République Tunisienne
Ministère de l'Education
Centre National de Formation des Formateurs en Education

Module de formation

Intitulé :

Programmation Android

Niveau 1

Discipline : Informatique

Réalisé par :

Raouf OURARI

Nizar Gannouni

Année 2016

Tables des matières

Préface

Le 21ème siècle est l'ère du numérique. La vulgarisation de l'utilisation des appareils de communication tactiles tels que les smartphones et les tablettes chez les jeunes a rendu obligatoire l'introduction de leurs utilisations dans l'écosystème scolaire.

Des nouvelles pratiques pédagogiques, qui développent des modalités d'apprentissage plus collaboratives et permettent un meilleur accompagnement des élèves, ont été mises en place. Ces nouveaux supports permettent de mettre en œuvre des activités diversifiées, adaptables à des contextes d'apprentissage variés et aux besoins spécifiques de chaque apprenant.

Dans les dernières années, plusieurs établissements de différents pays ont voulu ou veulent munir chaque élève d'une tablette tactile. En Tunisie, les expérimentations sont en progression continue et avec la rentrée scolaire 2016/2017 le ministère de l'éducation tunisien a décidé de tenter l'expérimentation à une échelle plus grande. Ainsi et afin de se préparer aux années avenir, la formation des professionnels de l'éducation dans le domaine de développement Android, système d'exploitation le plus utilisé avec les tablettes, demeure obligatoire dans le but de garantir une efficacité accrue des pratiques pédagogiques.

Dans ce cadre, se place la réalisation de ce module de formation intitulé **«Programmation Android Niveau 1»** destiné à la formation continue des inspecteurs et des enseignants d'informatique.

La première partie de ce document comporte un planning détaillé de la formation qui s'étale sur 5 journées de formation à raison de 5h par jour. Ensuite vient la section 1, dans laquelle on a présenté les notions de base de la programmation Android et enfin dans la section 2, on a proposé des exercices d'applications permettant au formé de développer ses capacités.

Objectifs et planning de la formation

L'objectif principal de cette formation est d'initier les enseignants au développement Android.

La formation prévue devra s'effectuer dans un laboratoire d'informatique à raison d'un apprenant par poste. Elle s'étale sur cinq journées où se chevauchent apprentissage, discussions et productions. Les tableaux ci-joints présentent en détail la consistance de chaque journée.

Journée I

Thème : Introduction L'objectif de cette session est de présenter la formation et connaitre les attentes des participants		
Contenu	**Durée**	**Remarques**
• Prise de contact • Présentation de la formation • Brainstorming sur les attentes des participants	30mn	
Thème : Installation de l'environnement L'objectif de cette session est de préparer l'environnement de développement des applications Android		
Contenu	**Durée**	**Remarques**
• Présentation • Java • Emulateur • Android Studio	2h30	Travail en groupe et manipulation directe sur machine
Thème : Création d'une application basique L'objectif de cette session est de créer la première application sous Android		
Contenu	**Durée**	**Remarques**
• Création d'un nouveau projet • Lancement de Genymotion • Exécution	2h	Travail individuel sur l'environnement de développement

Déroulement de la 1ère journée

Séquences	Activités et consignes	Activités du formateur	Modalités de travail	Activités des participants	Matériels et documents	Durées
1ère	Accueil des enseignants et présentation de la formation	Présente l'ordre du jour	Exposé	Ecoutent le formateur	Vidéo projecteur	**15 mn**
2ème	Brainstorming sur les attentes des participants	Organise la discussion	Discussion	Echange des idées	Tableau	**15mn**
3ème	Présentation, installation et configuration de l'environnement	Assiste la réalisation des actions demandées	Travail en groupe	Appliquent les instructions d'installation et de configuration	Vidéo projecteur et ordinateurs	**2h30**
4ème	Création de la première application sous Android	Présente les étapes et aide les formés dans les activités pratiques	Travail pratique individuel	Appliquent les étapes présentées	Vidéo projecteur et ordinateurs	**2h**

Journée II

Thème : La structure d'un projet Android L'objectif de cette session est de présenter la structure d'un projet Android développé sous Android Studio		
Contenu	**Durée**	**Remarques**
• Le fichier AndroidManifest.xml • Le fichier Build.gradle • Le fichier MainActivity.java • Le dossier layout • Le dossier Drawable • Le dossier Menu • Le dossier values	5h	Travail individuel ou/et en équipe

Déroulement de la 2ème journée

Séquences	Activités et consignes	Activités du formateur	Modalités de travail	Activités des participants	Matériels et documents	Durées
1ère	Accueil des enseignants et présentation de l'objectif de la journée	Présente l'ordre du jour	Exposé	Ecoutent le formateur	Vidéo projecteur	**10 mn**
2ème	Présentation du fichier AndroidManifest.xml	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**50 mn**
3ème	Présentation du fichier Build.gradle	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**30 mn**
4ème	Présentation du fichier MainActivity.java	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**1h**
5ème	Présentation du dossier Layout	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**1h 30**
6ème	Présentation du dossier Drawable	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**30 mn**
7ème	Présentation du dossier Menu	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**05 mn**
8ème	Présentation du dossier Values	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**25 mn**

Journée III

Thème : Interface utilisateur L'objectif de cette session est de présenter quelques composants de l'interface utilisateur		
Contenu	**Durée**	**Remarques**
• Layout • TextView • EditText • Bouton	5h	Travail individuel

Déroulement de la 3ème journée

Séquences	Activités et consignes	Activités du formateur	Modalités de travail	Activités des participants	Matériels et documents	Durées
1ère	Accueil des enseignants et présentation de l'objectif de la journée	Présente l'ordre du jour	Exposé	Ecoutent le formateur	Vidéo projecteur	**10 mn**
2ème	Présentation des Layouts	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**50 mn**
3ème	Présentation de la vue TextView	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**1h :30**
4ème	Présentation de la vue EditText	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**1h :30**
5ème	Présentation de la vue Button	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**1h**

Journée IV

Thème : Interface utilisateur L'objectif de cette session est de présenter quelques composants de l'interface utilisateur		
Contenu	**Durée**	**Remarques**
• RadioButton • CheckBox • Spinner • ImageView • Propriétés d'une vue	5h	Travail individuel et / ou en équipe

Déroulement de la 4ème journée

Séquences	Activités et consignes	Activités du formateur	Modalités de travail	Activités des participants	Matériels et documents	Durées
1ère	Accueil des enseignants et présentation de l'objectif de la journée	Présente l'ordre du jour	Exposé	Ecoutent le formateur	Vidéo projecteur	**10 mn**
2ème	Présentation de la vue RadioButton	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**1h**
3ème	Présentation de la vue checkBox	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**1h**
4ème	Présentation de la vue spinner	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**2h30**
5ème	Présentation de la vue imageView	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**20 mn**

Journée V

Thème : Application L'objectif de cette session est de présenter les notions d'évènements, intent et la génération de l'APK		
Contenu	**Durée**	**Remarques**
• Les évènements • Multi-activities et intent • Générer l'APK	5h	Travail individuel et discussion en plénière

Déroulement de la 5ème journée

Séquences	Activités et consignes	Activités du formateur	Modalités de travail	Activités des participants	Matériels et documents	Durées
1ère	Accueil des enseignants et présentation de l'objectif de la journée	Présente l'ordre du jour	Exposé	Ecoutent le formateur	Vidéo projecteur	**10 mn**
2ème	Présentation de la notion d'évènements	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**2h**
3ème	Présentation de la notion Intent et multi-activities	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**2h**
4ème	Génération du fichier APK	Expose et intervient dans la réalisation des activités pratiques	Travail individuel ou en équipe	Ecoutent et appliquent les actions présentées	Vidéo projecteur et ordinateurs	**50 mn**

Section I :

Initiation au langage Android

I- Introduction

Android est la plate-forme mobile la plus populaire au monde. Au dernier décompte, il y avait plus d'un milliard[1] d'appareils Android actifs dans le monde, et ce nombre croît rapidement.
Android est une plate-forme open source complète basée sur Linux et défendue par Google. C'est un cadre de développement puissant qui comprend tout ce dont on a besoin pour construire de grandes applications en utilisant une combinaison de Java et XML. De plus, il permet de déployer ces applications sur une grande variété d'appareils- téléphones, tablettes, etc...
Une application Android typique est constituée par :

- Les layouts définissant les interfaces utilisateurs :

  Une application Android typique se compose d'un ou de plusieurs écrans. Les layouts sont définis en utilisant XML, et peuvent inclure des éléments graphiques comme des boutons, des champs de texte, des étiquettes, etc.

- Le code Java définissant ce que l'application devrait faire :
  Les layouts ne définissent que l'apparence de l'application. Ce que l'application fait est définit en écrivant du code Java dans une classe spéciale appelée **Activity** qui indique à l'application comment répondre à une interrogation de l'utilisateur. À titre d'exemple, si une page comprend un bouton, on a besoin d'écrire du code Java dans l'activité appropriée pour définir ce que le bouton doit faire lorsque l'utilisateur appuie dessus.

- Parfois des ressources supplémentaires :

  En plus du code Java et des interfaces utilisateurs, les applications Android ont souvent besoin de ressources supplémentaires telles que des images, des données d'application, etc.

II- Installation de l'environnement

Pour développer une application Android, on a besoin d'un :

- ensemble d'outils de développement appelé SDK, Software Development Kit. Le SDK Android, développé par Google, est composé de plusieurs éléments nécessaires à la création et à la maintenance des applications :
  - ✓ des API (interfaces de programmation) ;

[1] http://www.frandroid.com/marques/google/313643_android-present-plus-de-14-milliard-dappareils-actifs

✓ des exemples de code ;
✓ de la documentation ;
✓ des outils, parmi lesquels un émulateur, qui permettent de couvrir toutes les étapes du cycle de développement d'une application.

- Java Development Kit (JDK) qui regroupe l'ensemble d'outils indispensables pour compiler, déboguer et créer le code binaire (byte code) de l'application.
- émulateur mobile pour simuler l'exécution des applications créées. **Exemples** : Genymotion, Andy Os, BlueStacks, etc.

1- Installation de l'environnement Java

Télécharger et installer le JDK, **Java Development Kit**. Il est téléchargeable à partir du site officiel d'Oracle :

http://www.oracle.com/technetwork/java/javase/downloads/index.html

2- Installation d'un émulateur

Télécharger un émulateur pour simuler l'exécution de toute application développée. On a choisi comme émulateur l'outil **Genymotion** qu'on pourra le télécharger à partir du lien suivant :

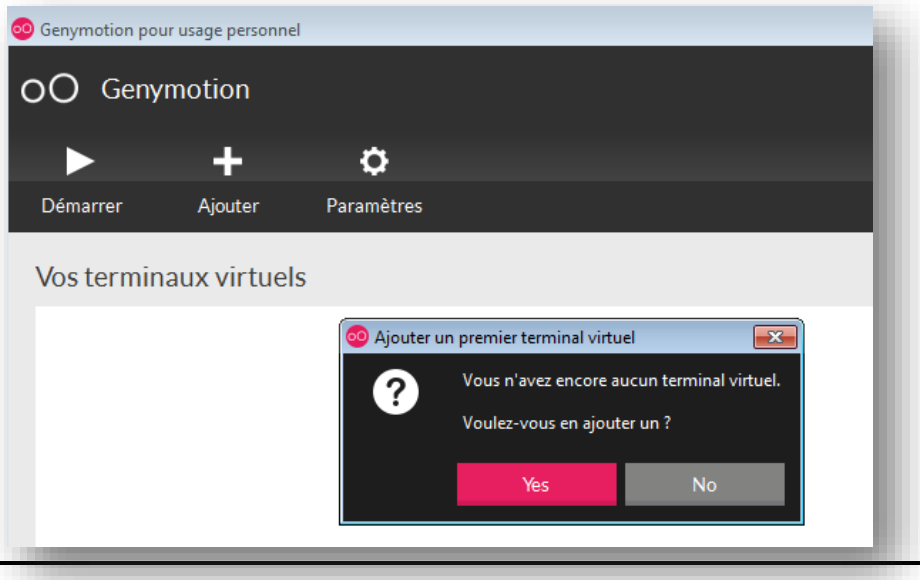
https://www.genymotion.com/

L'émulateur Genymotion s'exécute sur une machine virtuelle créée par **Virtual Box** téléchargeable à partir du site suivant :

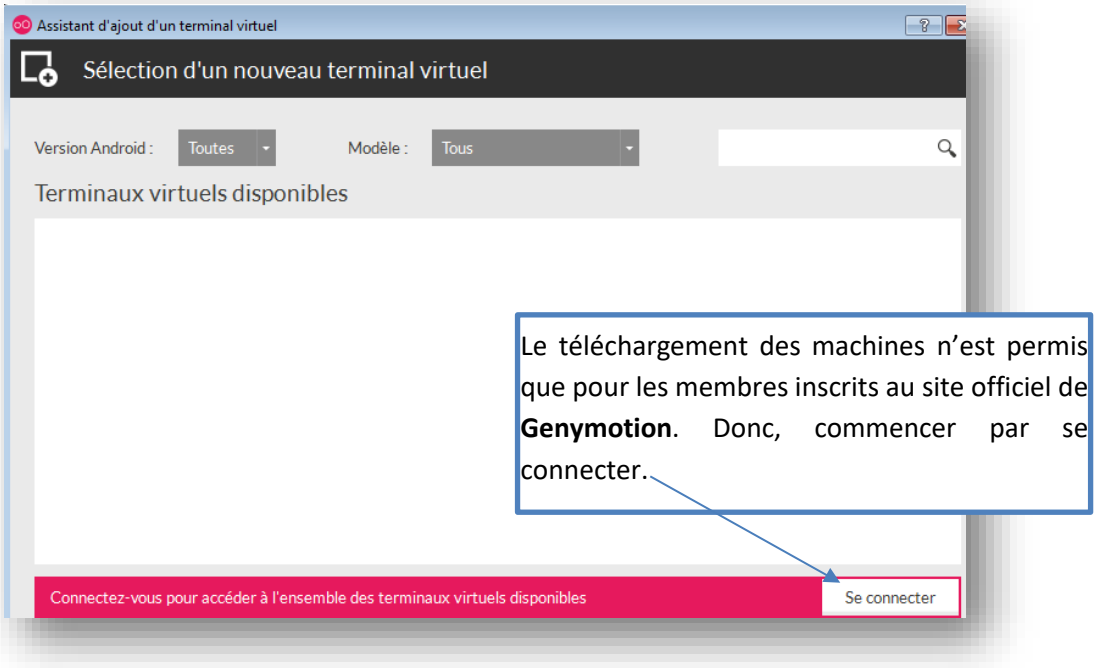
https://www.virtualbox.org/wiki/Downloads

Une fois **Genymotion** est installé, on devra y ajouter des machines virtuelles pour simuler l'exécution des applications Android.

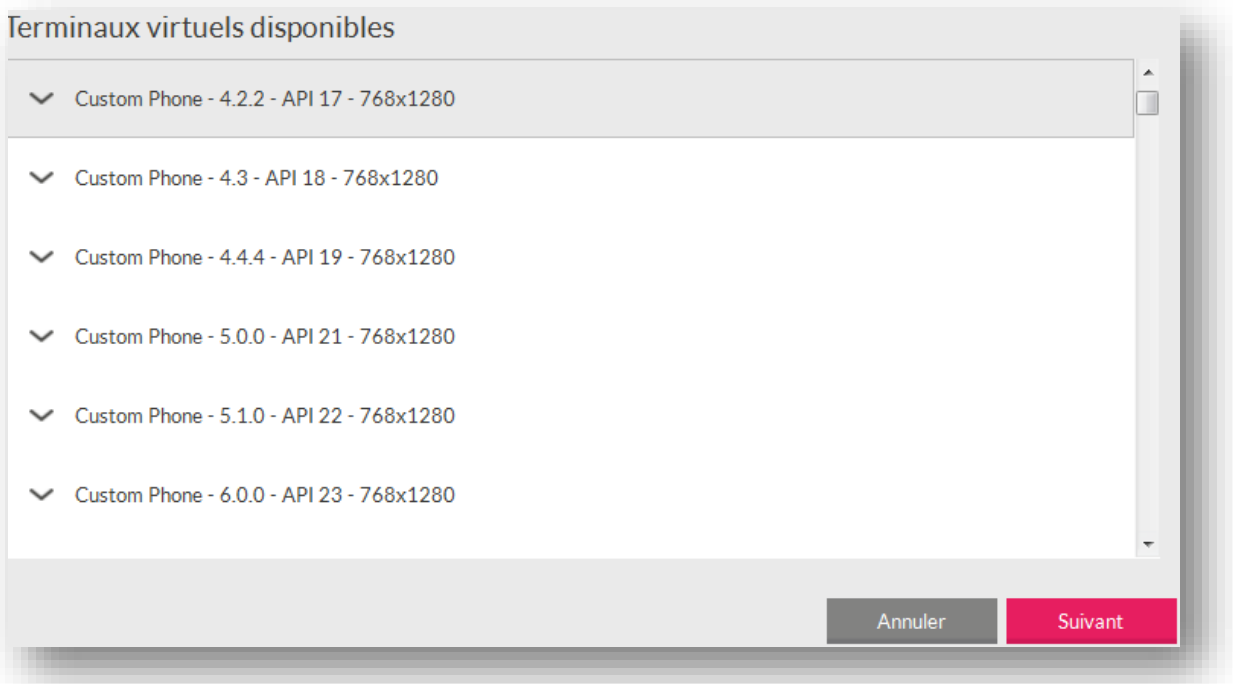
Dès qu'on démarre **Genymotion**, le message d'invitation de téléchargement des machines virtuelles apparait.



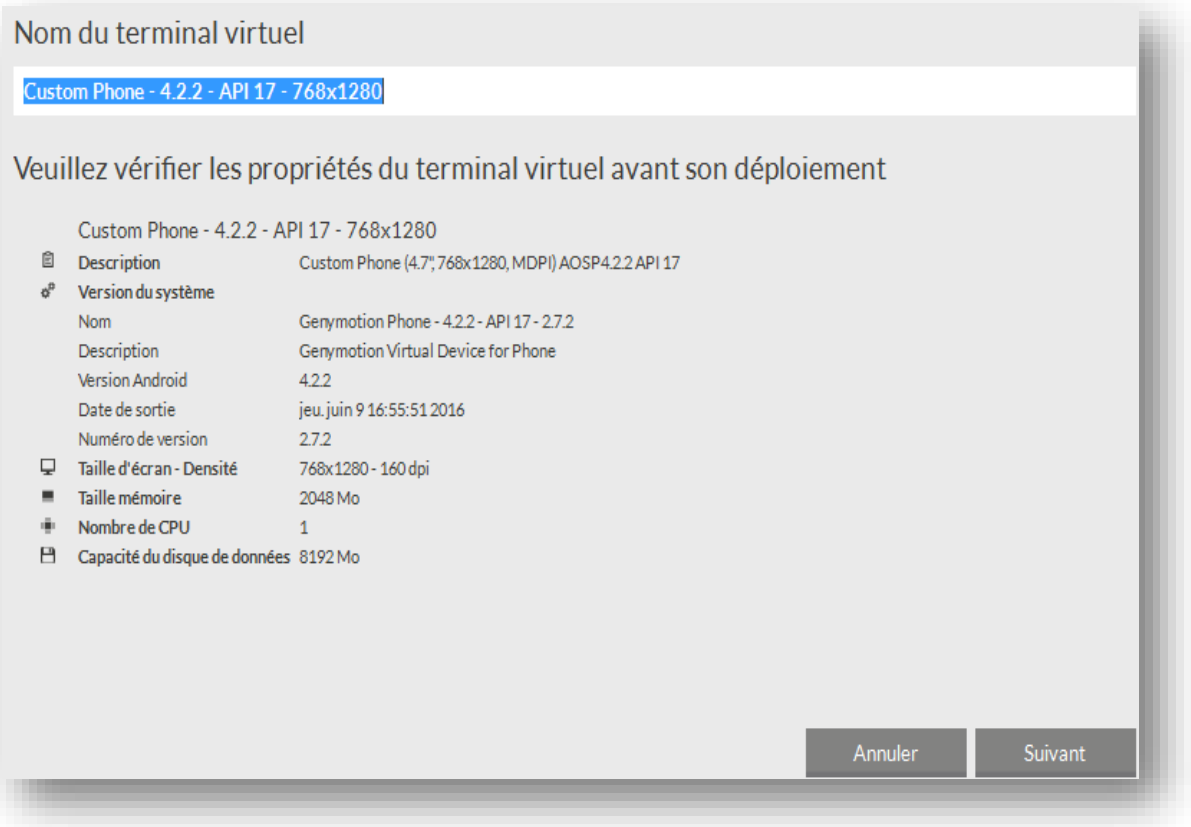
Le clic sur le bouton **Yes** mène à l'écran de sélection des machines virtuelles à installer dans **Genymotion**.



