Introduction

Systèmes d'exploitation (OS : Operating System) pour les terminaux mobiles :

- Pour les smartphones et les tablettes
- Android (Google), iOS (Apple), Windows mobile, Blackberry, etc.

Android :

- Un OS basé sur Linux, pour les smatphones et les tablettes.
- Un OS open source
- Développé par Google
- Initialement conçu pour les smartphones et tablettes tactiles, puis s'est diversifié : les télévisions (Android TV), les voitures (Android Auto), les ordinateurs (Android-x86) et les smartwatch (Android Wear).
- Le système d'exploitation mobile le plus utilisé au monde

Parts de marché des OS mobiles en 2022 :
(source : https://gs.statcounter.com/os-market-share/mobile/worldwide)

- Android 71,5%
- iOS : 27,8%
- Samsung : 0,4%
- Autres : 0,2%

Développement des applications Android :

- Développement en **Java** sous « **Android Studio** » : génération du fichier exécutable « **.apk** » (Android seuelement)
- Développement en **Kotlin** sous « **Android Studio** » : génération du fichier exécutable « **.apk** » (Android seuelement)
- Développement multi-plateformes en JavaScript avec Cordova : conversion en Java et génération du fichier exécutable « **.apk** »
    - L'exécution de l'exécutable « **.apk** » nécessite la machine virtuelle Dalvik (pour les anciennes versions du OS) ou ART (pour les nouvelles versions du OS).
- Développement natif en C++ (supporté par Android Studio)
- Développement en C# avec le framework .NET MAUI (Microsoft Visual Studio) : développement des applications multi-plateformes (Android, iOS, macOS et Windows)
- Développement avec **Flutter** (langage **Dart**) : développement des applications multi-plateformes (Android, iOS, macOS, Windows, **Linux**)

Télécharger une application Android :

- Google Play Store
- Market alternatifs (Amazon Appstore for Android, GetJar, F-Droid, Samsung Galaxy Apps, etc.)
- Téléchargement direct du fichier binaire (.apk) sur le smartphone

Google Play :

- Un magasin d'applications créé par Google (le 6 mars 2012)
- Une boutique d'applications pour le système d'exploitation Android
- En 2022, Google Play contient ~2,7 million applications (https://www.appbrain.com/stats)

Les Versions d'Android :

Version Android	**Date de sortie**	**API Level**	**Nom**
Android 13	15 Aout 2022	33	Android 13
Android 12.1	7 Mars 2022	32	Android 12L
Android 12	4 Octobre 2021	31	Android 12
Android 11	8 Septembre 2020	30	Android 11
Android 10	3 Septembre 2019	29	Android 10
Android 9.0	1 Décembre 2018	28	Pie
Android 8.1	6 Décembre 2017	27	Oreo
Android 8.0	21 août 2017	26	Oreo
Android 7.1	4 Octobre 2016	25	Nougat
Android 7.0	Aout 2016	24	Nougat
Android 6.0	Aout 2015	23	Marshmallow
Android 5.1	Mars 2015	22	Lollipop
Android 5.0	Novembre 2014	21	Lollipop
Android 4.4W	Juin 2014	20	Kitkat Watch
Android 4.4	Octobre 2013	19	Kitkat
Android 4.3	Juillet 2013	18	Jelly Bean
Android 4.2	Novembre 2012	17	Jelly Bean
Android 4.1	Juin 2012	16	Jelly Bean
Android 4.0.3	Décembre 2011	15	Ice Cream Sandwich
Android 4.0	Octobre 2011	14	Ice Cream Sandwich
Android 3.2	Juin 2011	13	Honeycomb
Android 3.1	Mai 2011	12	Honeycomb
Android 3.0	Février 2011	11	Honeycomb
Android 2.3.3	Février 2011	10	Gingerbread
Android 2.3	Novembre 2010	9	Gingerbread
Android 2.2	Juin 2010	8	Froyo
Android 2.1	Janvier 2010	7	Eclair
Android 2.0.1	Décembre 2009	6	Eclair
Android 2.0	Novembre 2009	5	Eclair
Android 1.6	Septembre 2009	4	Donut
Android 1.5	Mai 2009	3	Cupcake
Android 1.1	Février 2009	2	Base
Android 1.0	Octobre 2008	1	Base

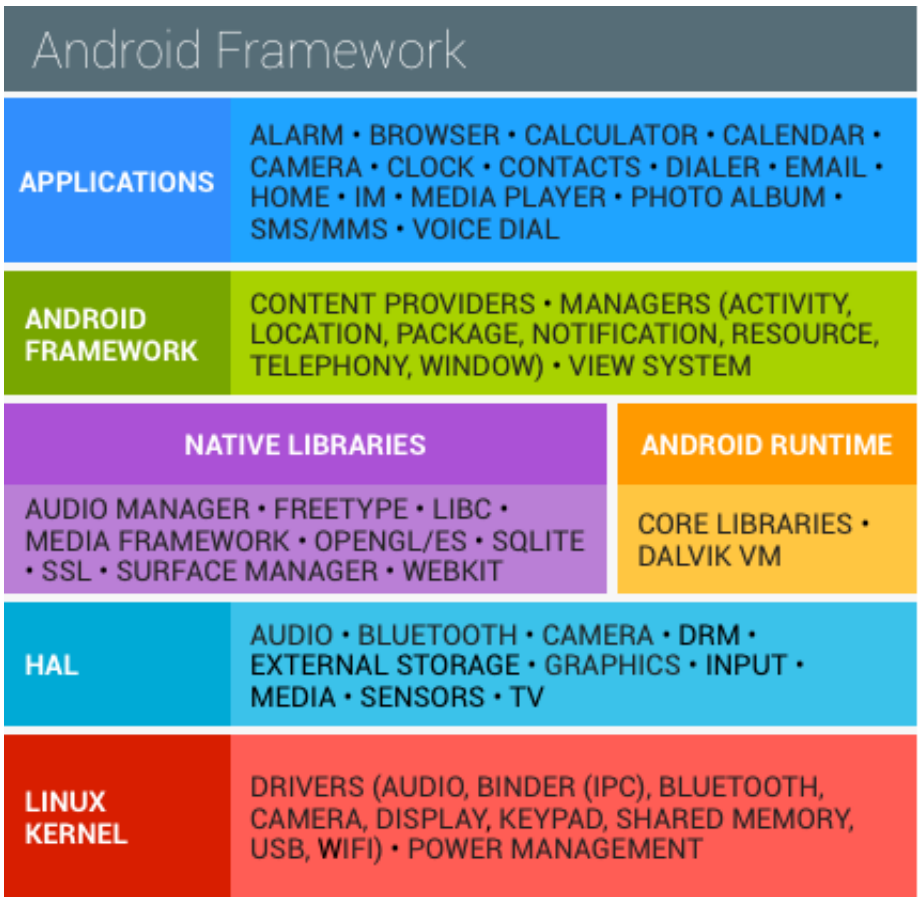
Remarque 1 : Chaque version d'Android est identifiée par un entier unique, nommé « API Level », et a un nom qui correspond à un dessert (sauf les versions >= 10) : Pie, Oreo, Nougat, Eclair, etc.

