifier la partie dynamique de l'application

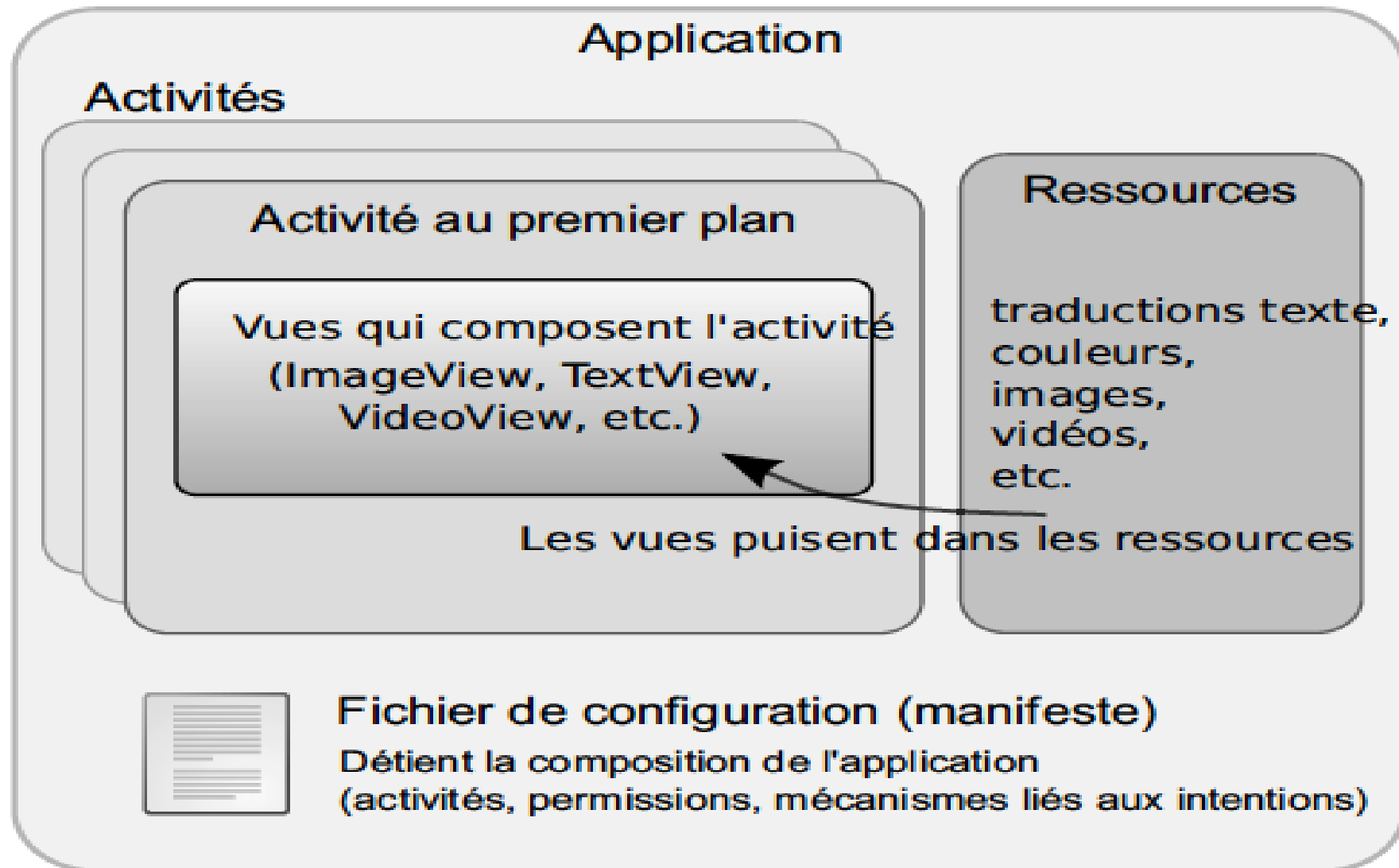
Les fichiers XML sont plus ou moins obligatoire

Structure d'un projet

- **AndroidManifest.xml** : décrit les propriétés de l'application
- **java/** : contient les fichiers source
- **res/** : contient les ressources (images, descriptions d'interfaces, valeurs) . Organisé en sous dossiers par types (drawable, layout, mipmap, values, ...)
- **Gradle Scripts** : Scripts de compilation