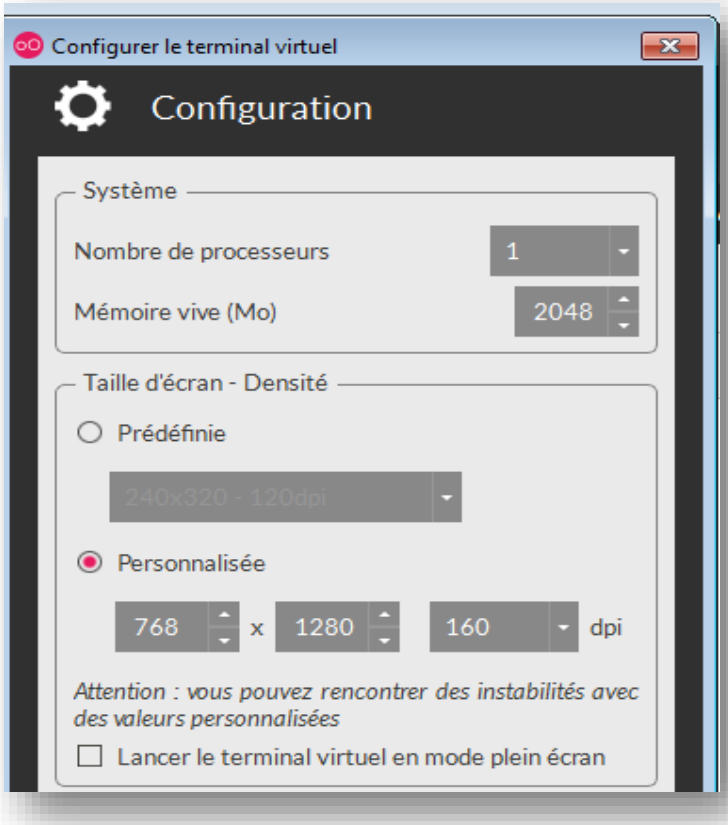
Pour ajouter des machines virtuelles, on commence par choisir l'appareil de simulation :



Ensuite, on vérifie les caractéristiques de l'appareil sélectionné :



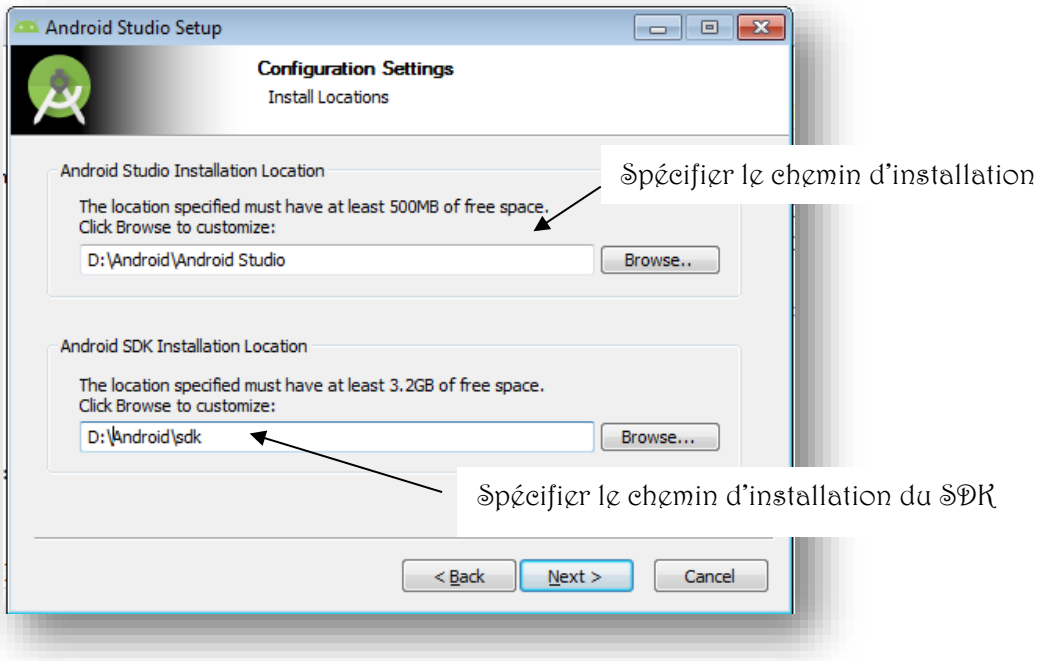
Finalement, on procède à la configuration de la dimension, la RAM et le nombre de processeurs alloués à la machine virtuelle choisit.



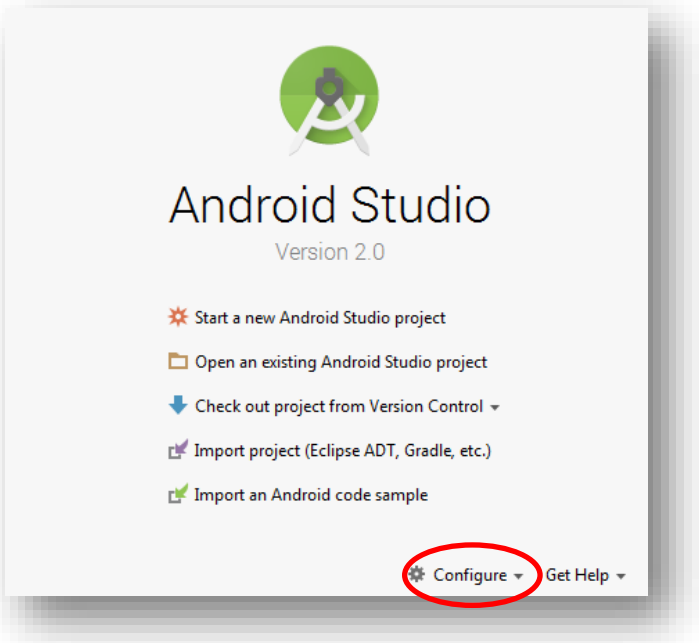
3- Installation de l'éditeur Android Studio

Télécharger et installer un environnement de développement : On a choisi comme éditeur **Android Studio** qu'on pourra le télécharger à partir du lien suivant : https://developer.android.com/studio/index.html

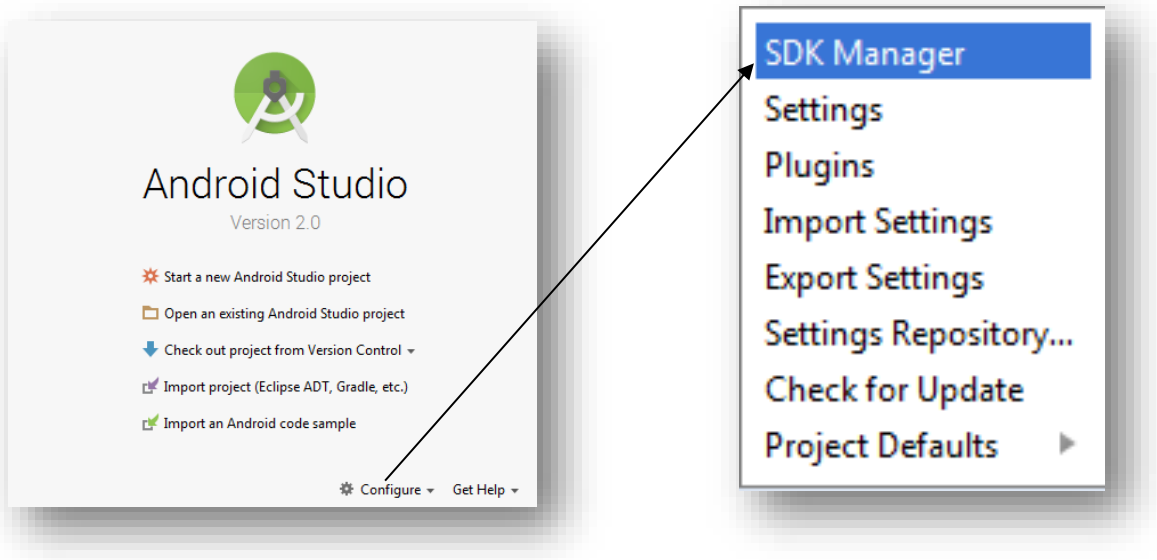
A cette étape, on est appelé à spécifier le chemin d'installation de l'IDE ainsi que celui du SDK.



Une fois l'IDE Android studio est installé et avant même de commencer le développement, on procède à sa configuration essentiellement celle du kit de développement (SDK).

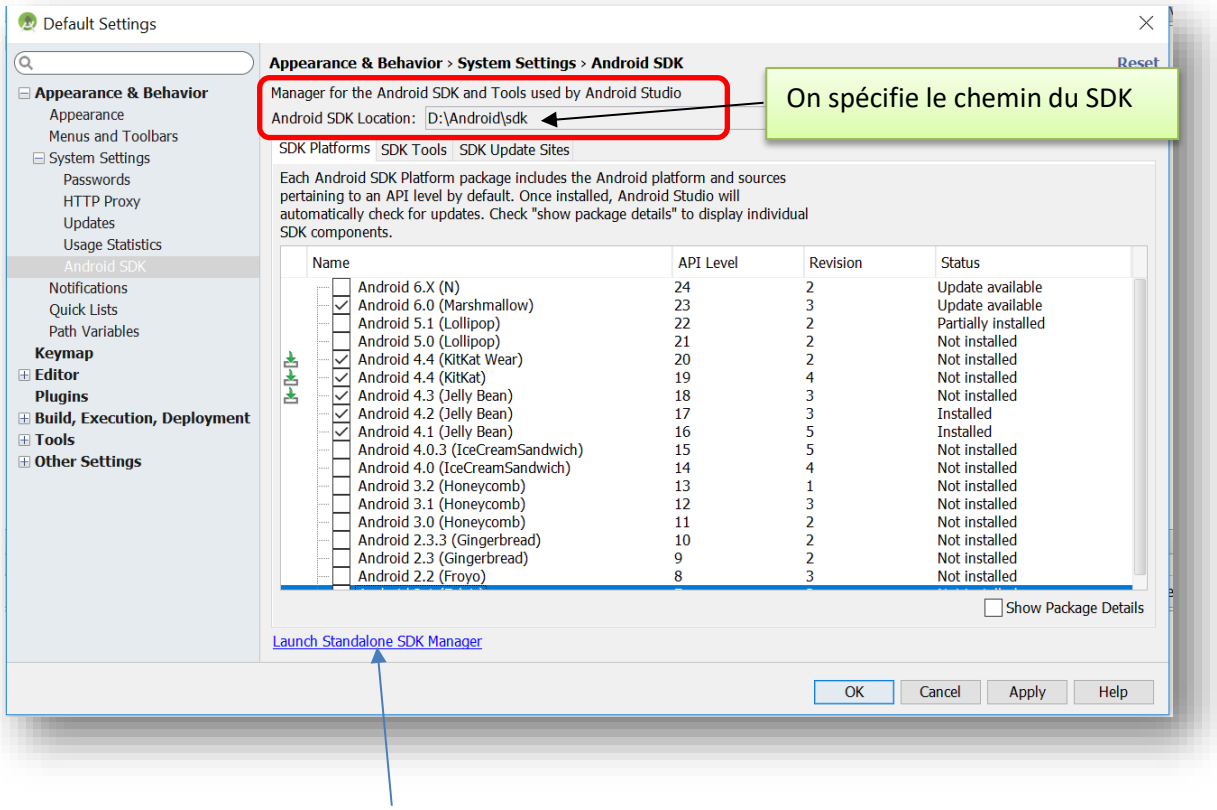


a. Configuration du SDK



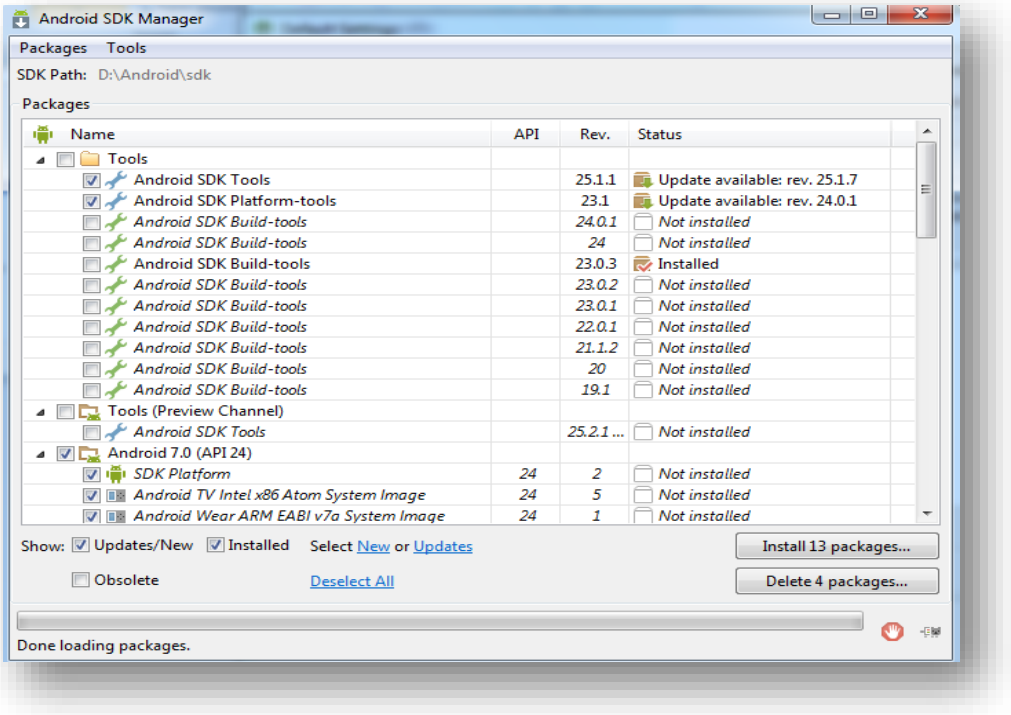
a-1- Installer l'API

Une API (Application Programming Interface) est un ensemble de classes regroupant des fonctions mises à la disposition des développeurs.



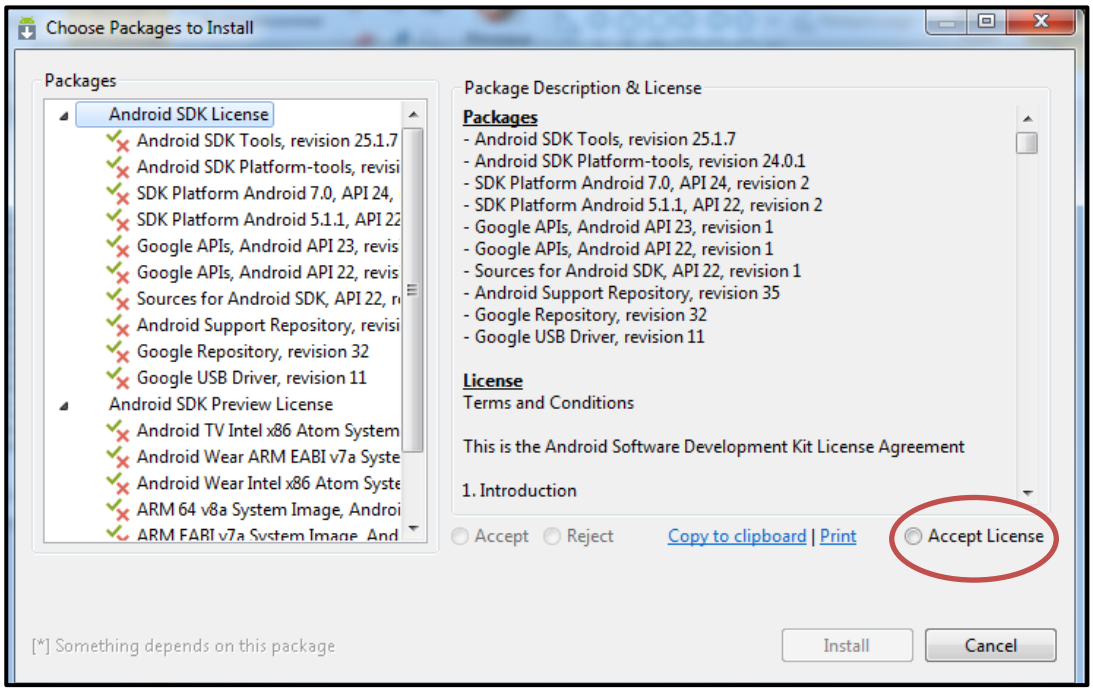
Le clic sur le bouton "**Launch Standalone SDK Manager**" permet l'ouverture de l'écran suivant :

a-2- Choisir et installer les bibliothèques de développement

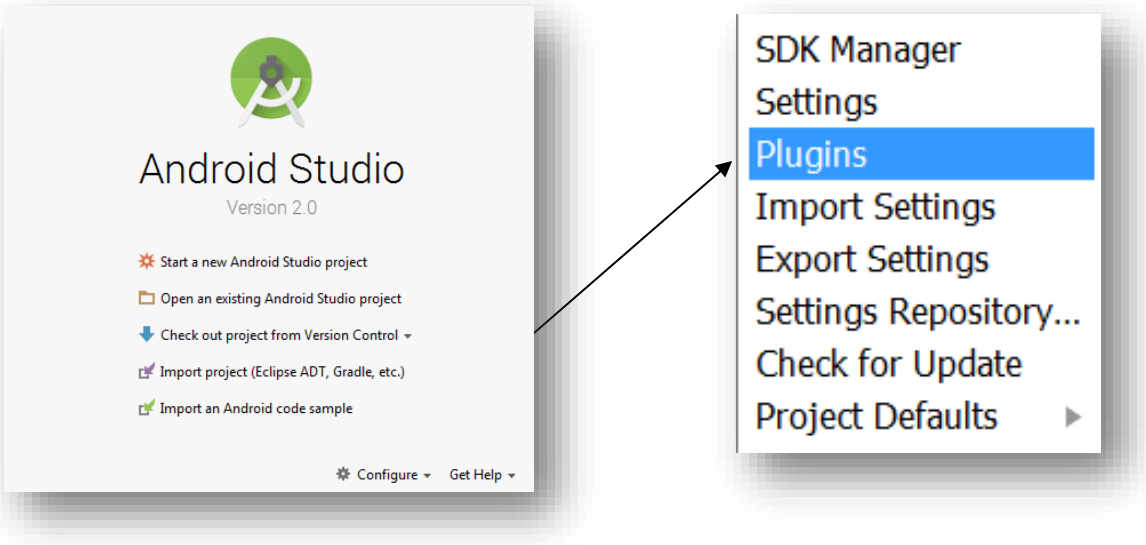


a-3- Licence d'utilisation

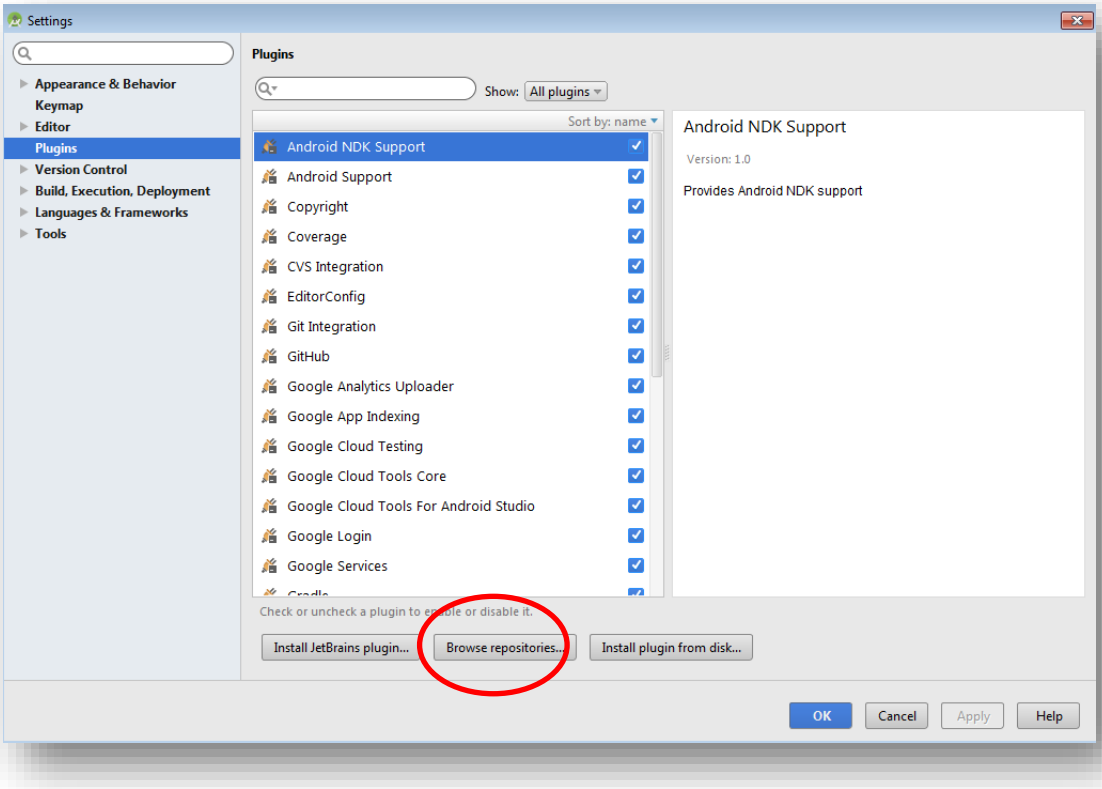
Accepter la licence d'utilisation des bibliothèques à installer afin de lancer l'installation.