Remarque 2 : Une application développée avec une API (exemple l'API 21), fonctionne correctement sur les API les plus récentes (et les OS correspondant) mais ne peut pas fonctionner sur les API les plus anciennes.

Principales différences entre les versions :

- Correction de bugs
- Nouvelles APIs avec des nouvelles fonctionnalités
- Nouveau mode d'exécution : passage d'une VM (Dalvik) à une autre (ART)
- Optimisation énergétique
- Modification de l'apparence de l'UI (thèmes, notifications, fenêtrage, etc.)
- Amélioration de la sécurité
- Support des nouvelles technologies (5G, IoT, waterfall displays : écrans incurvés, etc.)

Pile logicielle Android :



1) Application Android

a) Composants d'une application

Une application Android peut être composée des éléments suivants :

- Des activités (android.app.Activity) : il s'agit d'une partie de l'application présentant une interface à l'utilisateur ;
- Des services (android.app.Service) : il s'agit d'une tâche de fond sans interface associée ;
- Des fournisseurs de contenus (android.content.ContentProvider): permettent le partage d'informations au sein ou entre applications ;
- Des widgets (android.appwidget.*) : une vue accrochée au Bureau d'Android ;
- Des Intents (android.content.Intent) : permettent d'envoyer un message pour un composant externe sans le nommer explicitement ;
- Des récepteurs d'Intents(android.content.BroadcastReceiver): permettent de déclarer être capable de répondre à des Intents ;
- Des notifications (android.app.Notifications) : permettent de notifier l'utilisateur de la survenue d'événements.

b) Activité

Activity : une activité est une fenêtre sous Android. Elle peut contenir des vues (View). Il y a deux types de vues :

- Les vues simples, telles que :
    - Button : bouton
    - TextView : étiquette
    - EditText : champ de texte
- Les conteneurs de vues, tels que :
    - LinearLayout : un conteneur qui peut disposer les vues horizontalement ou verticalement
    - ListView : dispose les vues dans une liste

Une application peut contenir plusieurs activités. Il faut déclarer une activité par défaut qui s'affiche au lancement d'une activité.

c) Le Manifest de l'application

Le fichier AndroidManifest.xml déclare l'ensemble des éléments de l'application.

Exemple :

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

    <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.MyStore">
        <activity
            android:name=".RegisterActivity"
            android:exported="true" />
        <activity
            android:name=".LoginActivity"
            android:exported="true" />
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

d) Les ressources

Il y a plusieurs types de ressources :

- Ressources de type XML :
  - Layout
  - Menu
  - Valeurs : Strings, Colors, Styles, etc.
- Ressources de type image :
  - Mipmap
  - Drawable
- Resources binaires : row

Les ressources sont accessibles dans le code Java en utilisant la classe statique « R »

2) Environnement de développement

Pour développer des applications Android, il faut :

- L'environnement de développement (IDE) « Android Studio »
- Le « Software Development Kit » (SDK) qui contient les librairies (API) nécessaires pour développer des applications Android (c'est l'équivalent de la JDK dans le cas des applications bureautiques)
- Un smartphone Android pour tester les applications réalisées. Il y a 2 possibilités
    - On utilise un émulateur qui simule un Smartphone Android sur un ordinateur :
        - Si la machine supporte la Virtualisation, utiliser un émulateur **intel** (plus rapide)
        - Si la machine ne supporte pas la Virtualisation, utiliser un émulateur **arm** (lent)
        - Il y a d'autres émulateurs à part l'émulateur officiel, comme **BlueStack**
    - On utilise un Smartphone Android réel connecté par câble USB à l'ordinateur : Il faut activer l'option « USB debugging » à partir de « paramètres > options développeur »

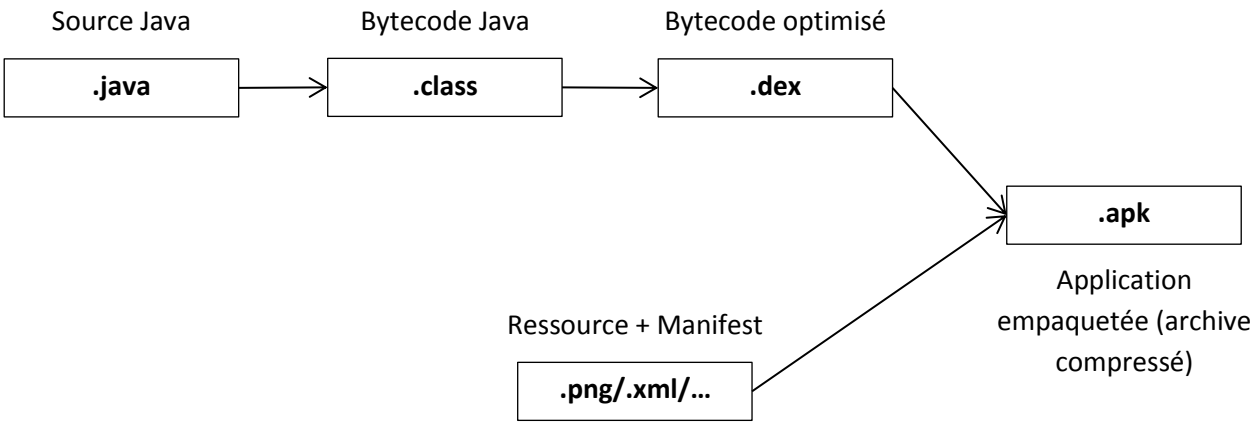
Nous allons installer « **Android Studio** » à partir de :
https://developer.android.com/studio/

Android studio contient les 2 outils suivants :

- Software Development Kit (SDK) Manager : permet de visualiser les différentes versions de SDK (un numéro API correspond à chaque version de SDK) disponibles et les télécharger. Accès possible à partir de « **Tools > Android > SDK Manager** »
- Android Virtual Device (AVD) Manager : gère les différents émulateurs. Il est possible de créer plusieurs émulateurs pour les différentes tailles de tablettes et de Smartphones. Accès possible à partir de « **Tools > Android > AVD Manager** »

3) Compilation

Java permet d'obtenir un exécutable (**.class**, **.jar**, **.war**, **.apk**, etc.) qui tourne sur une machine virtuelle. La machine virtuelle pour les applications bureautiques et web est Java Virtual Machine (JVM) (le programme **java.exe** pour Windows). Pour Android il y a l'ancienne machine virtuelle Dalvik qui est remplacée par Android Runtime (ART).