Application Android : Composition

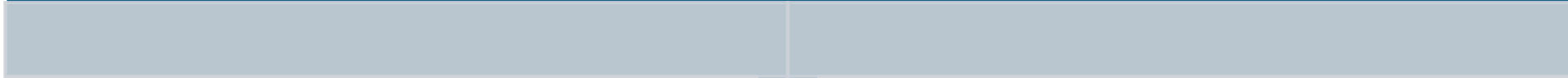


Application Android : Fichier de configuration

C'est un fichier indispensable à chaque application qui décrit entre autres :



Le point d'entrée de votre application (quel code doit être exécuté au démarrage de l'application);



Quels composants constituent ce programme ;



Les permissions nécessaires à l'exécution du programme (accès à Internet, accès à l'appareil photo...).

Exemple : AndroidManifest.xml

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <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>
```

LES vues





Définition



Les éléments de l'interface graphique que l'utilisateur voit et sur lesquels il pourra agir.



Les vues contiennent des composants,
organisés selon diverses mises en page :

les uns à la suite des autres,
en grille,
...

LES ACTIVITES



Développer une application Android

Contrairement à une application classique, une application Android est découpée en écran

Le système Android contrôle les activités, pas le développeur

Une activité est une classe qui hérite de *Activity* ou d'une classe dérivée de *Activity*

Chaque écran est géré par une activité (*Activity*)

Aucune méthode *main* dans un programme Android

Le code d'une activité réagit à des événements du système Android

État d'une activité

Une seule activité visible à l'écran au premier plan

Une activité peut se trouver dans 3 états qui se différencient surtout par leur visibilité :

Active (resumed) : elle est visible en totalité.

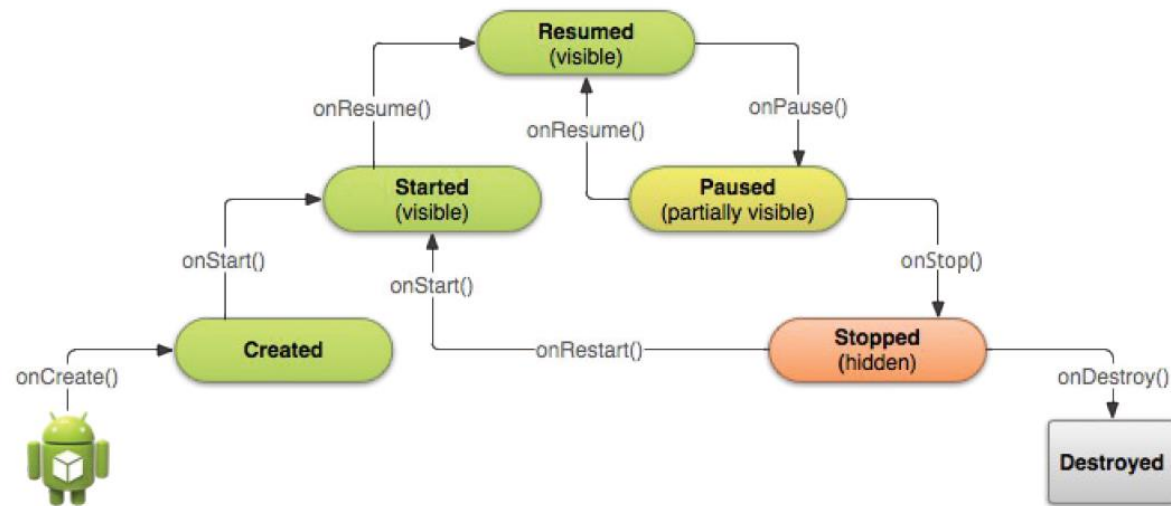
Suspendue (paused) : elle est partiellement visible à l'écran (c'est le cas quand vous recevez un SMS).

Arrêtée (stopped) : elle est tout simplement oblitérée par une autre activité, on ne peut plus la voir du tout.

- Passage de l'état visible à invisible :

- En lançant une nouvelle activité masquant l'activité courante/ En détruisant l'activité courante (finish(), bouton "retour")

Cycle de vie d'une activité



- L'activité réagit à la demande du système Android (qui lui-même réagit à la demande de l'utilisateur)

Activity.onXXX()

Les méthodes appelées lors des changements d'état :

- **onCreate(Bundle savedInstanceState)** : appelée à la création. Le paramètre permet de récupérer un état sauvegardé lors de l'arrêt de l'activité (si on a fait une sauvegarde)
 - **onStart()** : appelée quand l'activité démarre
 - **onRestart()** : appelée quand l'activité redémarre
 - **onResume()** : appelée quand l'activité vient en premier plan
 - **onPause()** : appelée quand l'activité n'est plus en premier plan
 - **onStop()** : appelée quand l'activité n'est plus visible
 - **onDestroy()** : appelée quand l'activité se termine
-
- On redéfinit les méthodes que l'on veut et on oublie pas d'appeler en premier **super.onXXX()**

Les Ressources



Architecture d'Android

Le répertoire `res/` contient toutes les ressources qui seront mises dans le fichier application (apk). Il est constitué de sous répertoires :

drawable (images)

layout (description en XML des interfaces)

values (définitions en XML de constantes (avec `i18n`) : string, color, dim, style...)

anim (description en XML d'animations)

menus (description en XML de menus pour l'application)

xml (fichiers XML utilisés directement par l'application)

raw (tous les autres types de ressources : sons, vidéos, ...)

On peut ajouter d'autres sous répertoires

Exemple : internationalization (i18n)

- Valeur par défaut:

dans res/**values**/string.xml