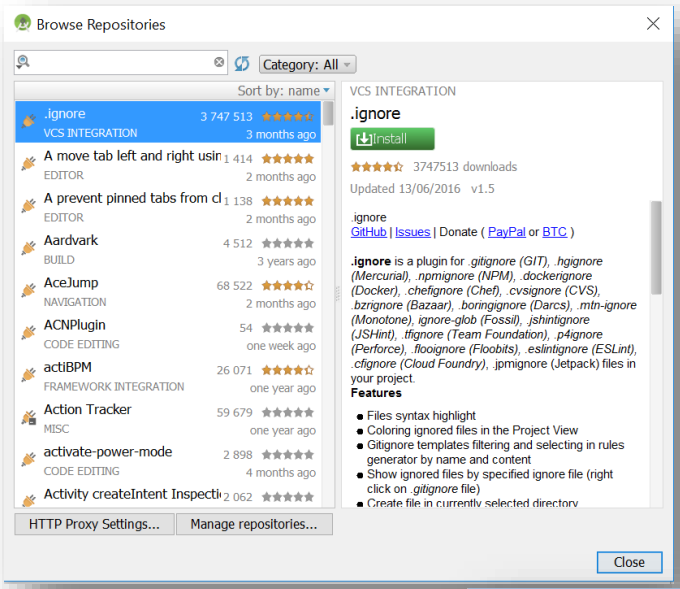
b. Installer les plugins



Pour intégrer un plugin au logiciel Android Studio, on doit le chercher dans la liste disponible ou bien cliquer sur le bouton "Browse repositories".

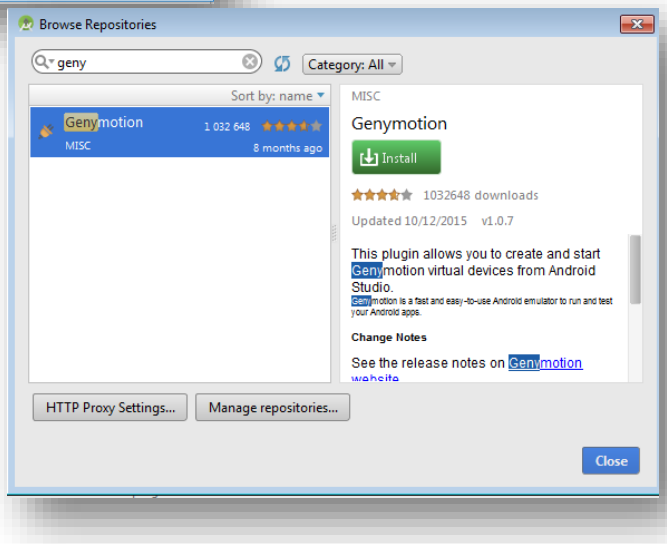


Pour installer un plugin :

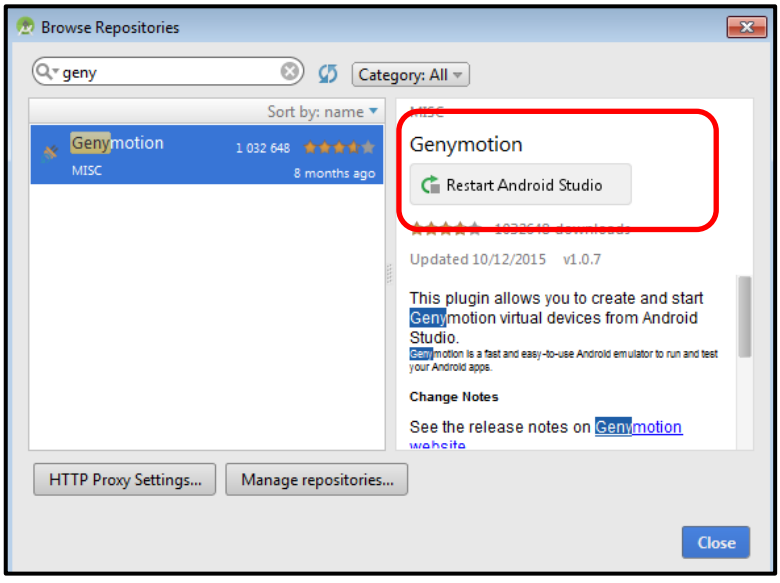


soit on cherche le nom du plugin à installer de la liste des dépôts.

soit on tape le nom du plugin dans la zone dédiée aux recherches.



Exemple d'installation du plugin **Genymotion**.

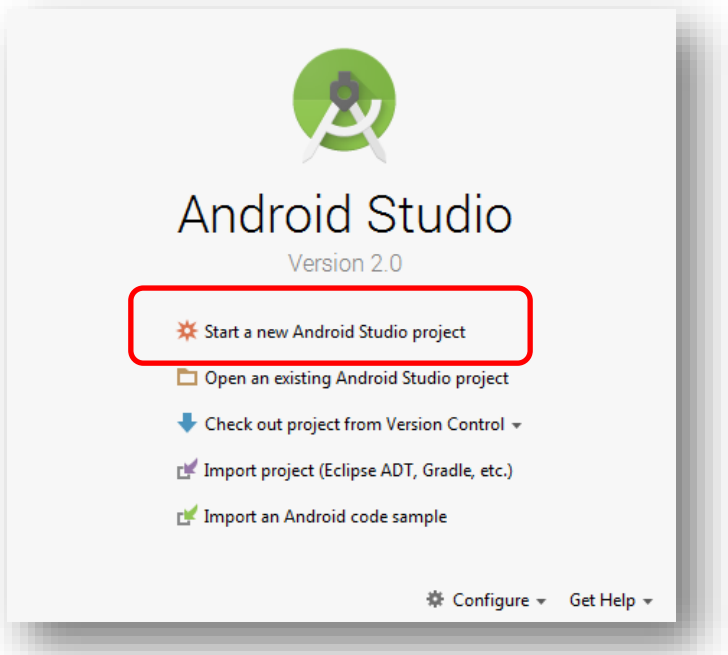


Achever l'installation du plugin par le clic sur "Restart Android Studio"

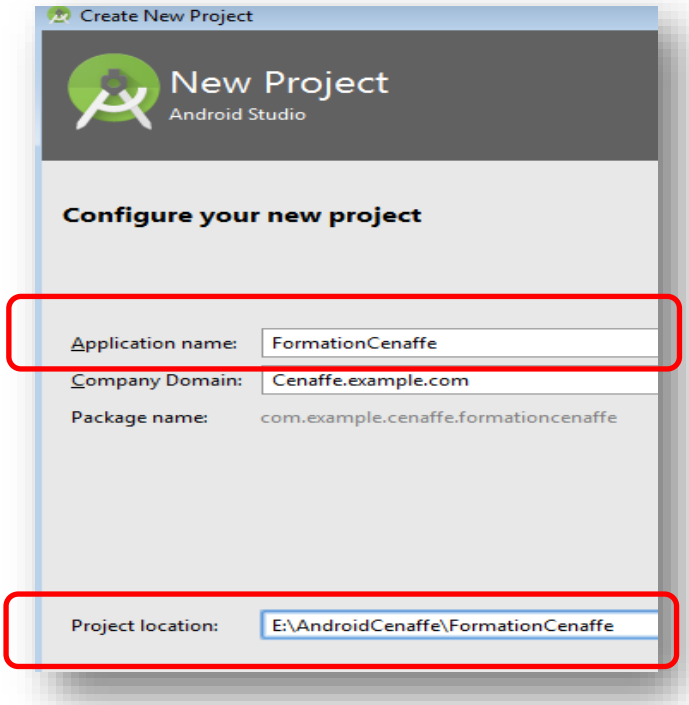
III- Application basique

1- Création d'un nouveau projet

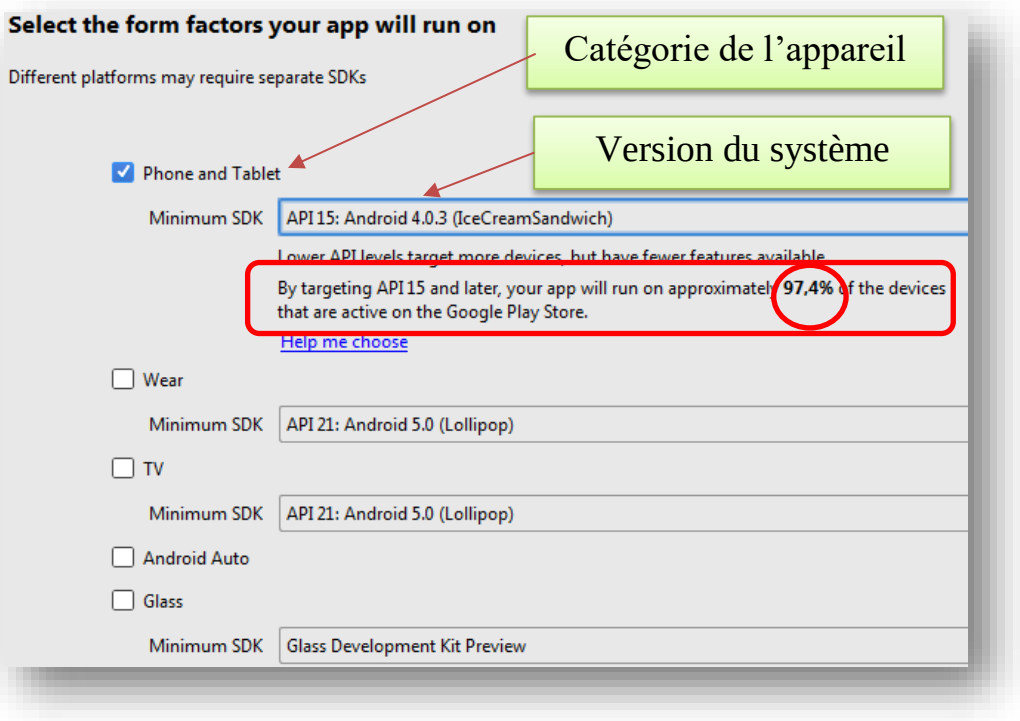
Activité : Créer une nouvelle application intitulée "**FormationCenaffe**".
Solution :
L'écran d'accueil d'Android Studio propose un certain nombre d'options, à savoir la création d'un nouveau projet, ouverture ou importation d'un projet existant.
On va choisir l'option "**Démarrer un nouveau projet Android Studio**".



1ère étape : Spécification du nom et emplacement du projet.
Un écran de configuration apparait permettant au développeur de spécifier le nom et l'emplacement du nouveau projet.



2ème étape : Spécification de l'API de l'application.
La catégorie d'appareils succeptible d'exécuter l'application à développer ainsi que la version de la bibliothèque à exploiter lors du développement doivent être spécifieés.

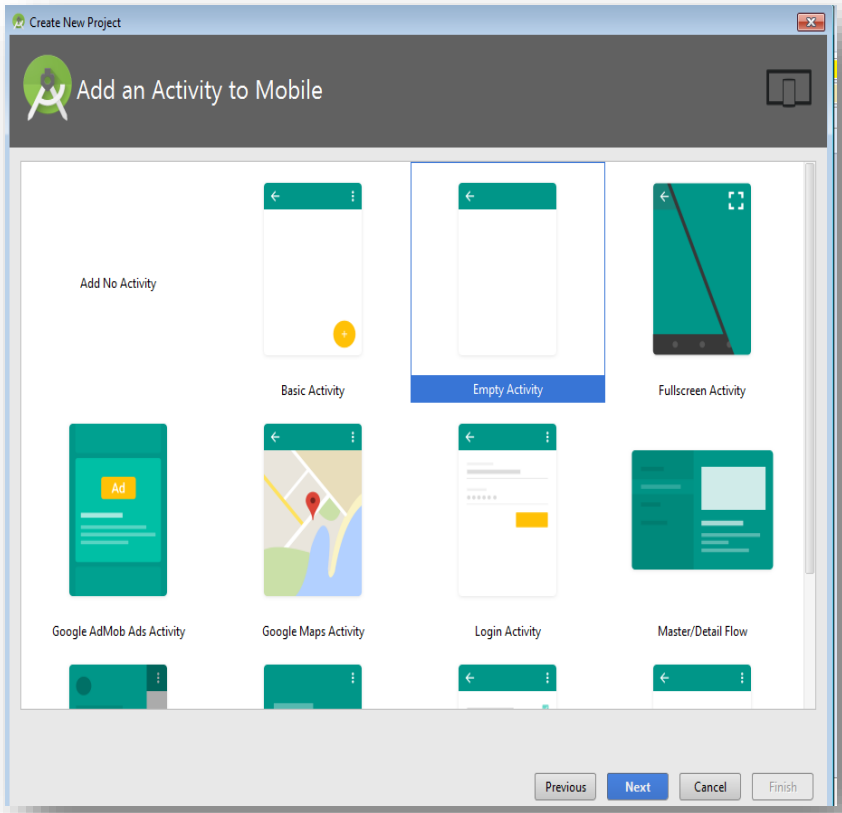


Dans l'exemple ci-dessus, on a choisi de développer une application exécutable sur des téléphones ou tablettes munis au moins de la version 4.0.3 du système d'exploitation Android.
On a sélectionné cette version d'API vu que 97,4 % des appareils sont munis de cette version du système Android mais en contre partie on ne peut pas intégrer, dans l'application, les fonctionnalités avancées d'Android qui sont apparûes dans les versions ultérieurs à 4.0.3.

3ème étape : Spécification du modèle de l'activité principale.

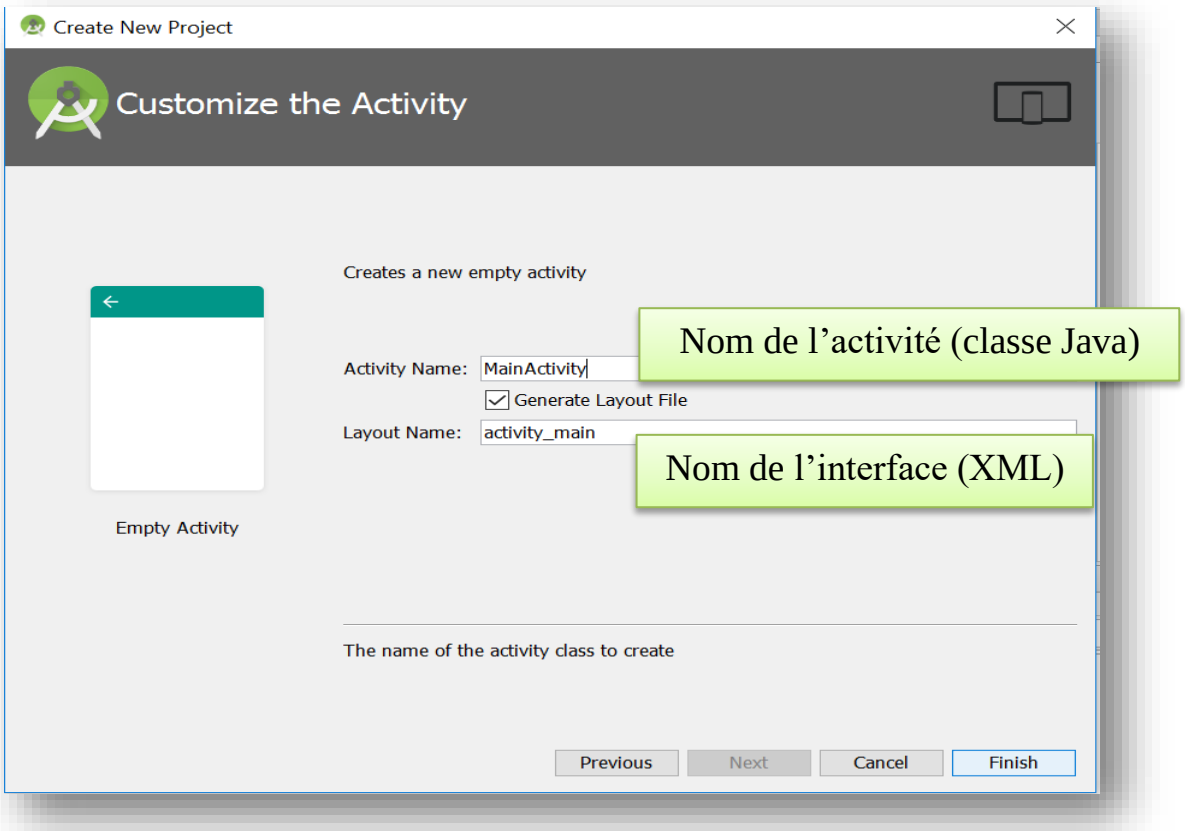
Une "**Activity**" est une classe Java décrivant une partie de l'application présentant une vue à l'utilisateur.

A cette étape, Android Studio propose aux développeurs une série de modèles d'activités prédéfinies qu'on peut se servir pour créer l'activité.



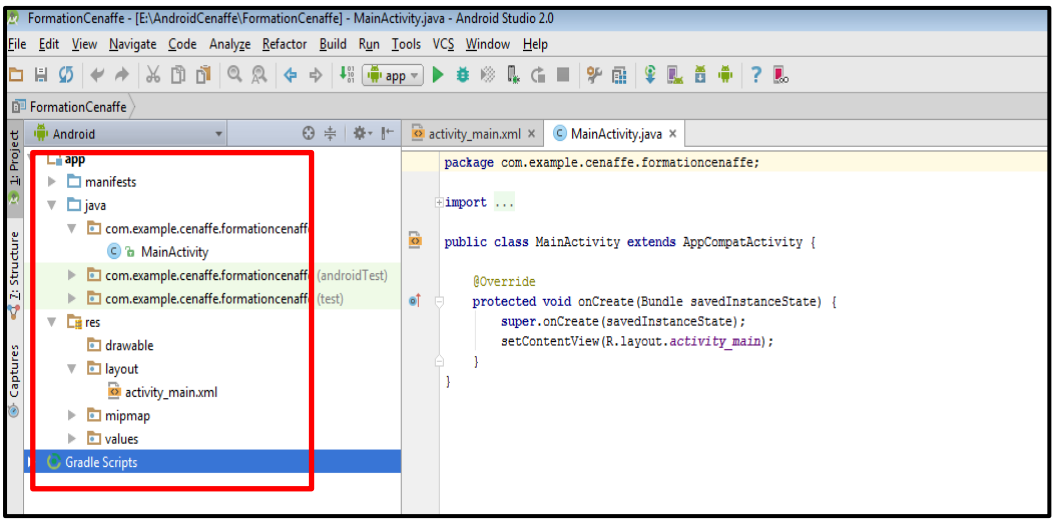
On a choisi une activité vide (Empty Activity) pour notre application de base.

4ème étape : Spécification des noms de l'activité et celui du layout de l'application.



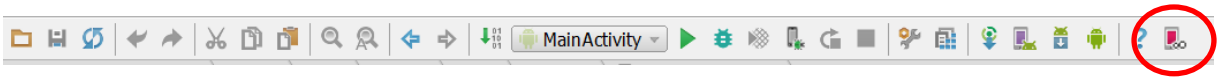
5ème étape : Validation et création de l'application.

En validant les choix de la fenêtre précédente, Android Studio crée une nouvelle application ainsi qu'un certain nombre de dossiers à savoir **manifests**, **java**, **res**, **drawable**, **layout**, **values**, etc.