4) Les outils « SDK Manager » et « AVD Manager »

Les outils « SDK Manager » et « AVD Manager » sont accessibles à partir de « Android Studio » ou à partir du dossier d'installation du SDK.

a) SDK Manager

Cet outil est accessible à partir de « Android Studio » : **Tools > Android > SDK Manager**

Il permet d'installer tous les éléments nécessaires pour programmer, tels que : API, Outils de compilation, Emulateurs, Etc.

b) AVD Manager

L'outil « AVD (Android Virtual Device) Manager » est accessible à partir de « Android Studio » :

Tools > Android > AVD Manager

Il permet de créer des nouveaux émulateurs avec différentes résolutions. Il permet aussi de lancer un émulateur.

Remarque 1 : Le lancement d'un émulateur peut être très lent (entre 1 à 10 min). Il est donc nécessaire de lancer l'émulateur une seule fois et de le garder ouvert pour l'utiliser avec tous les tests.

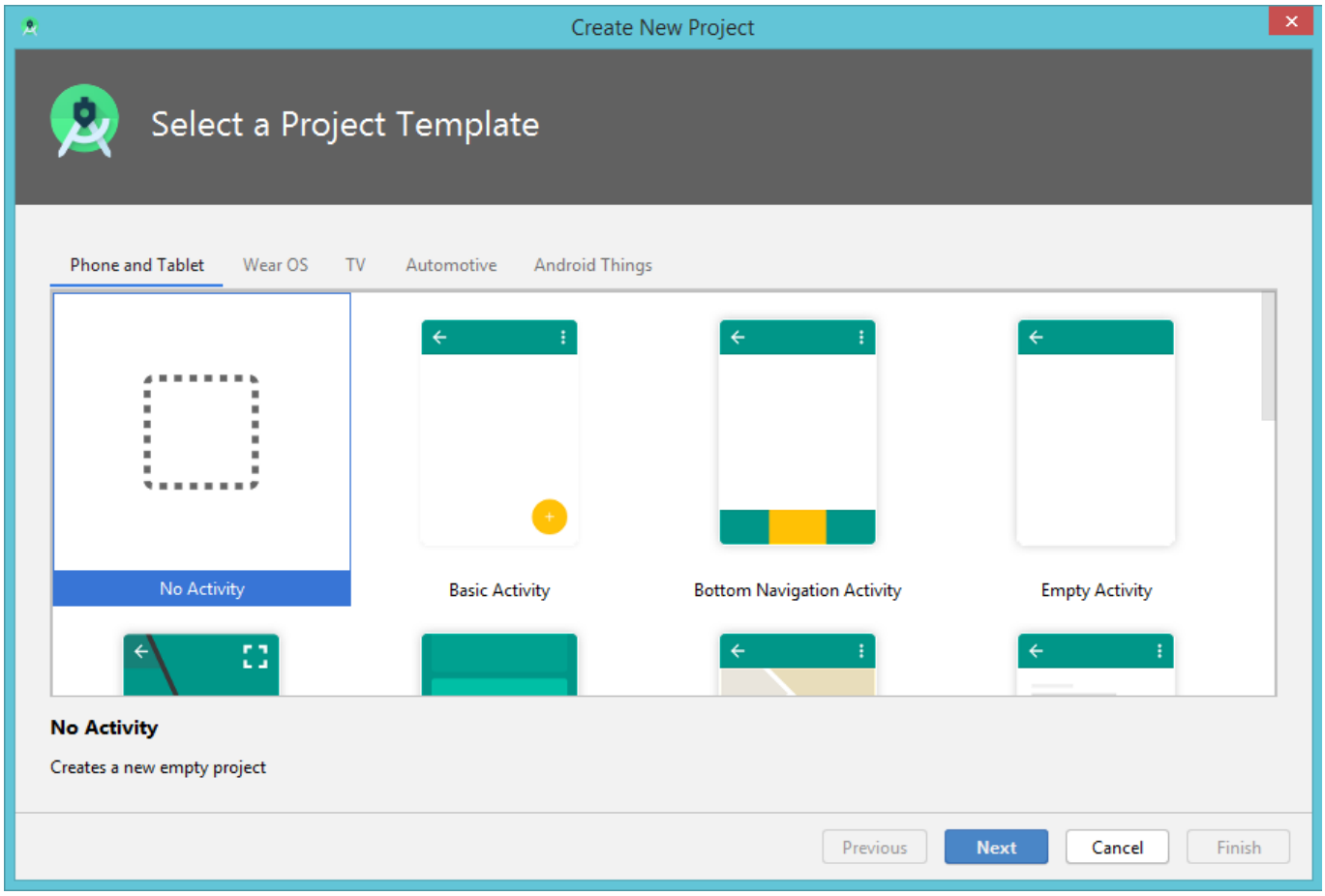
Remarque 2 : un émulateur est un téléphone virtuelle qui nécessite un système d'éxploitation. Donc il faut télécharger une version Android et l'installer sur l'émulateur en utilisant AVD.