```
<resources>  
  <string name="helloworld">Hello World</string>  
</resources>
```

- Valeur pour le francais:

dans res/**values-fr**/string.xml

```
<resources>  
  <string name="helloworld">Bonjour monde</string>  
</resources>
```

La classe R

C'est une classe générée par Android Studio qui permet à l'application d'accéder aux ressources

Elle contient des classes internes dont les noms correspondent aux types de ressources (id, drawable, layout ...)

Elle est constituée à partir des fichiers placés dans les sous répertoires du répertoire **res/**

Une propriété est créée pour :

- Chaque image placée dans drawable
- Chaque identificateur défini dans des fichiers XML (objets d'interface, constantes)
- Chaque fichier placé dans les répertoires xml , raw ...

Référence aux ressources

- Une ressource dans un fichier XML d'une autre ressource:
@[paquetage:]type/nomDeLaRessource
par exemple **@string/helloworld**
- Ressources **système** en utilisant le paquetage android
(par ex. **@android:color/blue**)
- Dans un fichier source Java, par une constante (int) générée dans le fichier R.java, par ex. **R.string.helloworld**
- Utilisation de Resources context.getResources()
(Activity hérite de Context) pour obtenir un getter de ressources
- Quelques exemples :
R.id.loginButton, getString(R.string.helloworld),
getColor(R.color.my_nice_color), getLayout(R.layout.activity_layout),
getDimension(R.dimen.world_width),
getDrawable(R.drawable.card_picture),
openRawResource(R.raw.bindata) (retourne un InputStream)

Les composants graphiques



Vues et gabarits

Les éléments graphiques héritent de la classe **View**. On peut regrouper des éléments graphiques dans une **ViewGroup**. Des **ViewGroup** particuliers sont prédéfinis: ce sont des gabarits (**layout**) qui proposent une prédispositions des objets graphiques:

- **LinearLayout** : dispose les éléments de gauche à droite ou du haut vers le bas
- **RelativeLayout** : les éléments enfants sont placés les uns par rapport aux autres
- **TableLayout** : disposition matricielle
- **FrameLayout** : disposition en haut à gauche en empilant les éléments
- **GridLayout** : disposition matricielle avec N colonnes et un nombre infini de lignes

Les déclarations se font principalement en XML, ce qui évite de passer par les instantiations Java.

Attributs des gabarits

Les attributs des gabarits permettent de spécifier des attributs supplémentaires. Les plus importants sont:

- **android:layout_width** et **android:layout_height** :
 - **"match_parent"** : l'élément remplit tout l'élément parent
 - **"wrap_content"** : prend la place minimum nécessaire à l'affichage
 - **"fill_parent"** : comme match_parent (deprecated, API<8)
- **android:orientation** : définit l'orientation d'empilement
- **android:gravity** : définit l'alignement des éléments

Exemple : LinearLayout

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout_width="match_parent"

android:layout_height="match_parent"

android:gravity="center"

android:id="@+id/accueilid"

>

</LinearLayout>

Exemples : ViewGroup

Un **ViewGroup** contient des éléments graphiques. Pour construire un gabarit linéaire on fera :

<LinearLayout

```
xmlns:android="http://schemas.android.com/apk/res/android"  
android:layout_width="match_parent"  
android:layout_height="match_parent"  
android:gravity="center"  
android:id="@+id/accueilid"  
>
```

<TextView