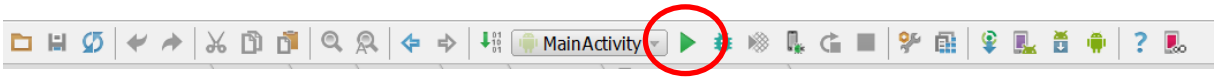


2- Exécution et simulation

L'émulateur permet de visualiser l'exécution d'une application sur une machine virtuelle donc il est primordiale de s'assurer de son fonctionnement.
On a intégré le plugin Genymotion dans notre environnement de développement alors son lancement doit se faire à travers l'icône  de la barre des raccourcis.

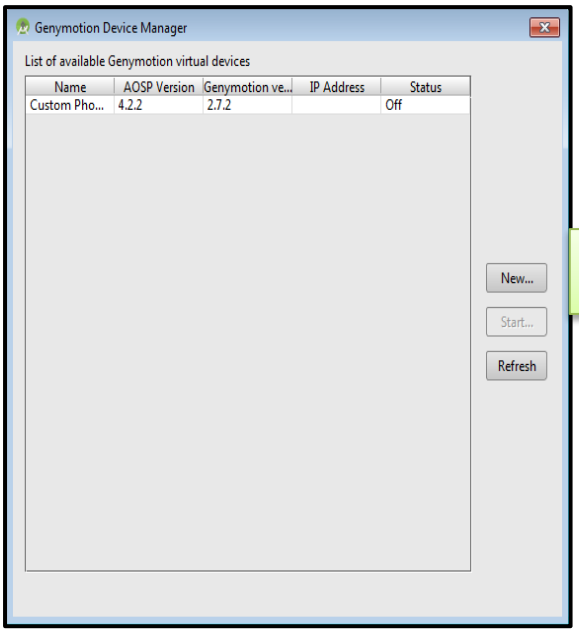


Tandis que l'exécution doit se faire avec l'icône  de la barre des raccourcis.

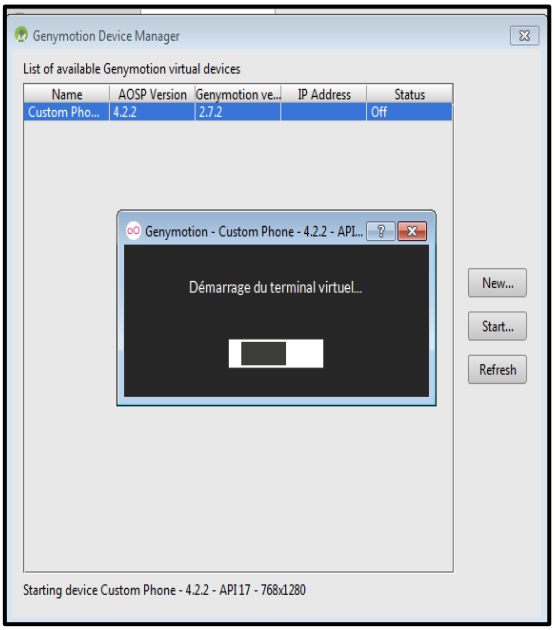


a. <u>Lancement de Genymotion</u>

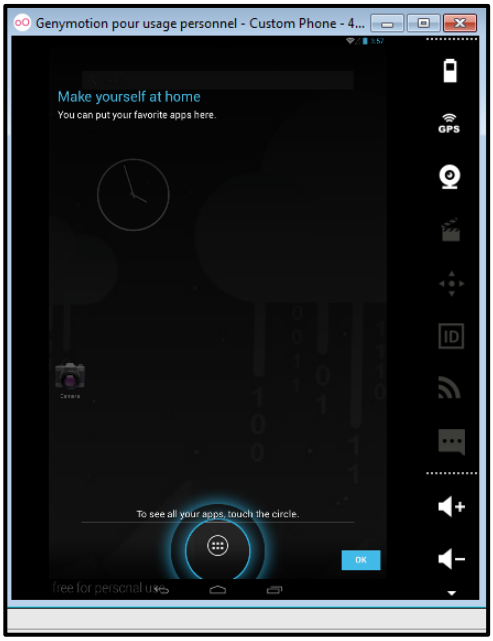
Une fois l'icône de Genymotion est cliquée, la liste des appareils virtuels installés apparaît en invitant le développeur de choisir une.



1-Sélectionner une machine virtuelle.



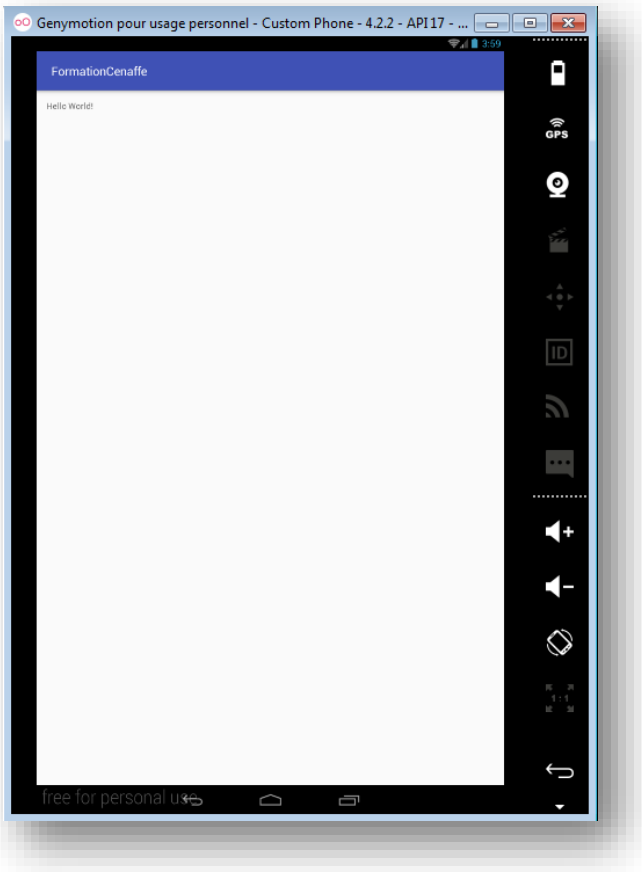
2-Choisir le bouton "Start".



3-Lancement de l'appareil virtuel

b. Exécution de l'application

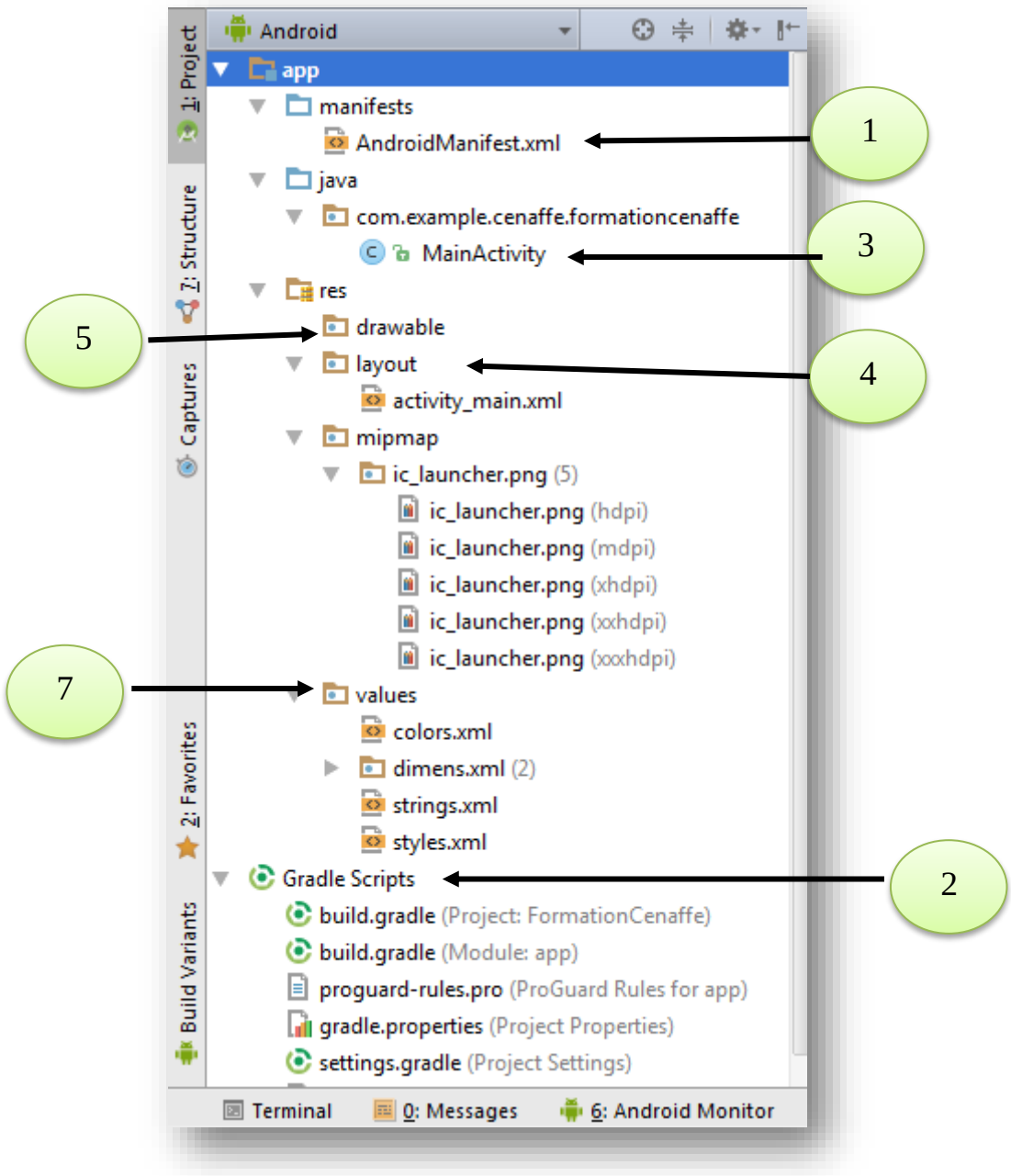
Une fois la machine virtuelle est lancée, le clic sur le raccourci d'exécution permet le lancement de l'application sur la machine virtuelle déjà démarrée.



IV- Structure d'un projet Android

Une application Android est un ensemble de fichiers valides selon une structure de dossier particulière.

Android Studio défini toute l'arborescence nécessaire lors de l'étape de création d'une nouvelle application.



1- Le fichier "AndroidManifest.xml"

Toute application Android contient un fichier spécial intitulé "**AndroidManifest.xml**". Il contient toutes les informations essentielles sur l'application, telles que les activités qu'elle contient, les bibliothèques nécessaires, les permissions, etc.

Le contenu du fichier **AndroidManifest.xml** d'une application de base est similaire à l'aperçu ci-dessous.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.cenaffe.formationcenaffe">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        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>
```

Interprétations :

- La première ligne définit la nature du fichier : XML
- La deuxième ligne définit les normes à respecter dans ce fichier. L'attribut **xmlns** indique le lien des normes : "**http://schemas.android.com/apk/res/android**".
- La troisième ligne spécifie le dossier de stockage de l'application (déjà spécifié lors de l'étape de la création de l'application)
- L'attribut "**allowBackup**" détermine si les données d'une application peuvent être sauvegardées et restaurées.
- L'attribut **icon** spécifie l'icône de l'application, par défaut Android Studio propose l'image "**ic_launcher.png**" du dossier **res\mipmap.**
- L'attribut "**label**" permet de spécifier le nom à afficher sur la barre des titres de l'application. Le label peut être soit la valeur d'une constante, soit la valeur d'une variable déclarée dans le fichier "**strings.xml**" du dossier **res\values.**

  Dans notre cas, on a précisé lors de l'étape de création du projet que l'application aura comme nom "**FormationCenaffe**", cette valeur a été enregistrée dans une variable intitulée "**app_name**" dans le fichier "**strings.xml**" du dossier **res\values.**
- L'attribut "**supportsRtl**" précise si les fenêtres de l'application supportent l'option "**right-to-left**"

- L'attribut **theme** décrit les apparences de l'application. Par défaut, **AppTheme** du dossier **res\style** a été attribué à notre application.
- Chacune des activités de l'application doit être décrite dans une balise "**activity**" où :
  - L'attribut **android:name** précisie le nom de l'activité ("MainActivity" dans notre cas). Le point "**.**" qui précède le nom montre qu'il est placé dans le dossier parent de l'application.
  - La valeur **android.intent.action.MAIN** signifie que cette activité est le point d'entrée de l'application.
  - La valeur **android.intent.category.LAUNCHER** signifie que cette activité sera lancée au démarrage de l'application.

Remarque :
Toutes les **activités** devront être déclarées dans le fichier **AndroidManifest.xml**. Si une **activité** n'est pas déclarée, alors le système ne saura pas qu'elle existe et elle ne sera jamais exécutée.

2- Les fichiers "Gradle"

Gradle est un outil, intégré dans Android Studio, qui permet de gérer la construction d'un projet utilisant plusieurs modules et les dépendances des librairies.
Android Studio crée les fichiers **gradles** nécessaires dont on cite principalement :

- **setting.gradle** est placé à la racine du projet Android Studio et il contient la liste des modules dont **gradle** doit gérer. Il contient pour le moment l'unique instruction : include **':app'**
  Si une application contient 2 modules : **app1** et **app2**, alors le fichier **setting.gradle** contiendra : **include ':app1', ':app2'**

- **build.gradle,** est le fichier principal et il contient la liste des informations partagées entre tous les modules de l'application (voir exemple ci-dessous).

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 24
    buildToolsVersion "24.0.1"

    defaultConfig {
        applicationId "com.example.cenaffe.formationcenaffe"
        minSdkVersion 15
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:24.1.1'
}
```

A chaque application développée, Android Studio lui crée un fichier **build.gradle** afin de décrire les options de compilation ainsi que la liste de ses dépendances. Dans ce fichier, on trouve essentiellement :

1. Le numéro de version d'Android Sdk utilisée pour compiler le projet : **compileSdkVersion**
2. Le nom complet de la version d'Android Sdk utilisée pour compiler le projet : **buildToolsVersion**
3. L'identifiant unique de l'application (le nom du package complet) : **applicationId**
4. La version minimale d'Android supportée : **minSdkVersion**
5. La version d'Android pour laquelle l'application sera compilée : **targetSdkVersion**
6. Le numéro de la version de l'application : **versionCode**
7. Le nom complet de la version : **versionName**
8. Les projets peuvent avoir des **dépendances** sur d'autres composants. Ces composants peuvent être des paquets binaires externes, ou d'autres projets Gradle.
    - On peut ajouter une dépendance sur tous les pots à l'intérieur du répertoire libs. **compile fileTree(dir: 'libs', include: ['*.jar'])**
    - On peut ajouter des dépendances vers des liens externes. **Exp : compile 'com.android.support:appcompat-v7:24.1.1'**

3- Le fichier MainActivity.java

Pour l'application "**FormationCenaffe**" qu'on a créée, on a attribué le nom **MainActivity** à l'activité principale. Android Studio nous a créé une classe Java ayant le contenu suivant :

```
1   package com.example.cenaffe.formationcenaffe;
2
3   import android.support.v7.app.AppCompatActivity;
4   import android.os.Bundle;
5
6   public class MainActivity extends AppCompatActivity {
7
8       @Override
9       protected void onCreate(Bundle savedInstanceState) {
10          super.onCreate(savedInstanceState);
11          setContentView(R.layout.activity_main);
12      }
13  }
```

Interprétations :

Ligne 1 : le mot réservé "**package**" décrit le dossier de l'application.

Lignes 3 et 4 : le mot "**import**" spécifie les bibliothèques à exploiter dans cette activité.

Ligne 6 : déclaration de l'entête de la classe **MainActivity** qui hérite des méthodes de la classe **AppComptaActivity.** Parmi les méthodes héritées, on trouve : **onCreate**, **setContentView**, etc.

Pour visualiser toutes les méthodes d'une classe, on clique sur son nom (Exemple : **AppCompatActivity)** toute en appuyant sur le bouton CTRL.

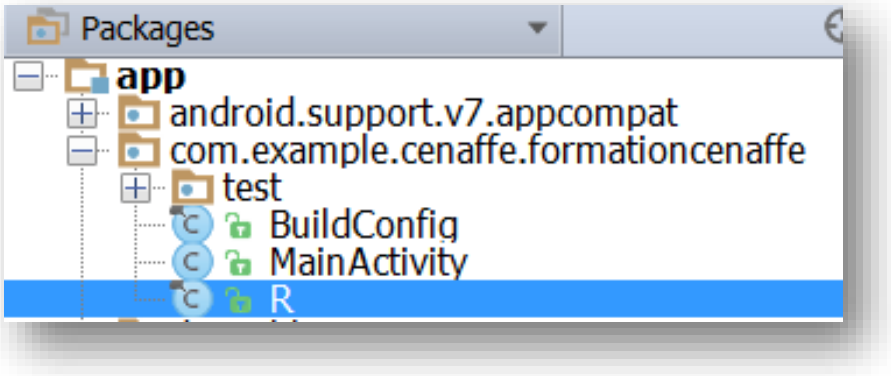
Ligne 8 : le mot "**@Override**" désigne que la classe **MainActivity** va redéfinir la méthode **onCreate** héritée de la classe **AppComptaActivity**.

Ligne 9 : La méthode **onCreate** s'exécute automatiquement dès la création de l'activité courante et elle a un seul paramètre de type **Bundle** (les éléments d'un écran).

Ligne 10 : le mot clé "**super**" désigne qu'on va utiliser la méthode **onCreate** de la classe **AppComptaActivity**

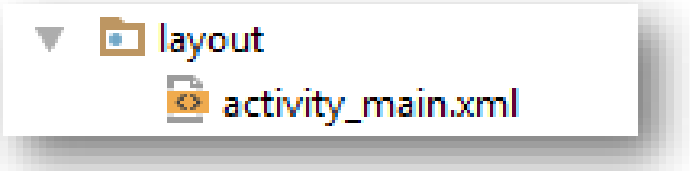
Ligne 11 : la méthode **setContentView**, à laquelle on a passé l'adresse d'un layout, permet de spécifier l'interface utilisateur appelée par la classe **MainActivity**.

Sachant qu'un **layout** est un fichier **xml**, placé dans le dossier **res\layout**, qui représente une interface graphique.
Dans un projet Android, il existe une classe java intitulée **R** réservée pour stocker les adresses des objets manipulés par l'application tels que les variables, les constantes, les éléments graphiques, les layouts, etc.



L'exécution de l'activité intitulée "**MainActivity**" fait appel à la méthode intitulée **onCreate,** héritée de la classe **AppComptaActivity,** qui sera lancée dès la création de l'activité. Cette méthode fait appel à son tour à la méthode **setContentView** pour visualiser l'interface graphique (layout) intitulé "**activity_main**" dont la description est détaillée dans le fichier **main_activity.xml**.