5) Projet Android

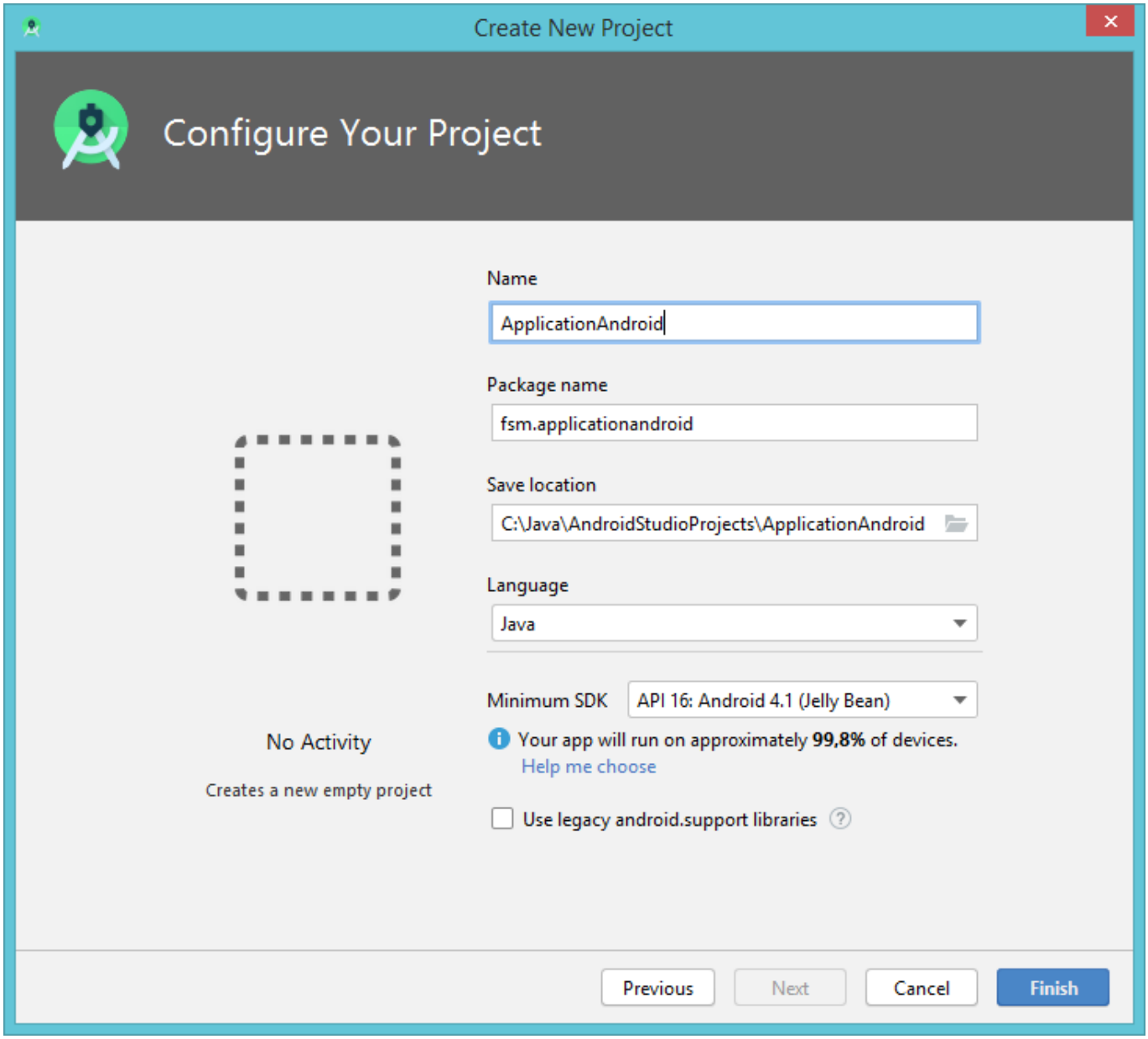
a) Création d'un projet Android

Pour créer une nouvelle application Android, il faut créer un nouveau projet sous « Android Studio » en suivant les étapes suivantes :

- Clic sur « **File > New > New Project** » et la fenêtre « Create New Project » s'ouvre :



- Choisir « **No Activity** » et clic sur le bouton « **Next** »
- Dans la fenêtre qui s'ouvre, renseigner le nom de l'application dans le champ « **Name** », le nom du package et l'emplacement de sauvegarde du projet. Choisir le langage de programmation **Java** et Cliquer sur le bouton « **Finish** ».



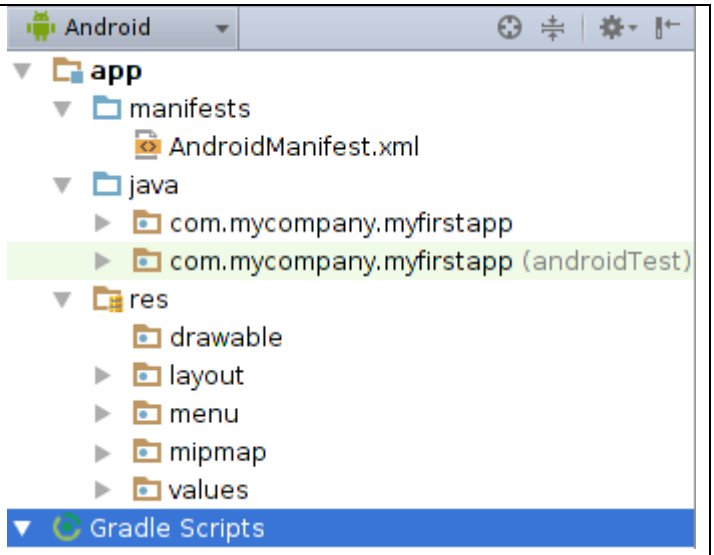
b) Structure d'un Projet Android

Chaque projet Android contient le module « **app** » qui regroupe l'ensemble des codes sources (fichiers java) et les fichiers ressources (fichiers layout, XML, images, constantes, etc.). Dans le module « app », les fichiers sont affichés dans les groupes suivants :

- **manifests** : contient le fichier « **AndroidManifest.xml** ». Ce fichier est essentiel et contient la liste des composants d'une application. Par exemple, il contient l'ensemble des activités disponibles et l'activité principale qui s'affiche au lancement de l'application.
  Remarque : si une activité n'est pas déclarée dans le fichier manifest, il n'est pas possible de l'afficher dans l'application.

- **Java** : contient tous les fichiers java

- **res** : contient tous les fichiers ressources (non java), tels que layouts, images, XML, etc.



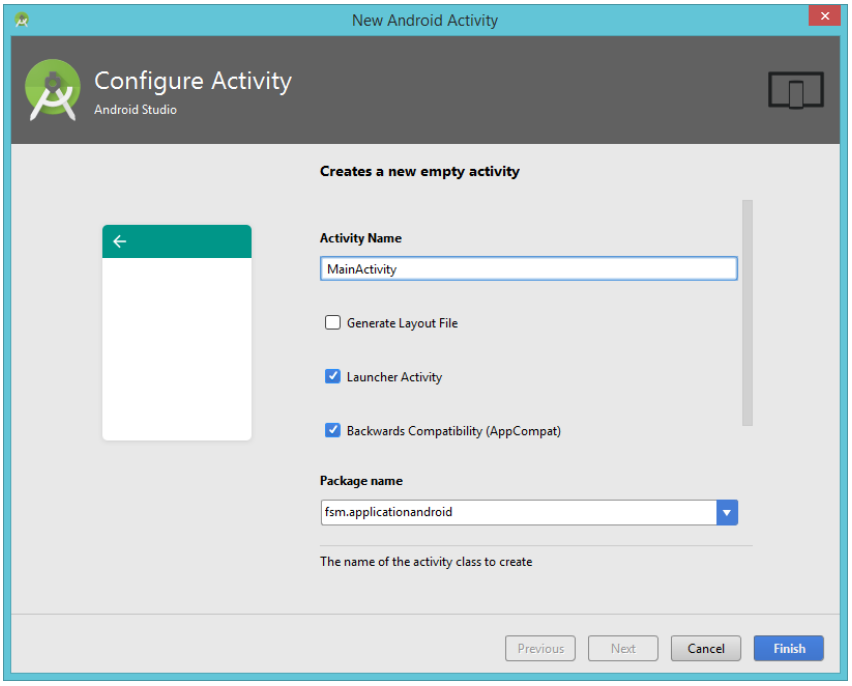
c) Ajouter des fichiers au projet

Pour ajouter un fichier au projet, vérifier d'abord que le module « app » est sélectionné.