```
android:id="@+id/le_texte" android:text="@string/hello"  
/>
```


<TextView

```
android:id="@+id/le_texte2" android:text="@string/hello2"  
/>
```

</LinearLayout>

View en Java

Depuis une activité :

- Activity.setContentView(int) 
demande l'affichage de la hierarchie à partir d'un layout/foo.xml
- View context.findViewById(int)
demande d'un composant **après** création hiérarchie

```
public class MainActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        EditText editText = (EditText)findViewById(R.layout.name);  
        ...  
    }  
}
```

View basique

Éléments de formulaire

- TextView : affiche une chaîne
- EditText : permet la saisie d'une chaîne (propriété `inputType` pour le type d'entrée attendu)
- Button : bouton cliquable, variante de type interrupteur avec `ToggleButton`
- CheckBox : case à cocher
- RadioButton : bouton radio regroupable dans un `RadioGroup`
- CheckedTextView : chaîne cochable (implante `Checkable`)
- ProgressBar : barre de progression (horizontale, circulaire), variante avec étoiles de notation avec `RatingBar`
- SeekBar : barre de réglage
- SearchView : champ de recherche avec proposition de suggestions

Éléments multimédias

- ImageView : affichage d'une ressource image
- ImageButton : bouton avec image
- VideoView : affichage contrôlable de vidéo

EditText

Champs texte editable par l'utilisateur

attributs XML les + fréquents:

- android:id, identifiant pour un findViewById dans l'Activity
- android:ems, taille en caractère (adapté à la fonte)
- android:text, texte
- android:hint, conseil qui sera affiché si pas de texte
- android:inputType, type valeur attendu, textPassword, textEmailAddress, date, time, phone

exemple,