4- Le dossier Layout



Le dossier layout contient toutes les interfaces graphiques utilisées dans une application Android.
Un layout est un fichier xml contenant des composantes graphiques, intitulées **vue** ou **view**.
Par défaut, Android Studio crée un layout contenant le texte "**Hello world**" dont le contenu est le suivant :

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.example.cenaffe.formationcenaffe.MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:id="@+id/text1" />
</RelativeLayout>
```

Interprétations :

Ligne 1 : décrit la nature du fichier qui est de type Xml.

Ligne 2 : la balise RelativeLayout permet de spécifier la façon de disposer les éléments graphiques du gabarit principal (conteneur).

Lignes 3 et 4 : décrivent le chemin du modèle Xml à respecter.

Lignes 5… 10 : spécifient les propriétés du layout.

Ligne 11 : décrit la classe java utilisant ce layout.

Ligne 12 : la déclaration d'un composant graphique (vue) de type zone de texte intitulé **TextView**.

Lignes 13 ..16 : spécifient les propriétés de cette zone de texte.

5- Le dossier Drawable

Le dossier **Drawable** situé sous le dossier **res** est réservé pour stocker les images et les icônes à exploiter dans l'application. Ce dossier est créé vide par défaut sous Android Studio.

6- Le dossier Menu

Ce dossier contiendra tous les menus de l'application, il est placé sous le dossier **res**. Ce dossier devra être créé manuellement sous Android Studio.

7- Le dossier Values

Le dossier **Values** est placé sous **res** et il contient 4 fichiers de type **xml**:

1. **colors.xml** : où on définit toutes les constantes de type couleur
2. **dimens.xml** : où on définit toutes les constantes de type dimension
3. **strings.xml** : où on définit toutes les constantes chaines
4. **styles.xml** : où on définit tous les modèles de style à utiliser pour présenter les interfaces graphiques de l'application.

Remarque : on reviendra ultérieurement sur les contenus de ces fichiers.

V- Interface utilisateur

L'application de base précédemment créée, a permis l'affichage de la valeur d'un élément graphique de type texte. Cet élément graphique (vue) a été placé dans un conteneur intitulé "**gabarit**".

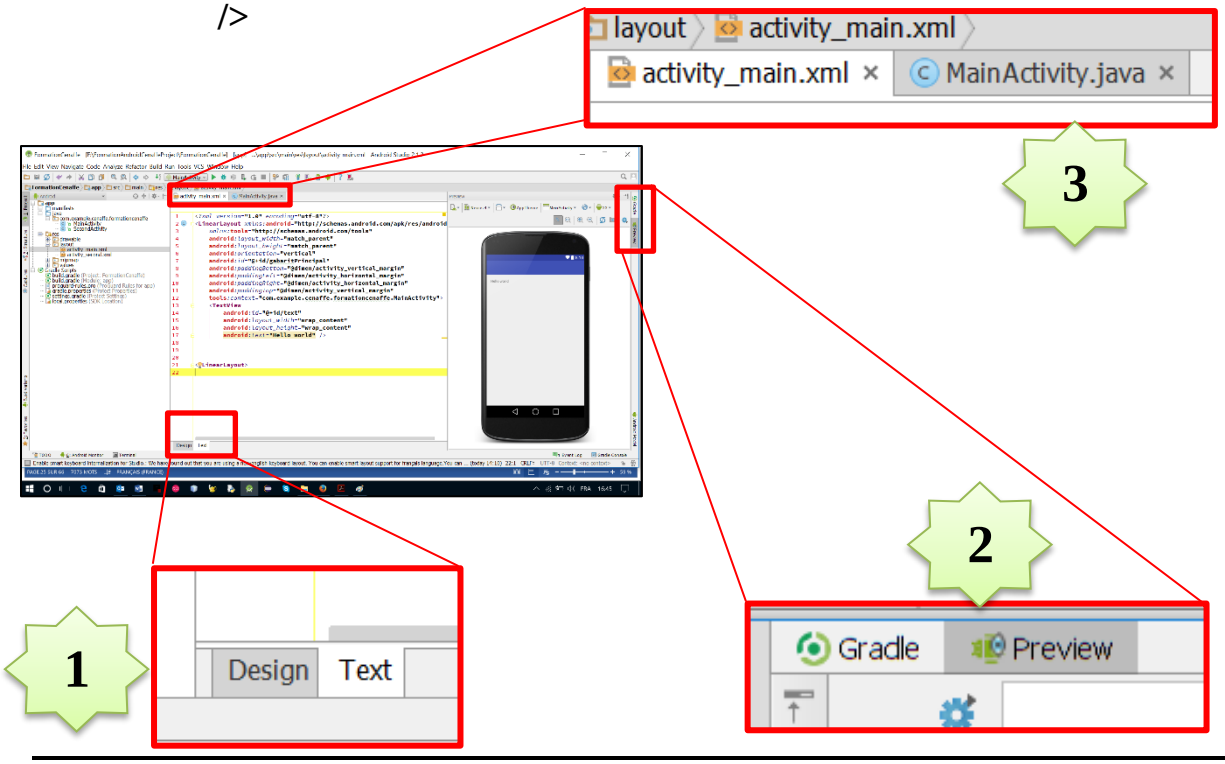
Android Studio permet la manipulation de l'interface utilisateur en deux modes à savoir le mode texte et le mode désigne.

1- Le mode texte

Dans ce mode, les déclarations et les attributs sont faites en langage Xml.

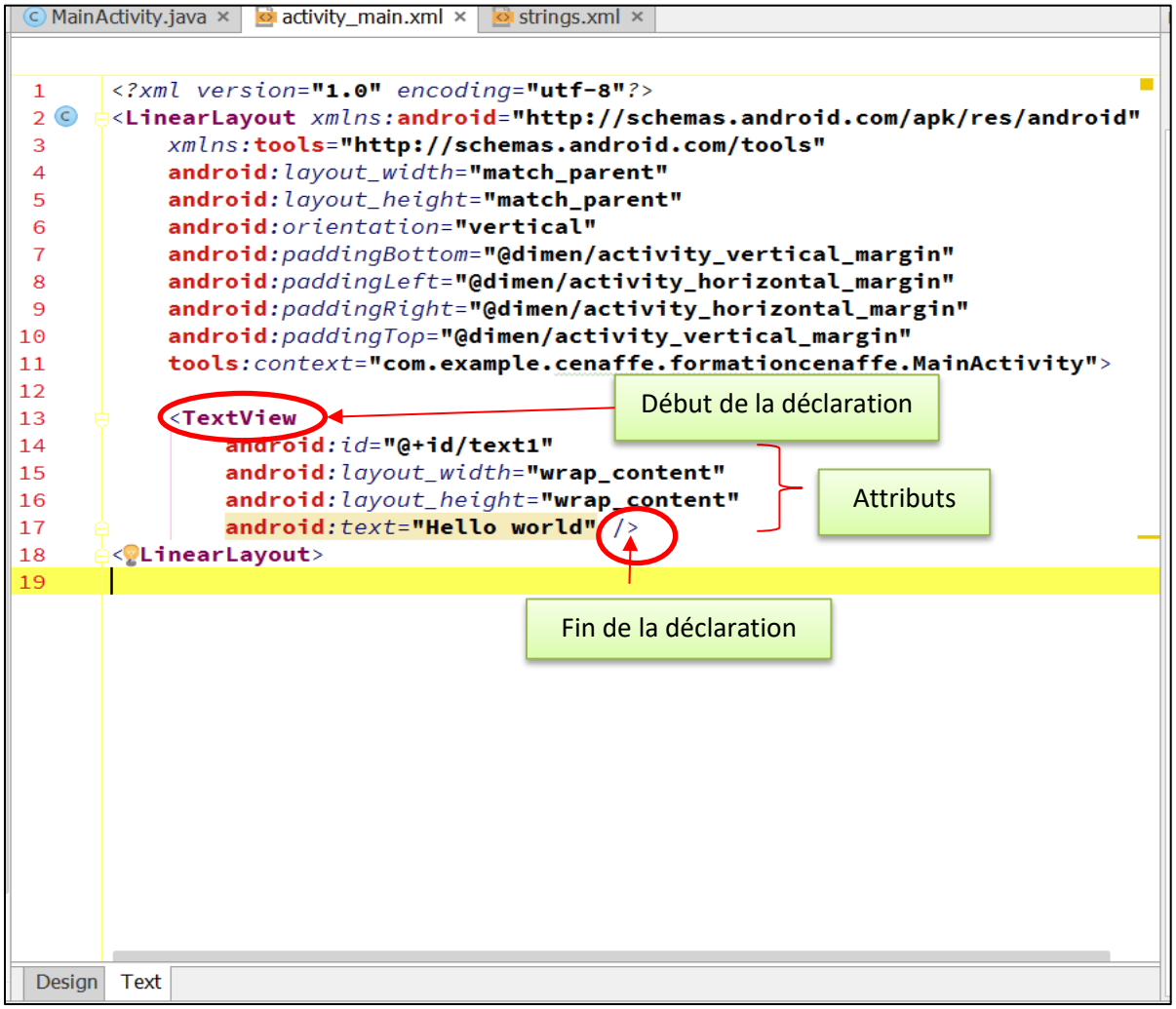
Chaque composant doit être déclaré comme suit :

<nom de la vue à déclarer
.....
.....
..... Liste des attributs
/>



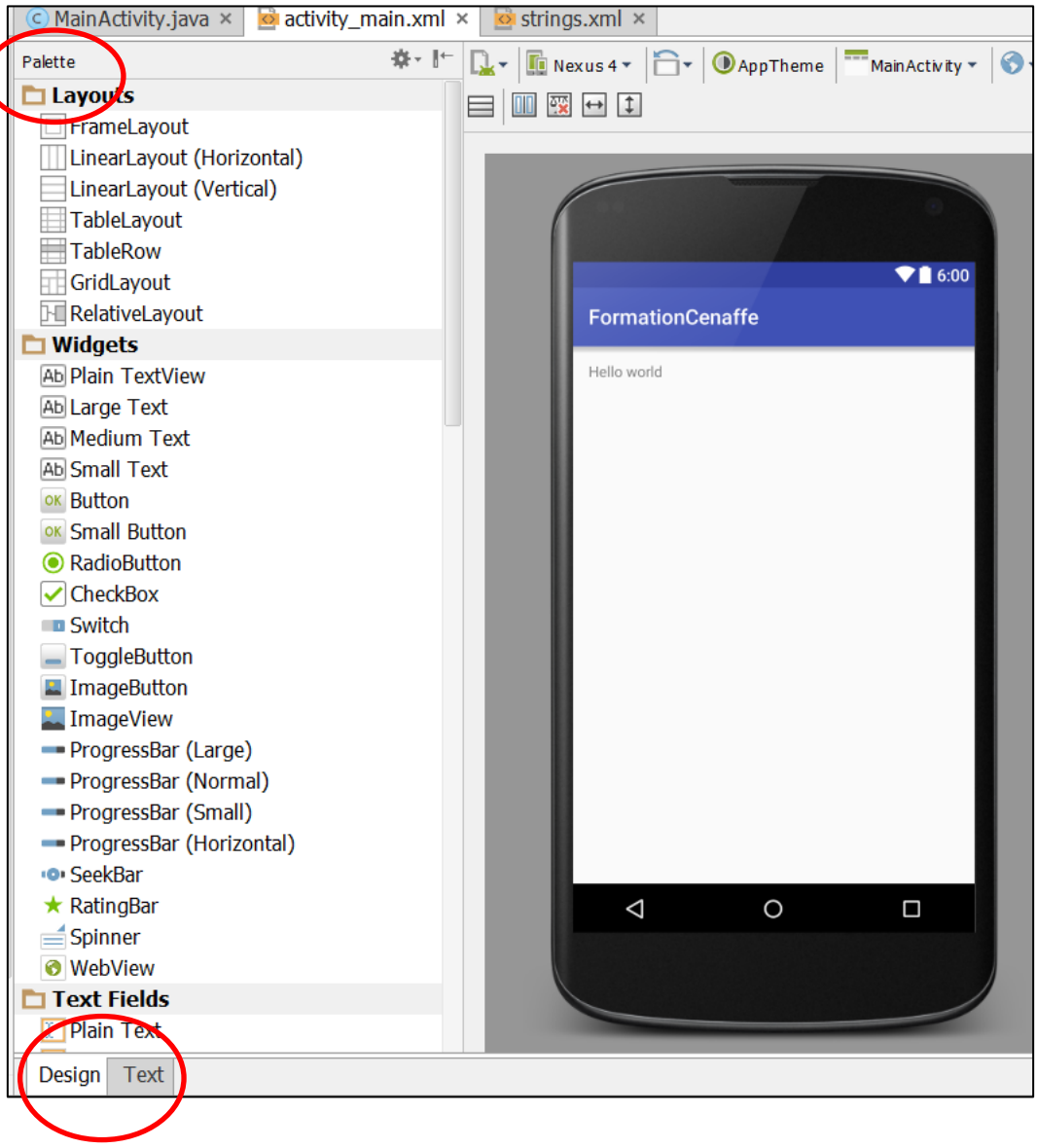
- De la **barre d'onglets (1)**, on peut passer du mode texte en mode graphique et inversement.
- En mode "Texte", on peut aussi visualiser l'interface graphique lorsque l'onglet "Preview" est activé de la **barre d'onglets (2)**.
- Lorsque le fichier xml n'est pas choisi de la **barre d'onglets (3)**, l'espace des aperçus devient invisible.

Le fichier "activity_main.xml" contient pour le moment les déclarations suivantes :



2- Le mode désigne

La conception de l'interface utilisateur est basée sur le principe de drag and drop des composants de la palette sur le simulateur de l'appareil téléphonique.



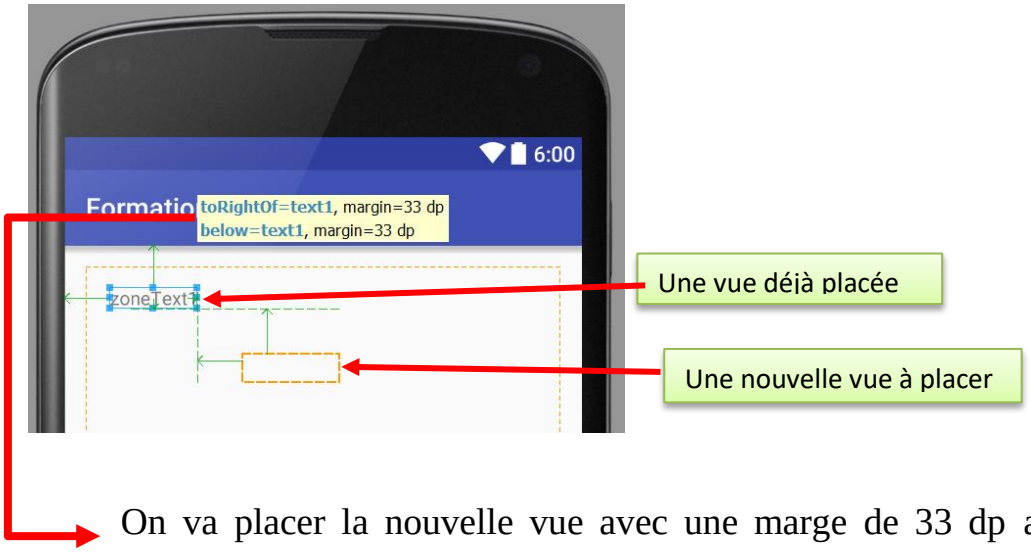
3- Gabarit

C'est un conteneur décrivant la manière de disposition des éléments graphiques qu'il contient. Il peut avoir l'une des dispositions suivantes :

a. RelativeLayout

Cette mise en page est basée sur les dispositions de relation pour placer les vues d'une interface graphique. On définit la position de chaque composant par rapport aux autres éléments ou bien par rapport au layout père.

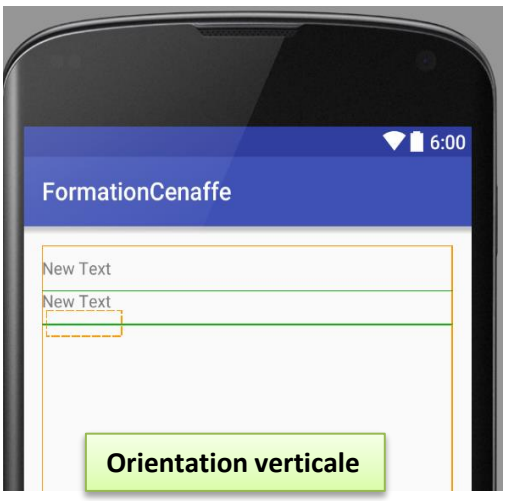
Exemple : L'écran ci-dessous illustre comment placer un nouvel élément relativement à la position d'un autre composant ou par rapport au gabarit global.

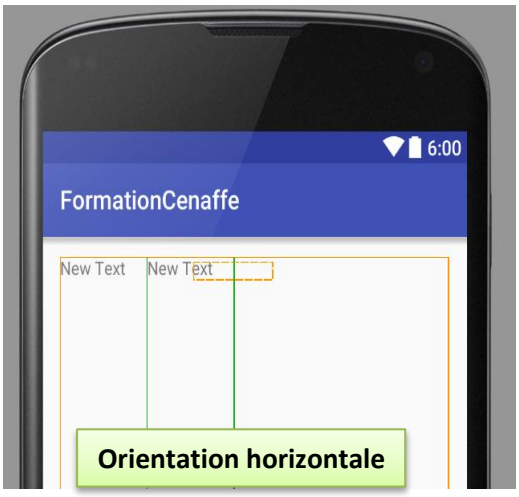


On va placer la nouvelle vue avec une marge de 33 dp au-dessous (below) et à droite (toRightOf) du composant intitulé text1.

b. LinearLayout

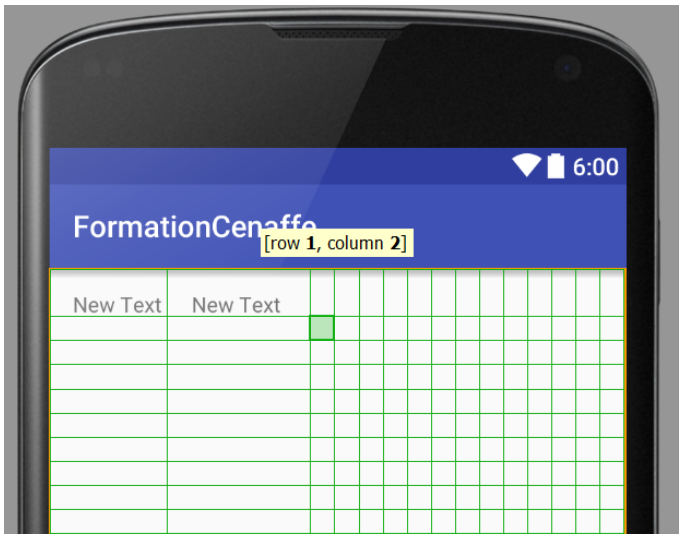
Cet ajustement permet de placer les vues verticalement ou horizontalement. Si l'ajustement est à la verticale alors les vues seront affichées dans une seule colonne, sinon elles seront affichées dans une seule ligne.





c. GridLayout

Cet arrangement divise l'écran en une grille de lignes, de colonnes et de cellules.



Activité : Déterminer la nature et les propriétés du layout "**activity_main**" définit pour l'application "**FormationCenaffe**".

Réponse :

Par défaut, Android Studio crée un layout ayant la description suivante :

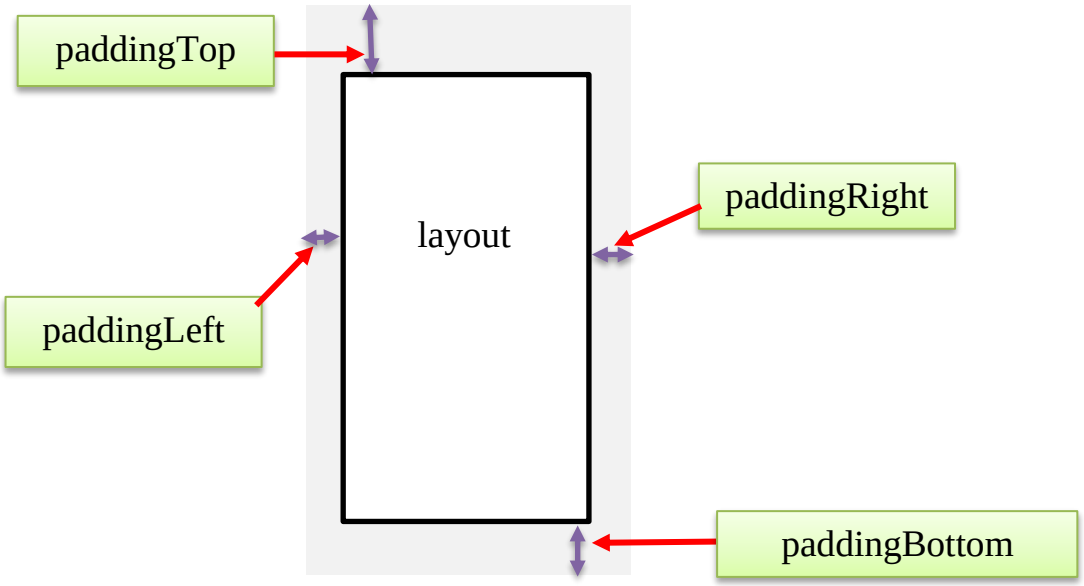
```
1   <?xml version="1.0" encoding="utf-8"?>
2   <RelativeLayout
3       xmlns:android="http://schemas.android.com/apk/res/android"
4       xmlns:tools="http://schemas.android.com/tools"
5       android:layout_width="match_parent"
6       android:layout_height="match_parent"
7       android:paddingBottom="16dp"
8       android:paddingLeft="16dp"
9       android:paddingRight="16dp"
10      android:paddingTop="16dp"
11      tools:context="com.example.cenaffe.formationcenaffe.MainActivity">
12      <TextView
13          android:layout_width="wrap_content"
14          android:layout_height="wrap_content"
15          android:text="Hello World!"
16          android:id="@+id/text1" />
17  </RelativeLayout>
```

Le conteneur des composants graphiques, gabarit, proposé par Android Studio est d'aspect **RelativeLayout** et ayant les caractéristiques suivantes :

1. **layout_width** : cette propriété spécifie la largeur de l'interface utilisateur et qui peut avoir l'une des 3 valeurs suivantes :
    - **match_parent** : la largeur du gabarit père c-à-d il va couvrir toute la largeur de l'appareil.

- **fill_parent** : la largeur du gabarit père (délaissé depuis l'Api < 8)
- **wrap_content** : la largeur minimale nécessaire à l'affichage

2. **layout_height** : pour spécifier la longueur du composant et on lui a attribué celle du gabarit père c-à-d il va couvrir tout l'écran de l'appareil.
3. **paddingTop** : pour définir la marge du haut par rapport à l'écran.
4. **paddingBottom** : pour définir la marge de bas par rapport à l'écran.
5. **paddingLeft** : pour définir la marge gauche.
6. **paddingRight** : pour définir la marge droite.



On a fixé une marge de 16 dp (dpi) de chaque côté (haut, bat, gauche et droite) pour le gabarit principal de ce layout.

Un survol de la souris sur n'importe quelle valeur de **padding** affiche l'image suivante :