Ajouter une activité

Pour ajouter une activité au projet, il faut suivre les étapes suivantes :

- Clic sur « **File > New > Activity > Empty Activity** », et la fenêtre « **Configure Activity** » s'ouvre :



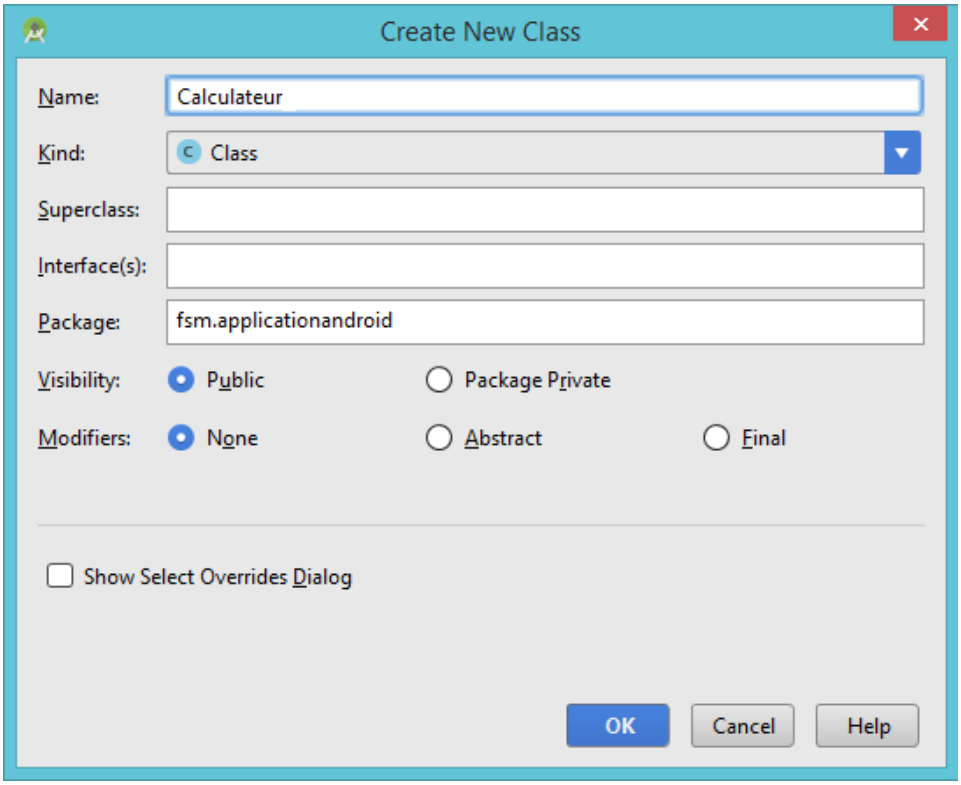
- Saisir un nom de cette activité dans le champ « **Activity Name** ». Décocher la case « **Generate Layout File** » Si cette activité doit s'afficher au moment de lancement de l'application, cocher « **Launcher Activity** », sinon décocher cette case. Clic sur le bouton « **Finish** » et une nouvelle activité s'ajoute au projet.

Remarque : Une activité est une classe Java qui doit être déclarée dans le fichier manifest du projet. Si on ajouter cette classe en suivant les étapes précédentes, « Android Studio » se charge de la déclarer dans le fichier manifest.

Ajouter une classe Java

Pour ajouter une classe java ordinaire, il faut suivre les étapes suivantes :

- Clic sur « **File > New > Java Class** » et la fenêtre suivante s'ouvre :

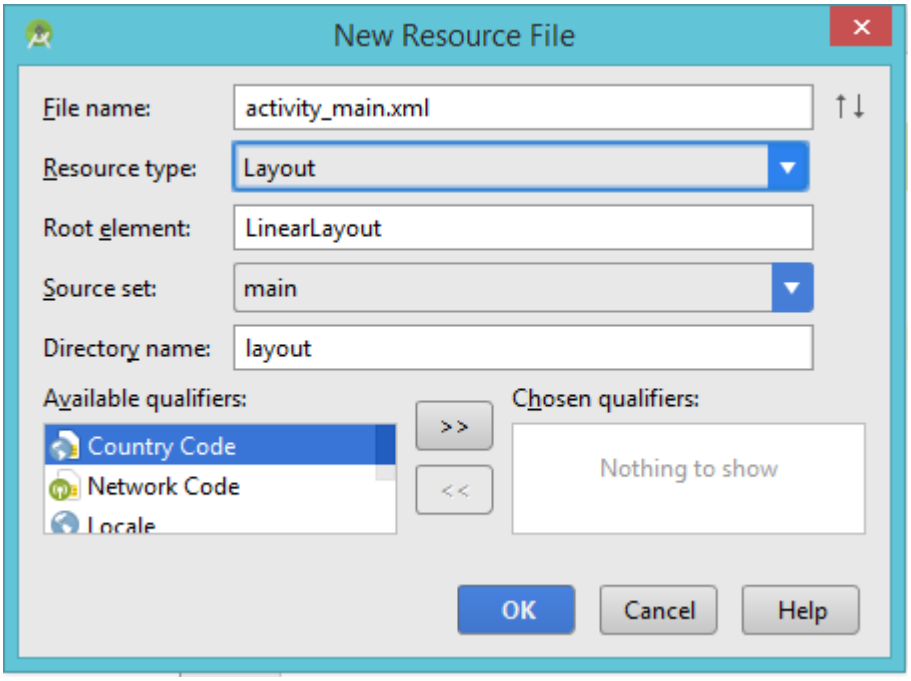


- Renseigner le nom de la classe dans le champ « **Name** ». Clic sur le bouton « **OK** » et la nouvelle classe s'ajoute au projet

Ajouter un fichier Layout

Un fichier Layout permet de construire l'interface graphique en utilisant le Designer et un code XML. Pour ajouter un fichier Layout, il faut suivre les étapes suivantes :