```
<EditText  
    android:id="@+id/password"  
    android:ems="16"  
    android:hint="password"  
    android:inputType="textPassword"/>
```



Développement Mobile Natif

- Le développement mobile natif consiste à créer des applications spécifiquement pour une plateforme particulière, en utilisant les langages de programmation et les outils recommandés par le fournisseur de la plateforme.
- Pour Android, le développement natif implique généralement l'utilisation de Java ou Kotlin avec le kit de développement logiciel Android (Android SDK) et l'environnement de développement intégré (IDE) Android Studio.
- Les avantages du développement natif incluent des performances optimisées, un accès complet aux fonctionnalités de la plateforme et une intégration étroite avec l'écosystème existant de l'appareil.



Développement Mobile Hybride

- Le développement mobile hybride est une approche de développement d'applications mobiles qui combine des technologies web telles que HTML, CSS et JavaScript avec des capacités natives. Les principales caractéristiques du développement mobile hybride sont :
 - - Utilisation de technologies web standard : Les développeurs utilisent HTML, CSS et JavaScript pour créer des interfaces utilisateur et des fonctionnalités d'application.
 - - Conteneur natif : L'application est encapsulée dans un conteneur natif qui permet d'accéder aux fonctionnalités natives du périphérique, telles que l'appareil photo ou le GPS.
 - - Déploiement multiplateforme : Une seule base de code peut être déployée sur plusieurs plateformes, telles qu'Android et iOS, ce qui permet d'économiser du temps et des ressources de développement.
 - - Flexibilité : Les applications hybrides peuvent être mises à jour rapidement en temps réel sans passer par un processus de validation d'app store.



Approche Web dans le Développement Mobile

- L'approche web dans le développement mobile consiste à créer des applications mobiles en utilisant des technologies web standard telles que HTML, CSS et JavaScript. Voici quelques points clés de cette approche :
- - Développement multiplateforme : Les applications web mobiles peuvent être utilisées sur différents appareils et systèmes d'exploitation sans nécessiter de développement spécifique pour chaque plateforme.
- - Accessibilité : Les applications web mobiles sont généralement accessibles via un navigateur web, ce qui les rend facilement disponibles pour les utilisateurs sans nécessiter d'installation depuis un App Store.
- - Mises à jour rapides : Les mises à jour des applications web mobiles peuvent être déployées instantanément sur les serveurs, ce qui permet aux utilisateurs d'accéder aux dernières fonctionnalités sans avoir à mettre à jour l'application sur leur appareil.
- - Limitations : Les applications web mobiles peuvent présenter des limitations en termes de performances et de fonctionnalités par rapport aux applications natives.

TP : Cycle de Vie d'une Application Android et Utilisation du Logcat

- **Objectif :** Comprendre le cycle de vie d'une application Android et apprendre à utiliser le Logcat pour le débogage.

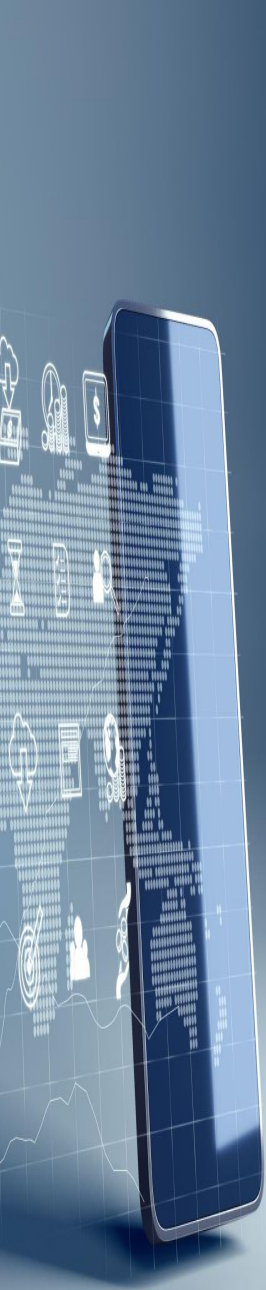
Instructions :

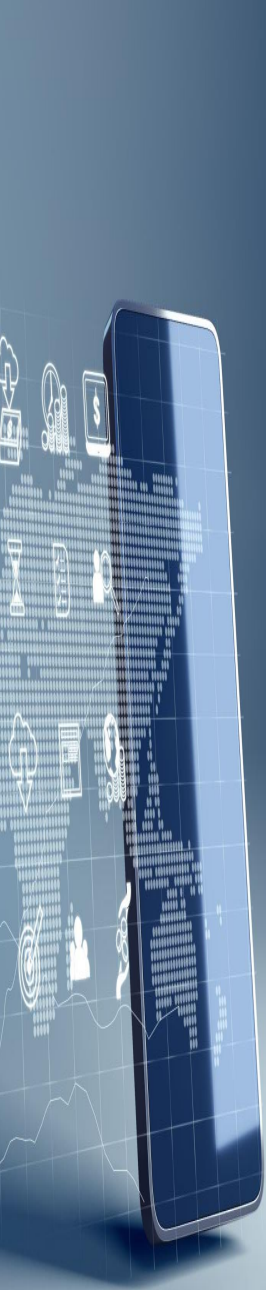
Création d'une Application :

- Créez une nouvelle application Android simple à l'aide d'Android Studio.
- Ajoutez des méthodes de journalisation (`Log.d()`, `Log.e()`, `Log.i()`, `Log.V()`, `Log.W()` etc.) à différentes parties du cycle de vie de votre application, en commençant par `onCreate()`, et en couvrant les autres états importants.
- Utilisez les messages de journalisation pour suivre le flux d'exécution de votre application et pour comprendre quand chaque méthode est appelée.

Débogage avec Logcat :

- Ouvrez l'outil Logcat dans Android Studio (Généralement situé dans la barre d'outils en bas de l'écran).
- Exécutez votre application sur un émulateur ou un appareil Android connecté.
- Observez les messages de journalisation générés par votre application dans le Logcat.
- Comprenez les différents types de messages (debug, error, warning, etc.) et apprenez à les interpréter. Par exemple, les messages de type "debug" sont généralement utilisés pour les informations de débogage, tandis que les messages de type "error" indiquent une erreur dans le code.
- Utilisez les filtres Logcat pour afficher uniquement les messages liés à votre application, ce qui facilite le





TP : Cycle de Vie d'une Application Android et Utilisation du Logcat

- **Objectif :** Comprendre le cycle de vie d'une application Android et apprendre à utiliser le Logcat pour le débogage.

Instructions :

Test et Observation :

- Exécutez votre application et observez les messages de journalisation dans le Logcat.
- Identifiez les messages associés à des événements spécifiques, tels que le lancement de l'application, la rotation de l'écran, la mise en arrière-plan, etc.
- Utilisez les messages de journalisation pour diagnostiquer les problèmes potentiels dans votre application, tels que les exceptions non capturées, les erreurs de logique, etc

Analyse et Rapport :

- Rédigez un rapport décrivant ce que vous avez appris sur le cycle de vie des applications Android et sur l'utilisation du Logcat pour le débogage.
- Incluez des captures d'écran du Logcat avec des annotations pour illustrer vos observations.