```
android:paddingBottom="@dimen/activity_vertical_margin"
```

Ce qui signifie que la valeur "16dp" n'est en fait que la valeur d'une constante intitulée "**activity_vertical_margin**" d'attribut **dimen** déclarée dans le fichier "**dimens.xml**" du dossier "**res\values**".
Ci-joint un extrait du fichier **dimens.xml** :

```
1  <resources>
2      <!-- Default screen margins, per the Android Design guidelines. -->
3      <dimen name="activity_horizontal_margin">16dp</dimen>
4      <dimen name="activity_vertical_margin">16dp</dimen>
5  </resources>
```

4- Les composants d'un layout : vue ou view

Un élément graphique, intitulé vue, est un composant placé dans un gabarit permettant l'affichage d'une information. A Chaque vue, on dispose d'un certain nombre :

- **d'attributs** à gérer au format xml et ayant la syntaxe suivante :

  android:nom de l'attribut = "une valeur"

  Exemples : **android:text** = "Bonjour"

  android:textAllCaps = "true"

  android:height = "30dp"

- **de méthodes** à gérer dans le code programme et ayant le format suivant :

  nomVariable.nomMéthode (liste des paramètres) ;

  Exemples : var1.setText ("Bonjour") ;

  var1.setTextColor (Color.RED) ;

  var1.getId ( );

Une vue peut être une zone de texte (**TextView**), un champ de saisie (**EditText**), un bouton, un bouton radio, une case à cocher, une liste (**Spinner**), etc.

a. TextView

Le composant **TextView** est utilisé pour dresser une zone de texte dans un gabarit, sa déclaration se fait en Xml dans le layout alors que sa manipulation se fait dynamiquement au niveau du code programme.

Déclaration Xml :

<TextView

..... } Liste des attributs

..... } Liste des attributs

..... } Liste des attributs

/>

Exemple de déclaration :

```
12      <TextView
13          android:layout_width="wrap_content"
14          android:layout_height="wrap_content"
15          android:text="Hello World!"
16          android:id="@+id/text1" />
```

Cette déclaration rassemble les attributs minimaux que peut avoir une vue.

Interprétations :

1. **Ligne 12** : début de la déclaration du composant TextView
2. **Ligne 13** : définit la largeur de la zone texte et on lui a attribué la taille correspondante à son contenu (**wrap_content**).

3. **Ligne 14** : définit la longueur de la zone texte

Les deux propriétés largeur et longueur peuvent avoir l'une des 3 types suivants :

- ✓ **Une taille fixe** : par exemple 30dp (dpi). Donc quel que soit la taille de l'écran du téléphone, l'élément occupera exactement 30dp.
- ✓ **fill_parent** : On demande au composant d'occuper tout l'espace réservé à son conteneur père.
- ✓ **wrap_content** : On demande au composant d'occuper une taille correspondante à son contenu.

4. **Ligne 15** : L'attribut "**text**" permet de spécifier le contenu de la zone
5. **Ligne 16** : On attribue à cette zone de texte un identifiant "text1" et qui sera enregistré dans le fichier **R**. Cet identifiant est utile pour manipuler ce composant dans le code programme.
   La combinaison "**@+id**" désigne que si cet identifiant ne figure pas dans le ficher R, alors on autorise au système d'y ajouter.
   La ligne 16 contient aussi un marqueur de fin de déclaration : "/>".
   Remarque : On peut créer notre propre fichier d'identifiants qui doit être intitulé "**ids.xml**" et qui doit avoir le format suivant :

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <resources>
3       <item  name="afficher" type ="id"></item>
4   </resources>
```

Activité :

Ajouter dans le layout principal de l'application "FormationCenaffe" deux zones de texte "zoneText2" et "zoneText3" et ayant respectivement les largeurs "match_parent" et "20dp".

Solution :
Après l'insertion des deux TextView demandés, le fichier "**activity_main.xml**" aura le contenu suivant :

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="zoneText1"
    android:id="@+id/text1" />

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="zoneText2"
    android:id="@+id/text2"/>

<TextView
    android:layout_width="20dp"
    android:layout_height="wrap_content"
    android:text="zoneText3"
    android:id="@+id/text3" />
```

Activité : Modifier, en mode programmation, le contenu du composant ayant l'identifiant "text2" par la phrase "formation au profit du Cenaffe".
Solution :

La manipulation dynamique d'une vue se fait via son identifiant qui est stocké dans le fichier R ; la phrase **R.id.identifiantComposant** permet de récupérer l'adresse du composant à traiter.

La méthode **findViewById** permet de chercher un composant graphique à travers son adresse toute en lui attribuant l'identifiant de l'élément cherché.

Dans notre cas, la méthode **findViewById(R.id.text2)** permet de récupérer la vue ayant comme identifiant "text2". Le résultat retourné doit être converti (casting) en un composant de même type que celui cherché puis stocker dans une nouvelle variable, ce qui permet d'avoir l'instruction suivante :

TextView var1= **(TextView) findViewById(R.id.text2)**;

Pour modifier le contenu de la variable var1, il suffit d'appeler la méthode **setText** comme s'est indiqué avec l'instruction suivante :

var1.setText ("formation au profit du Cenaffe") ;

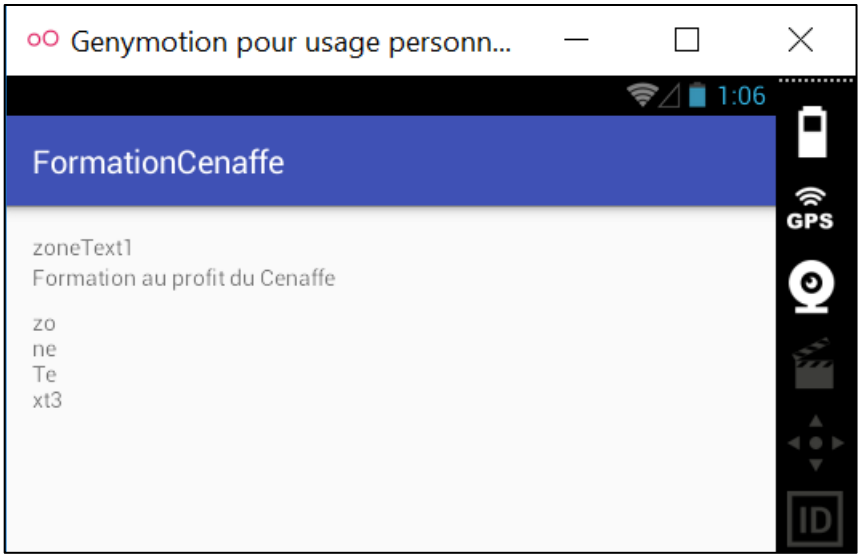
Le fichier "**MainActivity.java**" contient actuellement le code suivant :

```
6  public class MainActivity extends AppCompatActivity {
7
8      @Override
9      protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_main);
12     }
13 }
```

L'insertion des deux nouvelles instructions à la fin du fichier, rend le code comme suit :

```
7  public class MainActivity extends AppCompatActivity {
8
9      @Override
10     protected void onCreate(Bundle savedInstanceState) {
11         super.onCreate(savedInstanceState);
12         setContentView(R.layout.activity_main);
13         TextView var1=(TextView)findViewById(R.id.text2);
14         var1.setText("Formation au profit du Cenaffe");
15     }
16 }
17
```

L'exécution de ce code permet d'avoir l'écran ci-dessous.



Activité : Modifier l'interface utilisateur de l'activité précédente pour qu'elle contienne deux textes placés l'un en haut et l'autre au milieu de l'écran.

Solution :

L'attribut "**android:layout_weight**" permet de spécifier le poids qu'occupe un composant graphique c.a.d la quantité d'espace réservée pour une vue.

Dans notre cas, on va réserver à chaque composant 50% de l'espace d'affichage, donc on va attribuer la valeur 1 au poids de chaque composant.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      android:paddingBottom="@dimen/activity_vertical_margin"
8      android:paddingLeft="@dimen/activity_horizontal_margin"
9      android:paddingRight="@dimen/activity_horizontal_margin"
10     android:paddingTop="@dimen/activity_vertical_margin"
11     android:orientation="vertical"
12     tools:context="com.example.cenaffe.formationcenaffe.MainActivity">
13     <TextView
14         android:layout_width="wrap_content"
15         android:layout_height="wrap_content"
16         android:text="Texte 1"
17         android:id="@+id/text1"
18         android:layout_weight="1"/>
19     <TextView
20         android:layout_width="match_parent"
21         android:layout_height="wrap_content"
22         android:text="Texte 2"
23         android:id="@+id/text2"
24         android:layout_weight="1" />
25 </LinearLayout>
```

Dans la ligne 11, on a défini une orientation verticale du gabarit afin de permettre la disposition verticale des deux vues du layout.

L'exécution de l'application permet d'avoir l'aperçu suivant :

Activité : Varier le poids du texte1 pour qu'il occupe le 2/3 de l'écran.

Activité : Varier les poids des deux textes afin d'avoir le texte2 affiché en bas de l'écran.

Activité : Modifier l'affichage du texte1 pour qu'il soit horizontalement centré.

Solution :

La gravité d'un composant permet de modifier son alignement, les valeurs les plus utilisés de l'attribut "**android:gravity**" sont **left**, **center_horizontal**, **top**, **bottom**, **right**, pour aligner respectivement les vues à gauche, au centre, en haut, en bas et à droite.

Dans notre cas, il suffit d'ajouter **'andoid :gravity="center_horiznontal"'** dans la déclaration du composant texte1.

Remarques :

1. La combinaison CTRL + espace permet d'afficher la liste des attributs associés à une vue.
2. La liste de tous les attributs et les méthodes associés à un "TextView" est détaillée et expliquée sur le site :
   https://developer.android.com/reference/android/widget/TextView.html

b. EditText

C'est une zone de saisie des données qui se déclare comme suit :

```
<EditText
    ……
    …..    } Liste des attributs
    …..
/>
```

L'attribut **android:inputType** appliqué à un "EditText" permet de spécifier le type de donnée à saisir qui peut avoir l'une des valeurs suivantes : text, number, phone, date, time, password, etc.

Activité : Apporter les modifications nécessaires au layout "**activity_main**" afin de permettre à l'application précédente de saisir le nom d'une personne.

Solution :

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.cenaffe.formationcenaffe.MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Nom :"
        android:id="@+id/text1"/>

    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/editText" />
</LinearLayout>
```

Activité : Supprimer le composant "TextView" de l'activité précédente puis ajouter au composant "EditText" l'attribut **android:hint = "Taper le nom"** et en fin ré-exécuter l'application.

Solution :

L'attribut **hint** permet d'afficher un message d'aide à la saisie placé au fond d'un composant **EditText**. Cette aide sera affichée tant que la zone de saisie est vide.

La déclaration du composant "EditText" devient comme suit :

```
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="Taper un nom"
    android:id="@+id/editText" />
```

Activité : Refaire l'activité précédente mais en insérant le message du hint au niveau du code du programme.

Solution :

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    EditText var1=(EditText)findViewById(R.id.editText);
    var1.setHint("Taper un nom");

}
```

c. Bouton

Activité : Ajouter au-dessous du composant "EditText" du layout de l'activité précédente, un bouton ayant le texte "valider".

Solution :

```
18      <Button
19          android:layout_width="wrap_content"
20          android:layout_height="wrap_content"
21          android:text="@string/valider"
22          android:id="@+id/valider"
23          android:layout_gravity="right"
24          />
```

Ci-dessous le code complet du layout :

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <LinearLayout
3       xmlns:android="http://schemas.android.com/apk/res/android"
4       xmlns:tools="http://schemas.android.com/tools"
5       android:layout_width="match_parent"
6       android:layout_height="match_parent"
7       android:paddingBottom="@dimen/activity_vertical_margin"
8       android:paddingLeft="@dimen/activity_horizontal_margin"
9       android:paddingRight="@dimen/activity_horizontal_margin"
10      android:paddingTop="@dimen/activity_vertical_margin"
11      android:orientation="vertical"
12      tools:context="com.example.cenaffe.formationcenaffe.MainActivity">
13      <EditText
14          android:layout_width="fill_parent"
15          android:layout_height="wrap_content"
16          android:hint="@string/hint"
17          android:id="@+id/editText" />
18      <Button
19          android:layout_width="wrap_content"
20          android:layout_height="wrap_content"
21          android:text="@string/valider"
22          android:id="@+id/valider"
23          android:layout_gravity="right"
24          />
25  </LinearLayout>
```

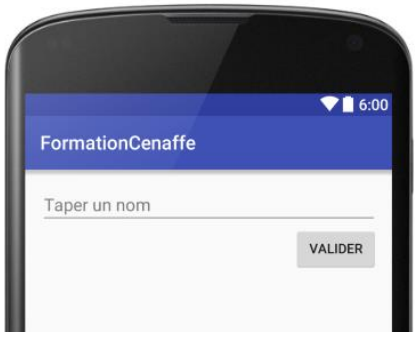
Interprétations :

1. Les lignes 16 et 21 montrent bien qu'on a déclaré dans le fichier "**strings.xml**" du dossier "**res/values**" deux variables intitulées "**hint**" et "**valider**" de type "string".
   Un extrait du fichier "**strings.xml**" :

```
1   <resources>
2       <string name="app_name">FormationCenaffe</string>
3       <string name="hint">Taper un nom</string>
4       <string name="valider">Valider</string>
5   </resources>
```

2. La ligne 23 permet de placer le bouton valider à droite de l'écran.

L'exécution de l'activité permet l'affichage suivant :

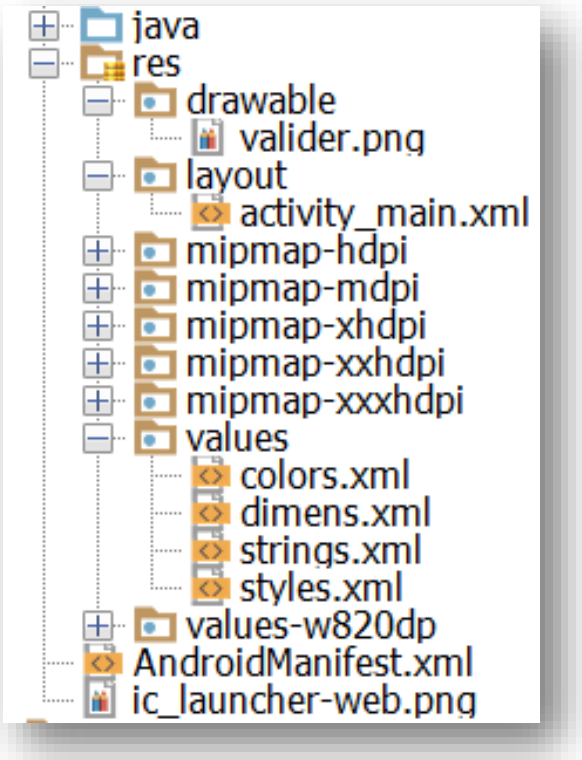


Activité : On veut accompagner l'étiquette du bouton valider par l'icône ✔ afin d'avoir le format suivant : ✔VALIDER

Solution :

On rappelle que le dossier "**res/drawable**" contient les images et les icônes à utiliser dans une application Android.

On commence par placer l'icône "valider.png" dans le dossier "**res/drawable**".



La bibliothèque de développement dispose d'un attribut intitulé **android:drawable** qui permet d'afficher une image sur un bouton et puisque nous voulons avoir un affichage à gauche du label alors on va faire appel à un attribut dérivé intitulé **android:drawableLeft** dont on va lui attribuer une valeur égale au nom de l'image placée dans le dossier drawable.

android:drawableLeft = "@drawable/valider"

On aura alors la déclaration du bouton comme suit :

```
18      <Button
19          android:layout_width="wrap_content"
20          android:layout_height="wrap_content"
21          android:text="@string/valider"
22          android:id="@+id/valider"
23          android:layout_gravity="right"
24          android:drawableLeft="@drawable/valider"/>
```

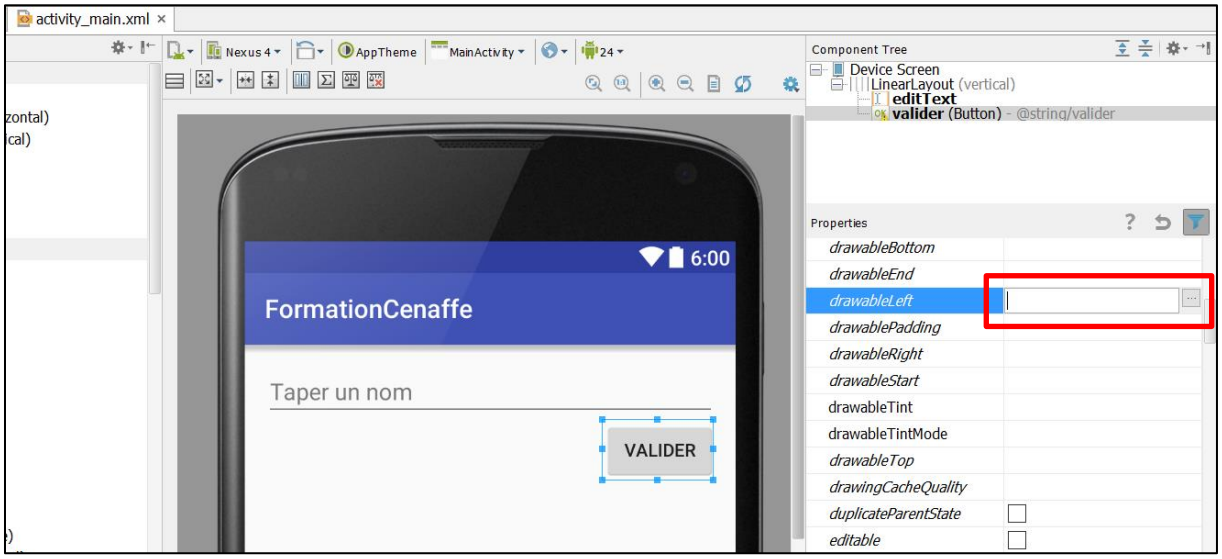
Remarque :

La bibliothèque d'images de l'environnement Android Studio dispose de l'image

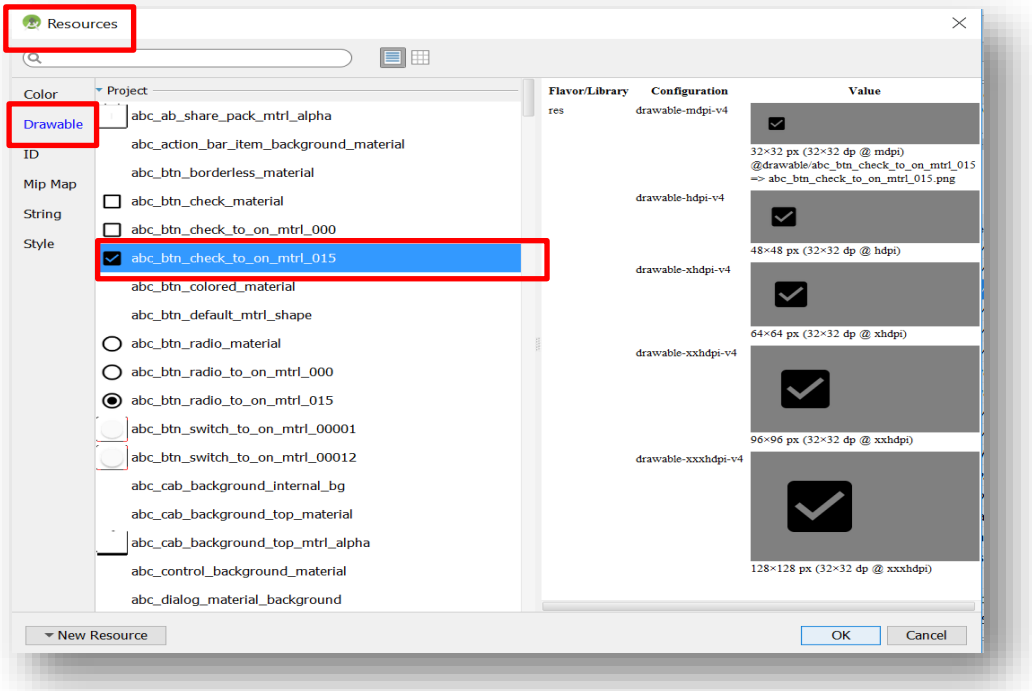
 qu'on pourra exploiter et ceci en procédant comme suit :

En mode "désigne" du layout :

1. on clique au-dessus du bouton "valider" ce qui permet d'afficher la liste de ses propriétés.
2. activer le champ de l'attribut "**drawableLeft**" puis **Shift+Entrée** (ou bien cliquer sur l'icône  de l'attribut )



3. de la fenêtre des ressources qui apparait choisissez l'onglet drawable.



4. sélectionner et valider l'image désirée.

Activité : Modifier l'activité précédente afin qu'elle permette d'afficher un message de salutation contenant le nom saisi suite à un clic sur le bouton "valider".

Solution :

On va développer une méthode intitulée "**affichage**" qui se chargera de l'affichage du message de salutation, ensuite on va l'attribuer à l'attribut **android:onClick** comme suit : **android:onClick = "affichage"**.

Pour récupérer le contenu de la zone de saisie, on procède comme suit :

String nom = ((**EditText**) **findViewById** (R.id.editText)).**getText().toString()**;

String message = **"Bonjour " + nom**;

Pour afficher un message, on peut adopter l'une des manières suivantes :

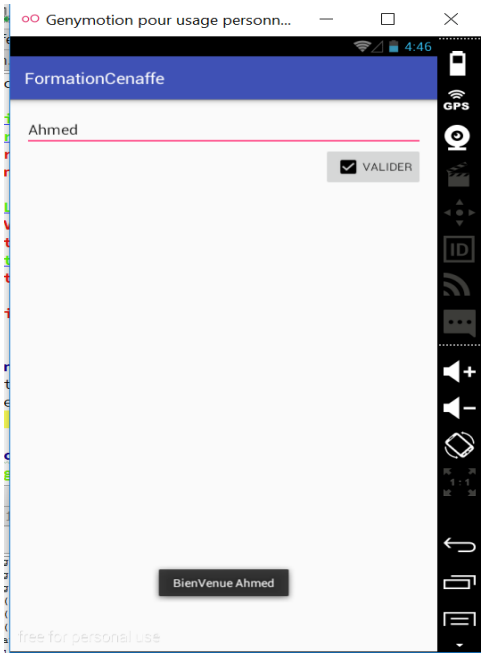
1. **En mode console :**
   - **System.out.println** (message);
   - **Log.d**("label", message);
2. **En mode graphique :**
   - **Toast** : cette méthode permet d'afficher une alerte visuelle pour une durée courte (LENGTH_SHORT) ou longue (LENGTH_LONG) et elle se déclare comme suit :

     Toast t = Toast.makeText (this, message, durée);
     t.show( );
     ou bien :
     Toast.makeText (this, message, durée)**.show( );**

Finalement, on déclare la méthode affichage comme suit :