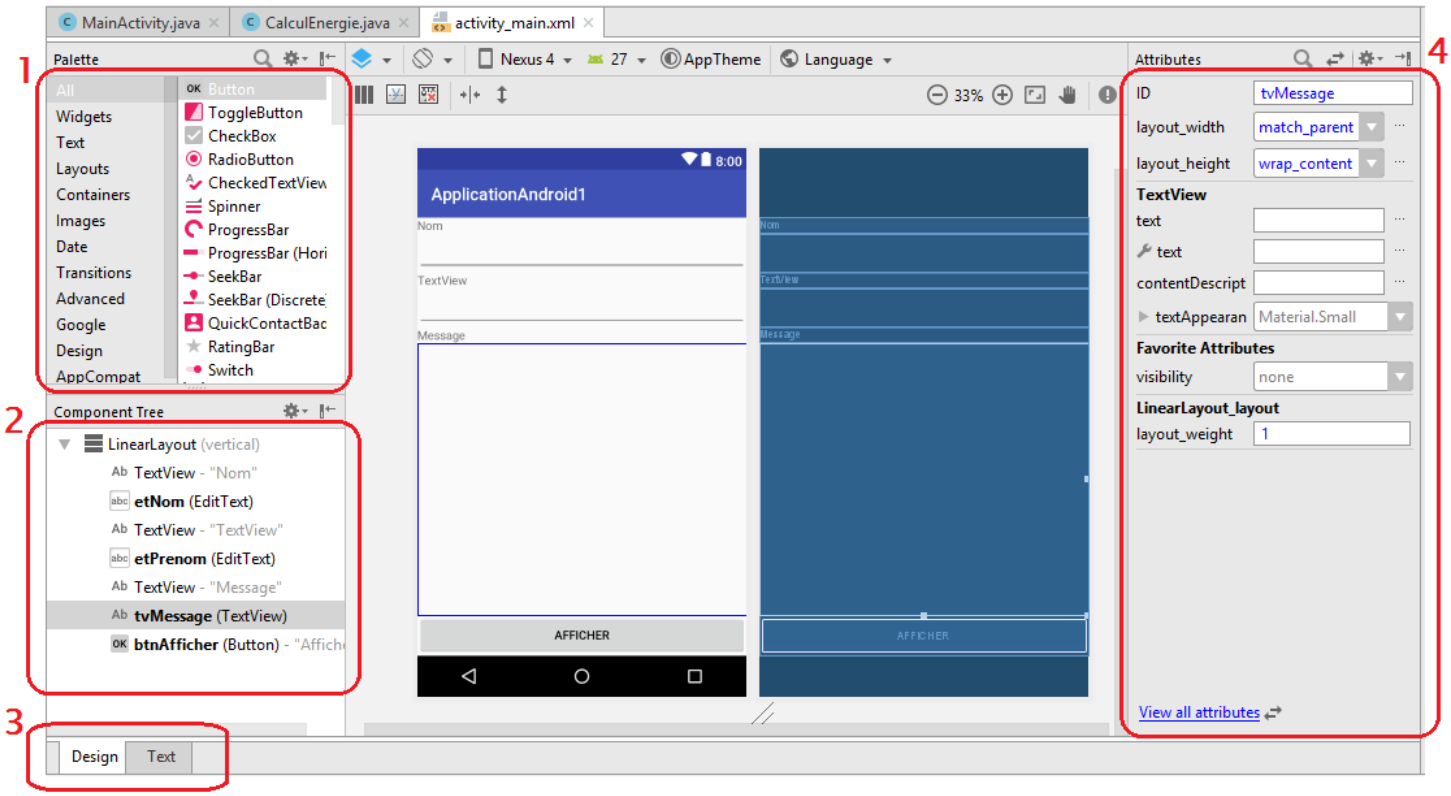
- Clic sur « **File > New > Android Resource File** », et la fenêtre suivante s'ouvre :



- Saisir le nom du fichier Layout dans le champ « **File name** » (**NE PAS** oublier l'extension XML du fichier). A partir du champ « **Resource type** », choisir le type « **Layout** ». Clic sur le bouton « **OK** ». Le nouveau fichier sera ajouter dans le groupe « **app > res > layout** ».

Construire une interface graphique

Après l'ajout d'un fichier layout, il est possible de créer l'interface graphique avec le designer à partir de la fenêtre suivante :



La **zone 1** contient les différentes vues (composants) qu'on peut ajouter à la GUI.

La **zone 2** contient les différentes vues de la GUI. Il faut sélectionner une vue pour voir et modifier ses attributs à partir de **la zone 4**.

La **zone 3** permet de basculer entre le designer et le code XML du fichier Layout.

La **zone 4** permet de renseigner la valeur des différents attributs (les attributs XML) d'une vue, tel que :

- **ID** (identifiant d'une vue) : permet d'accéder à la vue à partir du code Java. Il est possible de laisser une vue sans identifiant si elle ne sera pas modifiée à partir du code Java.
- **text** : le texte qui sera affiché sur la vue
- **layout_width** : la largeur de la vue dans son conteneur parent (LinearLayout dans cet exemple)
- **layout_height** : la hauteur de la vue dans son conteneur parent. Les valeurs possibles pour **layout_width** et **layout_height** sont : **wrap_content** (taille vue = taille de son contenue) et **match_parent** (taille vue = taille restant du parent : impossible d'ajouter d'autres vues).
- **layout_weight** (la part d'une vue de son conteneur parent) : si une seule vue a un **layout_weight** dans un LinearLayout, elle occupe tout l'espace libre du LinearLayout. Sinon, l'espace libre sera partagé selon la part de chaque vue (exemple : **layout_weight=1** pour vue1 et **layout_weight=2** pour vue2 implique vue1 occupe 33% et vue2 occupe 66% de l'espace libre).

Par défaut, la **zone 4** affiche les attributs principaux d'une vue. Pour afficher tous les attributs, il faut appuyer sur « **View all attributes** ». Si tous les attributs sont affichés, il est possible d'afficher les attributs principaux en appuyant sur « **View fewer attributes** » (tout en bas de la liste des attributs).

Le designer permet de créer le fichier Layout correspondant à l'interface. Pour consulter le code XML de ce fichier, appuyer sur « Text » à partir de la **zone 3**. Exemple de code :

```
<Button
    android:id="@+id/btnAfficher"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Afficher" />
```

La balise XML « **Button** » contient les différents attributs de la vue « **Button** ».

d) Les ressources

Les ressources de l'application (disponibles dans le dossier virtuel « **res** ») sont accessibles à partir du code Java à travers la class statique « R ». Cette classe est régénérée automatiquement à chaque changement dans les ressources du projet.