```
public void affichage(View view) {
    String message = "Bonjour " + ((EditText) findViewById(R.id.editText)).getText().toString();
    Toast.makeText(MainActivity.this, message, Toast.LENGTH_LONG).show();
}
```

L'exécution de l'application a permis d'avoir l'écran suivant :



d. Bouton radio

Activité : On va ajouter à notre application deux boutons radio permettant à l'utilisateur de spécifier son genre.

Solution :

La structure de déclaration d'un bouton radio est la suivante :

<RadioButton
.....
.....
.....
/>

Liste des attributs

Les déclarations des boutons radio d'un même sujet doivent être regroupées dans un groupe de boutons dont la déclaration est la suivante :

<RadioGroup
.....
.....
.....
/>

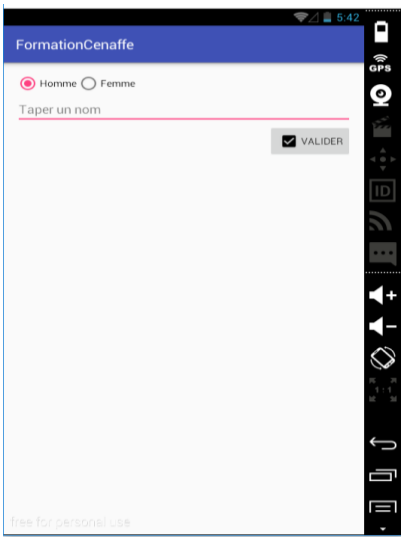
Liste des attributs

On a réorganisé le layout de notre application pour que l'écran d'exécution contienne dans sa première ligne les boutons radio et on a déclaré un groupe de boutons juste avant la déclaration de la zone de saisie.

```
14      <RadioGroup
15          android:layout_width="wrap_content"
16          android:layout_height="wrap_content"
17          android:orientation="horizontal"
18          android:id="@+id/groupeSexe">
19
20          <RadioButton
21              android:layout_width="wrap_content"
22              android:layout_height="wrap_content"
23              android:text="@string/homme"
24              android:id="@+id/b1"
25              android:checked="true" />
26
27          <RadioButton
28              android:layout_width="wrap_content"
29              android:layout_height="wrap_content"
30              android:text="@string/femme"
31              android:id="@+id/b2"
32              android:checked="false" />
33
34      </RadioGroup>
```

1. Les lignes 23 et 30 montrent que les libellés des boutons sont des constantes déclarées dans le fichier "**strings.xml**".
2. Les lignes 25 et 32 contiennent l'attribut "**android:checked**" pour spécifier quel bouton sera sélectionné par défaut.

L'exécution de l'application a permis d'avoir l'écran suivant :



La manipulation des boutons doit se faire dans le code programme de l'activité concernée.

Activité : Apporter les modifications nécessaires à l'application précédente afin d'ajouter l'abréviation correspondante au genre sélectionné au message affiché.

Exemples : Bonjour **Mr**. xxx

Bonjour **Mme** xxx

Solution :

En mode programmation, la méthode **isChecked** retourne une valeur booléenne indiquant l'état du bouton radio.

Le code de la méthode affichage aura le contenu suivant :

```
public void affichage(View view) {
    String sexe = "Mr.";
    RadioButton varBouton = (RadioButton) findViewById(R.id.b2);
    if (varBouton.isChecked()) {
        sexe = "Mme";
    }
    String message = "Bonjour " + sexe + " " + ((EditText) findViewById(R.id.editText)).getText().toString();
    Toast.makeText(MainActivity.this, message, Toast.LENGTH_LONG).show();
}
```

Dans les lignes 25, 28 et 30 on a déclaré des variables chaînes contenant les constantes "Mr.", "Mme." et "Bonjour", et puisque il est recommadé de séparer les constantes du code programme alors on va les déclarer dans le fichier "**strings.xml**" puis on fait des appels à ces ressources.

A la ligne 26, on a déclaré une variable intitulée "varBouton" de type "radioButton" et on lui a affecté l'adresse du composant d'identifiant b2 (R.id.b2).

Le fichier "**strings.xml**" aura le code suivant :

```
1  <resources>
2      <string name="app_name">FormationCenaffe</string>
3      <string name="hint">Taper un nom</string>
4      <string name="valider"> Valider</string>
5      <string name="homme">Homme</string>
6      <string name="femme">Femme</string>
7      <string name="mr">Mr.</string>
8      <string name="mme">Mme</string>
9  </resources>
```

La manipulation d'une ressource de ce fichier en mode programmation se fait via l'instruction **getRessources** comme suit :

getRessources( ).getString(R.string.nomDeLaConstanteChaine) ;

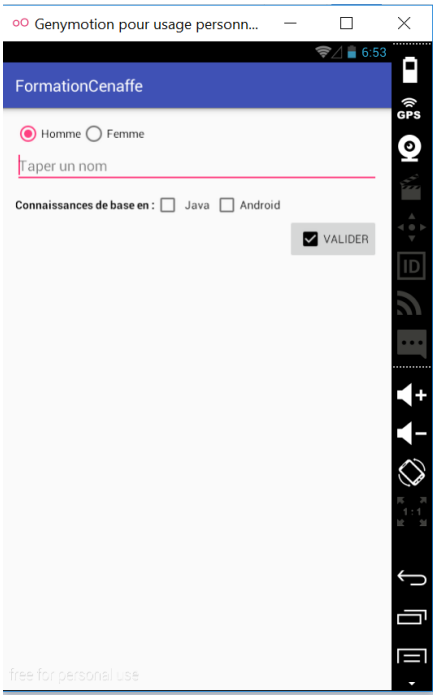
Le code de la méthode affichage devient alors comme suit :

```
public void affichage(View view) {
    String sexe = getResources().getString(R.string.mr);
    RadioButton varBouton = (RadioButton) findViewById(R.id.b2);
    if (varBouton.isChecked()) {
        sexe = getResources().getString(R.string.mme);
    }
    String message = "Bonjour " + sexe + " " + ((EditText) findViewById(R.id.editText)).getText().toString();
    Toast.makeText(MainActivity.this, message, Toast.LENGTH_LONG).show();
}
```

e. Case à cocher

Activité : Modifier l'application précédente afin de permettre à l'utilisateur de répondre s'il a des connaissances en langages de programmation (Java – Android). De plus, le clic sur le bouton "valider" permet d'afficher les connaissances de cet utilisateur en langages de programmation.

Nous désirons avoir une interface utilisateur similaire à la suivante :



Solution :

On va commencer par la déclaration, dans le fichier "strings.xml", de 4 constantes chaînes comportant respectivement "Java", "Android", "Avez-vous des connaissances en :" et "Vous avez des connaissances en".

```xml
<resources>
    <string name="app_name">FormationCenaffe</string>
    <string name="hint">Taper un nom</string>
    <string name="valider"> Valider</string>
    <string name="homme">Homme</string>
    <string name="femme">Femme</string>
    <string name="mr">Mr.</string>
    <string name="mme">Mme</string>
    <string name="java">Java</string>
    <string name="android">Android</string>
    <string name="connaissances">Avez-vous des connaissances en :</string>
    <string name="messageConnaissance">Vous avez des connaissances en </string>
</resources>
```

La déclaration d'une case à cocher se fait comme suit :

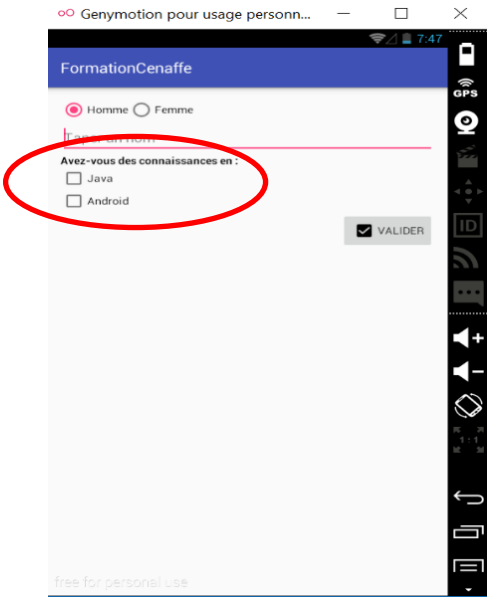
<CheckBox
.....
.....
..... } Liste des attributs
/>

On va déclarer dans le layout de l'application un TexView et 2 cases à cocher de la manière suivante :

```xml
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/connaissances"
    android:id="@+id/connaissances"
    android:textStyle="bold"
    android:textColor="@color/bright_foreground_material_light" />
<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/java"
    android:id="@+id/caseJava" />

<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/android"
    android:id="@+id/caseAndroid" />
```

Ce qui permet d'avoir l'écran d'exécution suivant :



Puisque le gabarit du layout est de type **LinearLayout** avec une orientation **verticale**, alors les nouveaux composants insérés seront disposés verticalement. Pour les arranger horizontalement, on est appelé de les insérer dans un nouveau gabarit de type **LinearLayout** avec une orientation **horizontale**, ce qui rend la partie de déclaration, à insérer dans le fichier "activity_main", comme suit :

```
43  <LinearLayout
44      android:orientation="horizontal"
45      android:layout_width="match_parent"
46      android:layout_height="wrap_content"
47      android:paddingTop="10dp">
48      <TextView
49          android:layout_width="wrap_content"
50          android:layout_height="wrap_content"
51          android:text="@string/connaissances"
52          android:id="@+id/connaissances"
53          android:textStyle="bold"
54          android:textColor="@color/bright_foreground_material_light" />
55      <CheckBox
56          android:layout_width="wrap_content"
57          android:layout_height="wrap_content"
58          android:text="@string/java"
59          android:id="@+id/caseJava" />
60      <CheckBox
61          android:layout_width="wrap_content"
62          android:layout_height="wrap_content"
63          android:text="@string/android"
64          android:id="@+id/caseAndroid" />
65
66  </LinearLayout>
```

Manipulation d'une case à cocher :

La méthode **isChecked** appliquée à une vue de type "case à cocher" permet de tester si ce composant est coché ou non.

Dans notre cas, on va insérer le code suivant dans la méthode affichage :

```
31          CheckBox choixJava = (CheckBox)findViewById(R.id.caseJava);
32          CheckBox choixAndroid = (CheckBox)findViewById(R.id.caseAndroid);
33          String connsissances="";
34          if(choixJava.isChecked()){
35              connsissances=getResources().getString(R.string.messageConnaissance)+" "+getResources().getString(R.string.java);
36              if(choixAndroid.isChecked()){
37                  connsissances +=" et en " +getResources().getString(R.string.android);
38              }
39          }
40          else if(choixAndroid.isChecked()){
41              connsissances=getResources().getString(R.string.messageConnaissance)+" "+getResources().getString(R.string.android);
42          }
```

Interprétations :

1. La ligne 31 : Déclaration d'une variable "case à cocher" intitulée "choixJava" qui se pointe vers la vue d'identifiant "caseJava".
2. La ligne 32 : Déclaration d'une variable "case à cocher" intitulée "choixAndroid" qui se pointe vers la vue d'identifiant "caseAndroid".
3. Ligne 34 : Teste si la case correspondante au langage Java est cochée.
4. Ligne 35 : On affecte à la variable "**connaissances**" la concaténation de la valeur de la constante intitulée "**messageConnaissance**" avec celle intitulée "**java**".
5. Ligne 36 : Teste si la case correspondante au langage Android est cochée.
6. Ligne 37 : On concatène à la variable "**connaissances**" la valeur de la constante intitulée "**android**"
7. Ligne 41 : On affecte à la variable "**connaissances**" la concaténation de la valeur de la constante intitulée "**messageConnaissance**" avec celle intitulée "**android**".

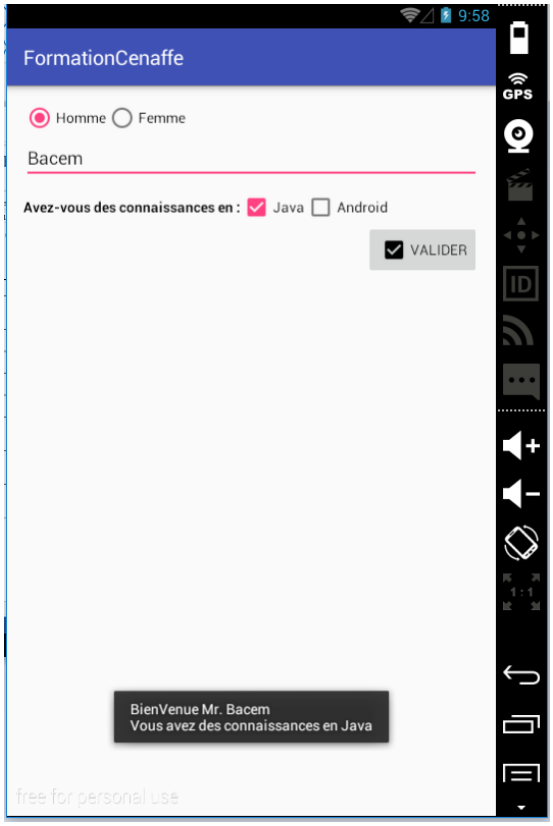
Le code complet de la méthode affichage sera comme suit :

```
25      public void affichage(View view) {
26          String sexe = getResources().getString(R.string.mr);
27          RadioButton varBouton = (RadioButton) findViewById(R.id.b2);
28          if (varBouton.isChecked()) {
29              sexe = getResources().getString(R.string.mme);
30          }
31          CheckBox choixJava = (CheckBox)findViewById(R.id.caseJava);
32          CheckBox choixAndroid = (CheckBox)findViewById(R.id.caseAndroid);
33          String connsissances="";
34          if(choixJava.isChecked()){
35              connsissances=getResources().getString(R.string.messageConnaissance)+" "+getResources().getString(R.string.java);
36              if(choixAndroid.isChecked()){
37                  connsissances +=" et en " +getResources().getString(R.string.android);
38              }
39          }
40          else if(choixAndroid.isChecked()){
41              connsissances=getResources().getString(R.string.messageConnaissance)+" "+getResources().getString(R.string.android);
42          }
43          String message = "BienVenue " + sexe +" " +((EditText) findViewById(R.id.editText)).getText().toString()
44                          +"\n"+connsissances;
45          Toast.makeText(MainActivity.this, message, Toast.LENGTH_LONG).show();
46      }
```

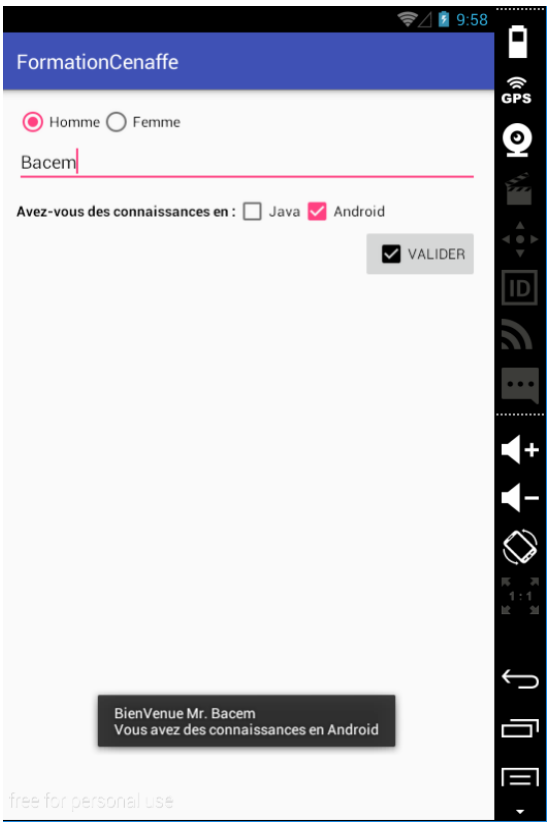
Dans la ligne 44, on a modifié l'affichage en ajoutant la valeur de la variable "**connaissances**". "**\n**" permet un affichage avec un retour à la ligne.

Exécutions :

Cas 1 : Java uniquement



Cas 2 : Android uniquement



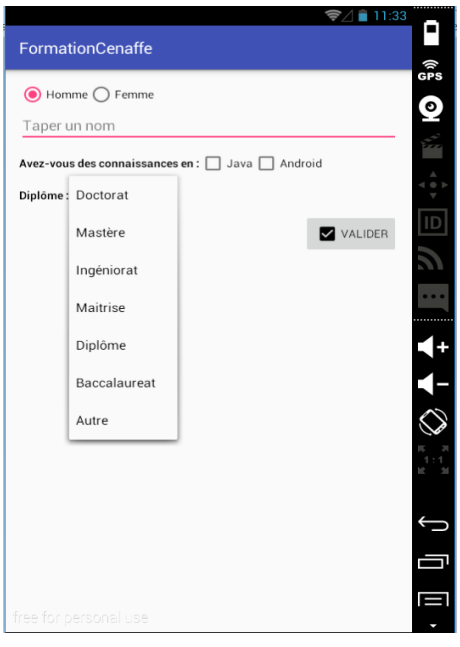
Cas 3 : Java et Android

Cas 4 : Ni java ni Android

f. Spinner

Activité : Ajouter à l'activité précédente, une liste des diplômes obtenus dont l'utilisateur est appelé à sélectionner la valeur qui lui convient.

Le résultat attendu de l'activité :



Solution :

La déclaration d'une liste déroulante, intitulée "Spinner", se fait comme suit :

<Spinner

.....
.....
.....
/>

Liste des attributs

On va déclarer dans le layout de l'activité précédente un gabarit avec une orientation **horizontale** dans lequel on va précéder la liste par l'étiquette "Diplôme : ".

Le code à insérer aura la structure suivante :

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/niveauLabel"
        android:textColor="@color/abc_input_method_navigation_guard"
        android:textStyle="bold" />

    <Spinner
        android:id="@+id/spinnerDiplome"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

En plus, on va déclarer un tableau de chaînes (**string-array**) contenant la liste des diplômes à afficher dans le layout et ce on insérant le bloc de déclaration ci-dessous dans le fichier "**strings.xml**" :