Remarque : Si l'un des fichiers ressource contient des erreurs, la classe « R » devient inaccessible (càd l'appel de la classe « R » génère une erreur).

Les ressources sont accessibles à partir du code Java de la manière suivante :

`R.type_ressource.nom_ressource`

Il y a plusieurs types de ressources tel que :

- **R.id** : accès par identifiant
- **R.layout** : accès aux fichiers layout
- **R.menu** : accès aux ressources de type menu
- **R.string** : accès aux chaînes de caractères
- Etc. (voir documentation)

Le nom de la ressource peut être :

- le nom du fichier ressource (sans l'extension)
- l'attribut **« android:name »** du fichier XML
- l'identifiant de la ressource (attribut **« android:id »**) si on y accède avec « **R.id** »

La classe « R » retourne l'identifiant de la ressource. Cet identifiant permet ensuite de récupérer l'instance de la ressource en utilisant les méthodes adéquates, telles que :

- La méthode « **`getString()`** » de la classe « Ressources »
- La méthode « **`findViewById()`** » de la classe « Activity » : permet d'accéder à des composants graphiques (Vues) à partir de leurs identifiants.

Exemple 1 :

```
Resources res = getResources();
String title = res.getString(R.string.app_title);
```

Exemple 2 :

```
Button btn = (Button) findViewById(R.id.ok_button);
Btn.setText("suivant");
```

6) Interface utilisateur

a) Vues et Layouts

Les composants graphiques héritent de la classe « **View** ». On peut regrouper plusieurs composants graphiques dans un « **ViewGroup** ». Un « **ViewGroup** » peut contenir des vues simples et d'autres « **ViewGroup** ». Parmi les « **ViewGroup** » on trouve les layouts qui permettent d'organiser et de positionner les différents composants graphiques.

Exemples de layouts :

- **LinearLayout** : dispose les éléments de gauche vers la droite ou du haut vers le bas, selon l'orientation du layout (horizontale ou verticale).
- **RelativeLayout** : dispose les éléments les uns par rapport aux autres.
- **TableLayout** : disposition matricielle avec N lignes et M colonnes.
- **GridLayout** : disposition matricielle avec M colonnes et un nombre indéterminé de lignes.
- **ListView** : dispose les éléments dans une liste.

Exemples de vues simples : Button, TextView, EditText, etc.

b) Les activités (Activity)

Une application peut contenir une ou plusieurs activités. Le fichier Manifest permet de déclarer l'activité principale qui sera affichée au lancement de l'application (Launcher Activity). Toutes les activités doivent être déclarées dans le manifest.

Par défaut, une activité contient la méthode « onCreate ». Cette méthode est héritée de la classe « Activity ». Elle est appelée lorsque l'activité est créée par le système et entre dans l'état **Created**. Généralement, les opérations effectuées dans cette méthode servent à mettre en place l'interface graphique, à initialiser les variables, à configurer les listeners ou à se connecter à la base de données.

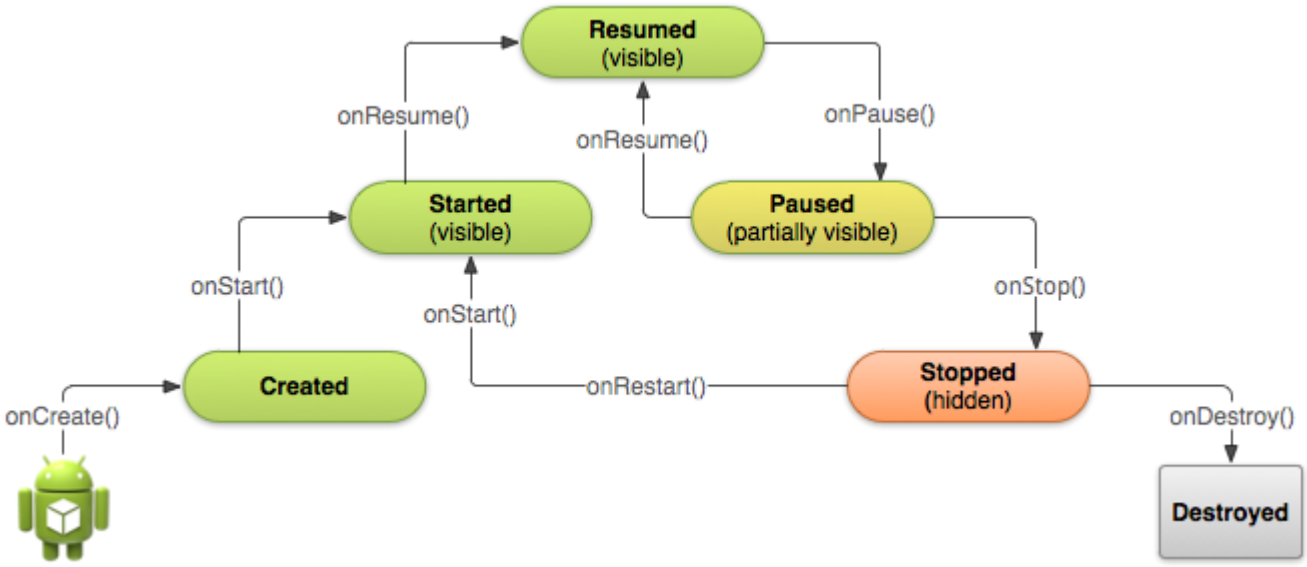
Exemple d'une activité :

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        //your code here
    }
}
```

A l'état **Created**, l'activité est créée mais elle n'est pas encore visible. Les différents états d'une activité sont : **Created**, **Started**, **Resumed**, **Paused**, **Stopped** et **Destroyed**.



Différents états d'une activité

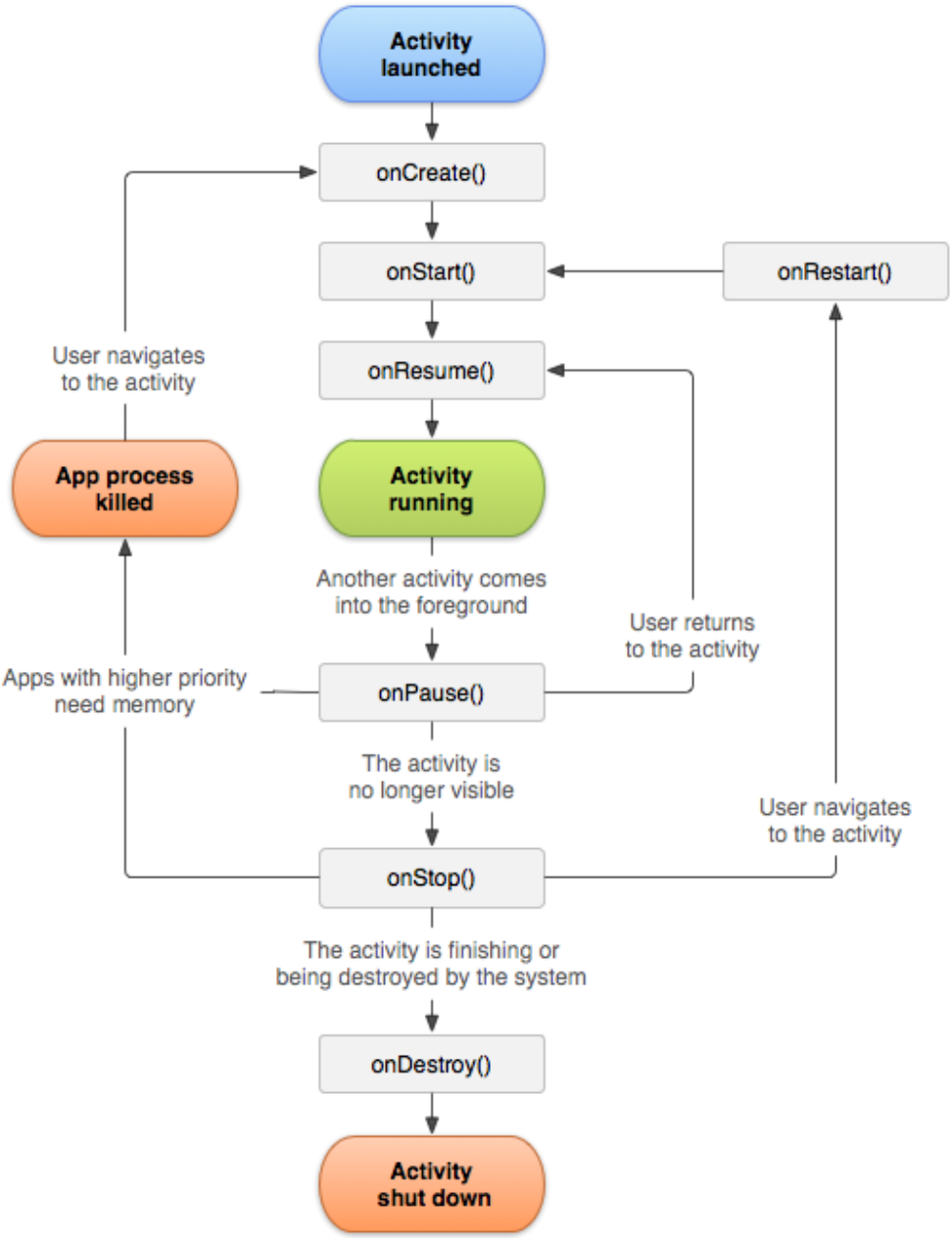
onStart() : Cette méthode est appelée par le système lorsque l'activité entre dans l'état **Started**. L'interface graphique devient visible à l'utilisateur, mais il ne peut pas encore interagir avec les différents éléments.

onResume() : Cette méthode est appelée lorsque l'activité entre dans l'état **Resumed**. L'activité devient entièrement opérationnelle. L'utilisateur peut utiliser l'application et cliquer sur les différents éléments graphiques. L'application reste dans cet état tant qu'il n'y a pas d'interruption, comme par exemple, la réception d'un appel téléphonique, le démarrage d'une nouvelle activité ou l'affichage d'une boîte de dialogue.

onPause() : Cette méthode est appelée lorsque l'activité entre dans l'état **Paused**. Tout ce qui est initié dans onResume() doit être mis en pause dans cette méthode. Par exemple, une animation présentée à l'utilisateur est démarrée dans onResume() puis stoppée dans onPause().

onStop() : Cette méthode est appelée lorsque l'activité entre dans l'état **Stopped**. Par exemple, lorsqu'une nouvelle activité est démarrée, l'activité appelante va se retrouver dans cet état. Elle n'est donc plus visible à l'utilisateur. Les traitements liés à la mise à jour de l'interface graphique peuvent être arrêtés. Les traitements effectués dans cette méthode peuvent être plus importants (comme sauvegarder certaines valeurs dans les SharedPreferences par exemple).

onDestroy() : Cette méthode est appelée lorsque l'activité est arrêtée. Par exemple, ce peut être après avoir appelée la méthode finish(), ou si le système décide d'arrêter l'activité pour libérer de la mémoire.



Activity Lifecycle (https://developer.android.com/guide/components/activities/activity-lifecycle)

Afficher un fichier Layout dans une activité (Activity)

Pour afficher une interface graphique (GUI : Graphical User Interface) dans une activité, on appel la fonction suivante:

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

L'accès à partir d'une activité aux différents composants d'un layout se fait moyennant leurs identifiants. Cela est possible comme suit :

```
findViewById(R.id.idVue); // idVue est l'identifiant renseigné dans
                          // l'attribut XML « android:id »
```

Exemple :

```
EditText etNom = findViewById(R.id.etNom);
EditText etPrenom = findViewById(R.id.etPrenom);
TextView tvMessage = findViewById(R.id.tvMessage);
```

Ajouter un listener à un bouton

Pour ajouter un listener à un bouton, il est possible d'utiliser la fonction « **setOnClickListener(OnClickListener l)** » d'un bouton comme suit :

```
    btnAfficher.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            // code
        }
    });
```

Lancement d'une nouvelle activité

Il est possible de lancer une nouvelle activité (nommé activité enfant) à partir de l'activité courante (nommé activité parent). Pour quitter une activité, on peut utiliser la fonction « **finish()** ».

Sans attendre un résultat

Pour lancer une nouvelle activité sans attendre un résultat, on utilise la classe « **Intent** » et on lui donne le nom de l'activité à lancer. Ensuite on appelle la fonction « **startActivity** ».

Exemple de lancement de l'activité « **AjouterCommandeActivity** » :

```
Intent intent = new Intent(this, AjouterCommandeActivity.class);
startActivity(intent);
```

Il est possible d'envoyer des données à l'activité qui sera affichée (activité enfant) à travers la méthode « **putExtra** » de l'intent de lancement. Cette méthode reçoit des pairs <**clé, valeur**>. Exemple :

```
Intent intent = new Intent(this, AjouterCommandeActivity.class);

intent.putExtra("description", "Nouvelle commande pour Tunis");
intent.putExtra("client", "Ahmed Abid");
intent.putExtra("numero", 1142512);
intent.putExtra("paiement", false);

startActivity(intent);
```