```
    <string-array name="listeDiplomes">
        <item >Doctorat</item>
        <item>Mastère</item>
        <item>Ingéniorat</item>
        <item>Maîtrise</item>
        <item>Diplôme</item>
        <item>Baccalauréat</item>
        <item>Autre</item>
    </string-array>
```

Manipulation d'une liste :

Pour manipuler un spinner, on procède comme suit :

a. Déclarer une variable qui pointe vers le spinner du layout.

Spinner spinnerVar = (**Spinner**) **findViewById** (**R.id.*spinnerDiplome***);

spinnerVar est une variable de type "Spinner" qui pointe vers le spinner déclaré dans le layout et ayant l'identifiant "spinnerDiplome".

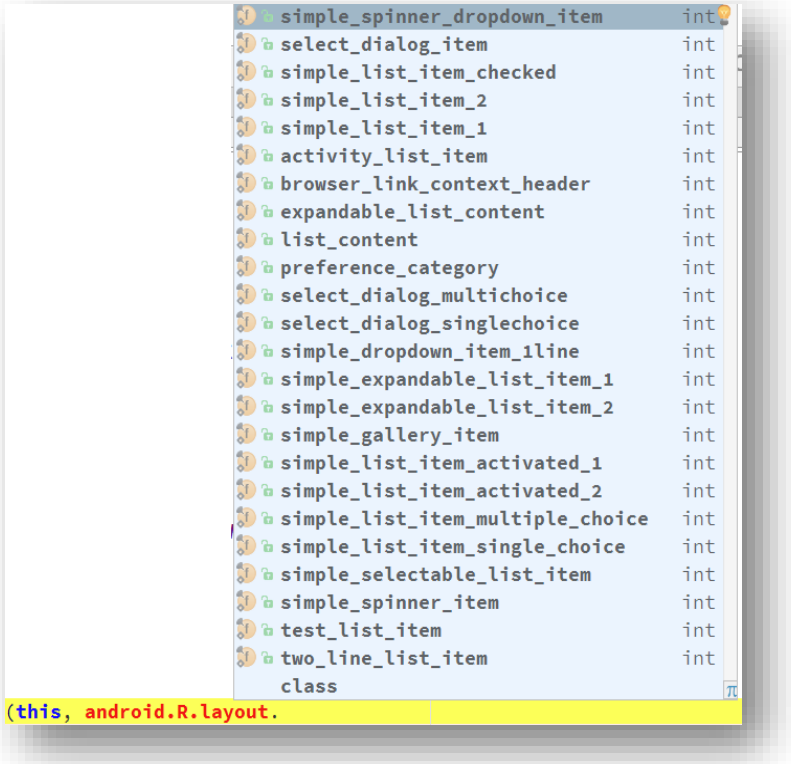
b. Déclarer une variable de type tableau de chaînes pour récupérer les valeurs de la liste des diplômes déclarée dans le fichier "strings.xml"

String[ ] listeVar = **getResources**().**getStringArray**(**R.array.*listeDiplomes***);

c. Créer un adaptateur de type **ArrayAdapter** qui permet d'adapter la liste des diplômes (listeVar) en un spinner selon un modèle de présentation (dans notre cas, on a opté pour "**simple_spinner_dropdown_item**")

ArrayAdapter <**String**> **adapterVar** = **new ArrayAdapter** <**String**> (**this**, **android.R.layout.*simple_spinner_dropdown_item***, **listeVar**);

Plusieurs modèles d'adaptation sont fournies par le système de développement et pour les visualiser, il suffit de taper "**android.R.layout.**"



d. Appliquer la méthode **setAdapter** à la variable adaptée.

spinnerVar.setAdapter(adapterVar);

On va regrouper, dans une nouvelle méthode, toutes ces actions et qui aura le contenu suivant :

```
47  private void remplir(){
48      spinnerVar = (Spinner)findViewById(R.id.spinnerDiplome);
49      String[] listeVar= getResources().getStringArray(R.array.listeDiplomes);
50      ArrayAdapter<String> adapterVar = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_dropdown_item,listeVar);
51      spinnerVar.setAdapter(adapterVar);
52  }
```

La méthode **getSelectedItem** appliquée à une variable de type "Spinner" permet de récupérer l'élément sélectionné et pour l'afficher dans le message de l'application, on va apporter des légères modifications à la méthode **affichage** afin d'avoir le code suivant :

```
24  public void affichage(View view) {
25      String sexe = getResources().getString(R.string.mr);
26      RadioButton varBouton = (RadioButton) findViewById(R.id.b2);
27      if (varBouton.isChecked()) {
28          sexe = getResources().getString(R.string.mme);
29      }
30      CheckBox choixJava = (CheckBox)findViewById(R.id.caseJava);
31      CheckBox choixAndroid = (CheckBox)findViewById(R.id.caseAndroid);
32      String connsissances="";
33      if(choixJava.isChecked()){
34          connsissances=getResources().getString(R.string.messageConnaissance)+" "+getResources().getString(R.string.java);
35          if(choixAndroid.isChecked()){
36              connsissances +=" et en " +getResources().getString(R.string.android);
37          }
38      }
39      else if(choixAndroid.isChecked()){
40          connsissances=getResources().getString(R.string.messageConnaissance)+" "+getResources().getString(R.string.android);
41      }
42
43      String message = "BienVenue " + sexe +" " +((EditText) findViewById(R.id.editText)).getText().toString()
44                      +"\n"+connsissances+"\n"+spinnerVar.getSelectedItem();
45      Toast.makeText(MainActivity.this, message, Toast.LENGTH_LONG).show();
46  }
```

La variable "spinnerVar" (ligne 44) n'est lisible en dehors de la méthode **remplir** que lorsqu'elle est déclarée **public**, en plus le module de remplissage doit être appelé au niveau du programme principal comme suit :

```
14  public class MainActivity extends AppCompatActivity {
15      Spinner spinnerVar;
16      @Override
17      protected void onCreate(Bundle savedInstanceState) {
18          super.onCreate(savedInstanceState);
19          setContentView(R.layout.activity_main);
20          remplir();
21
22      }
```

g. ImageView

Activité : Ajouter l'image "**cenaffe.png**" à l'interface utilisateur de l'application.

Solution :

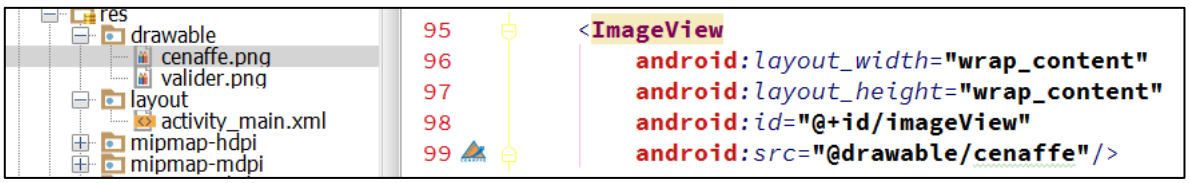
La balise <**ImageView**

.....
.....
..... } Liste des attributs

/>

Cette balise permet de déclarer une image dans un layout.

L'attribut "**android:src**" permet de choisir la source de l'image qui doit être placée dans le dossier "**res/drawable**" de l'application.



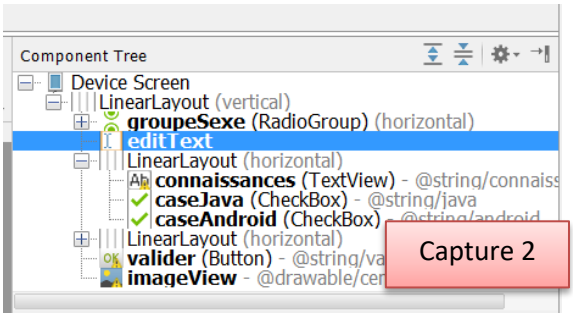
La ligne 99 permet de spécifier la source et l'aperçu de l'image à afficher.

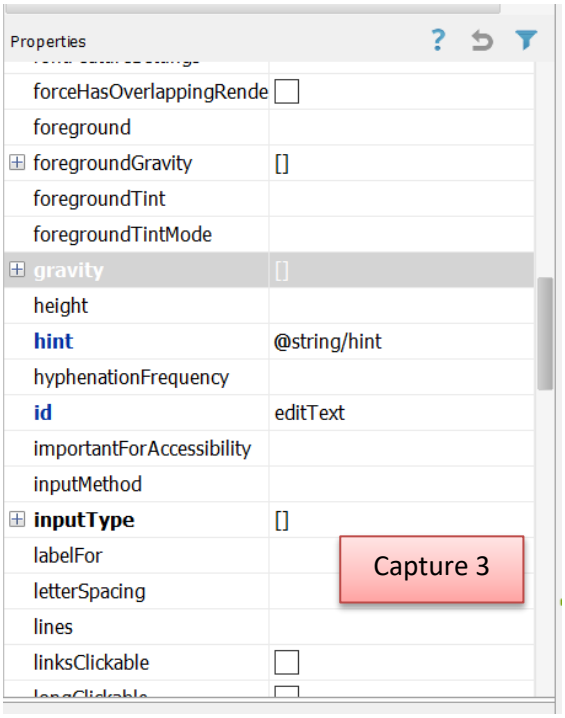
5- Propriétés d'une vue

Chaque vue admet des propriétés gérables en trois modes.

a. Mode graphique

À travers la palette "**properties**" d'Android Studio (capture 3), on peut gérer graphiquement les propriétés de chaque vue. Cette palette apparait lorsqu'on active le composant de l'écran de désigne (capture 1) ou lorsqu'on clique au-dessus à partir de l'arborescence des vues (capture 2).





b. Mode texte

Android Studio propose aux développeurs une assistance contextuelle pour modifier les propriétés d'une vue. En effet, il suffit de taper **CTRL** + **Espace** pour visualiser toutes les propositions.

c. Mode programme

Une vue peut être créée ou modifiée à travers des instructions Java.

Exemple :

La propriété **setTextColor** permet de spécifier la couleur du texte d'une vue.

Activité : Ajouter, dynamiquement, au formulaire précédent un bouton ayant l'étiquète "Afficher" et coloré en bleu.

Solution :
On commence par attribuer un identifiant au gabarit du layout de l'activité précédente afin qu'on puisse le récupérer pour y insérer le nouveau composant.

android*:id*="**@+id/gabaritPrincipal**"

Ensuite, on crée une instance de ce gabarit :

LinearLayout var1 = (**LinearLayout**)findViewById(**R.id.*gabaritPrincipal***);

Puis, on crée la nouvelle vue de type Button :

```
Button nouveauBouton = new Button ( this );
nouveauBouton.setText ("Afficher");
nouveauBouton.setTextColor (Color.BLUE);
```

Enfin, on ajoute le nouveau composant au gabarit :

```
var1.addView (nouveauBouton);
```

6- Les évènements

Sous Android, toutes les actions de l'utilisateur sont perçues comme un événement, que ce soit le clic sur un bouton, le maintien du clic, le déplacement d'un élément, etc. Ces événements peuvent être déclenchés par les éléments de l'interface pour exécuter des actions.

Le mécanisme de déclenchement repose sur la notion d'écouteurs, appelés listeners dans la documentation Java. Il permet d'associer un événement à une méthode à appeler en cas d'apparition de cet événement. Si un écouteur est défini pour un élément graphique, la plate-forme Android appellera la méthode associée dès que l'événement sera produit sur cet élément.

Pour un événement **OnClick** (élément cliqué), la méthode associée sera **OnClick**() : il suffit alors de redéfinir cette méthode pour qu'elle soit appelée lorsque l'utilisateur cliquera sur l'élément graphique associé. Ainsi, pour que l'interface réagisse à un événement sur un élément, il faudra choisir l'élément et l'événement à intercepter, définir un écouteur sur cet événement et redéfinir la méthode associée.

Dans notre application, on a déjà associé, en mode xml, la méthode affichage à l'attribut **onClick** du bouton valider.

Activité :

Développer en mode programmation, un évènement d'écoute sur le bouton nouvellement créée dans l'activité précédente qui permet d'afficher la valeur du nom saisi.

Solution :

On va redéfinir la méthode **onClick** appliquée sur le bouton intitulée "Afficher" et ceci en utilisant l'une des deux méthodes suivantes :

1ère méthode : On redéfini la méthode **onClick** au sein d'une nouvelle instanciation de l'interface **onClickListener** comme suit :

```
33  nouveauBouton.setOnClickListener(new View.OnClickListener() {
34      @Override
35      public void onClick(View view) {
36          String message = ((EditText) findViewById(R.id.editText)).getText().toString();
37          Toast.makeText(MainActivity.this, message, Toast.LENGTH_LONG).show();
38      }
39  });
```

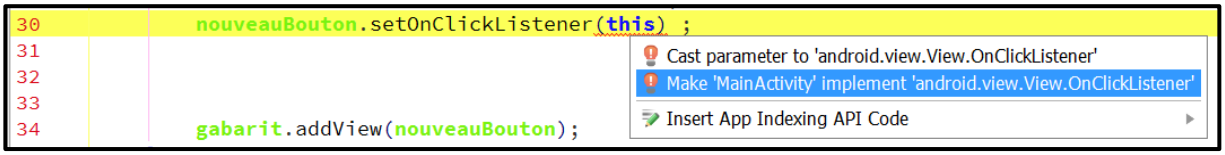
Interprétations :

Ligne 33 : On a appliqué l'évènement **setOnClickListener** sur le bouton intitulé "nouveauBouton" et on lui a passé une nouvelle instanciation de **onClickListener** dans laquelle on va redéfinir le comportement de la méthode **onClick**.
Ligne 34 : Désigne une action de redéfinition
Ligne 35 : C'est la méthode **onClick** qu'on va redéfinir
Ligne 36 : On cherche le contenu de la zone de saisie d'identifiant "editText"
Ligne 37 : Affichage du message.

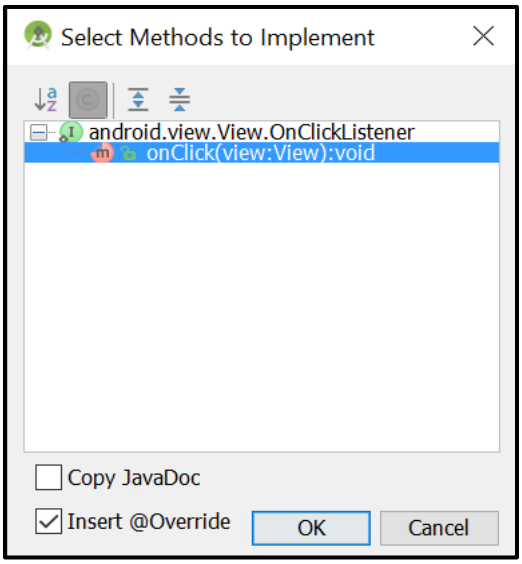
2ème méthode : On redéfini la méthode **onClick** au sein d'une nouvelle instanciation de l'interface **onClickListener** comme suit :

```
30          nouveauBouton.setOnClickListener(this) ;
```

Une erreur se produit concernant le mot **this** et pour visualiser les propositions de correction, on l'active ensuite on tape la combinaison **Alt + Entrée** afin d'afficher l'écran suivant :



La 2ème alternative de correction propose l'implémentation des méthodes abstract de la classe **View.OnClickListener**. Le choix de cette alternative permet l'apparition d'un écran demandant la sélection des méthodes à développer comme suit :



L'acceptation d'implémenter le code de la méthode **onClick** entraine les 2 actions suivantes :

- La modification de l'entête de la classe MainActivity en lui ajoutant l'interface dont on va développer l'une de ses méthodes.

```
17  public class MainActivity extends AppCompatActivity implements View.OnClickListener {
```

- L'insertion du squelette du module **onClick** à développer :

```
67      @Override
68      public void onClick(View view) {
69
70      }
```

Puis on insère dans la ligne 69 les mêmes instructions de la 1ère méthode afin d'avoir le code suivant :

```
67      @Override
68      public void onClick(View view) {
69          String message =((EditText) findViewById(R.id.editText)).getText().toString();
70          Toast.makeText(MainActivity.this, message, Toast.LENGTH_LONG).show();
71      }
```

Activité : Modifier l'activité précédente afin d'afficher le libellé du diplôme sélectionné dès qu'on choisit un élément de la liste proposée.

Solution :

On va mettre en écoute la liste des diplômes obtenus afin de déclencher un affichage dès la sélection d'un élément d'où on va redéfinir la méthode **onItemSelected** de la classe **OnItemSelectedListener**. De même, deux façons sont possibles pour développer cette écoute :

1ère méthode : On redéfini la méthode **onItemSelected** au sein d'une nouvelle instanciation de l'interface **OnItemSelectedListener** comme suit :

```
34      final Spinner spinnerDiplome = (Spinner)findViewById(R.id.spinnerDiplome);
35      spinnerDiplome.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
36          @Override
37          public void onItemSelected(AdapterView<?> adapterView, View v, int p, long l) {
38              System.out.println(spinnerDiplome.getItemAtPosition(p));
39          }
40
41          @Override
42          public void onNothingSelected(AdapterView<?> adapterView) {
43
44          }
45      });
```

Interprétations :

Ligne 34 : On crée une variable (spinnerDiplome) référant à la vue spinnerDiplome du layout.

Ligne 35 : On associe à la variable "spinnerDiplome" un évènement d'écoute sur l'élément sélectionné de la liste.

Ligne 36 : Déclaration d'une action de réécriture

Ligne 37 : Définition de la méthode **onItemSelected** qu'on lui a passé comme paramètre une vue "**v**" à la position "**p**".

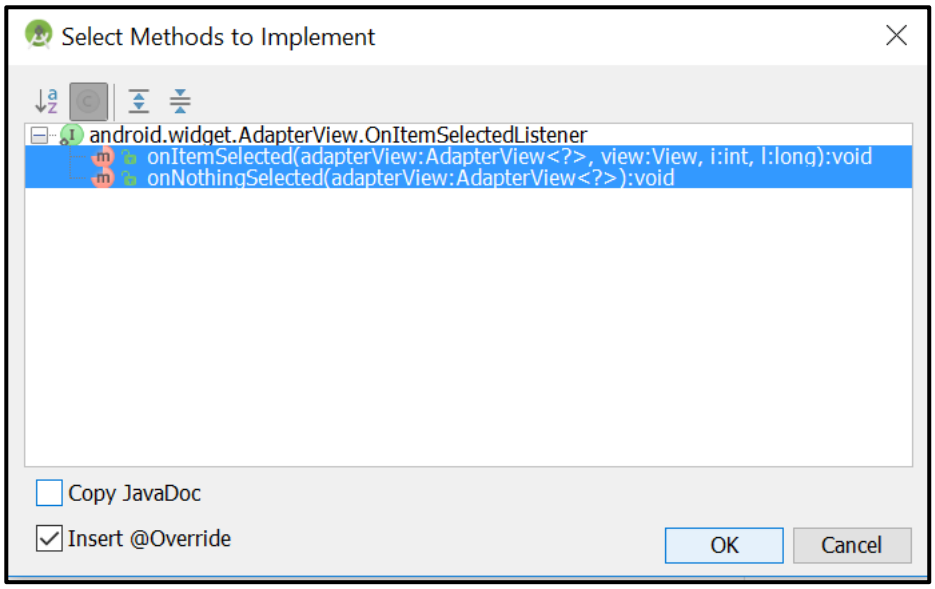
Ligne 38 : On va afficher, en mode console, la valeur de l'élément de la liste **spinnerDiplome** qui se trouve à la position "**p**".

2ème méthode : On redéfini la méthode **onItemSelected** au sein d'une nouvelle instanciation de l'interface **OnItemSelectedListener** comme suit :

```
34      final Spinner spinnerDiplome = (Spinner)findViewById(R.id.spinnerDiplome);
35      spinnerDiplome.setOnItemSelectedListener(this);
```

Dans la ligne 35, le système détecte une erreur d'implémentation et il nous propose les 2 alternatives suivantes :

Le choix de la 2ème alternative engendre l'apparition de l'écran ci-dessous invitant le développeur à choisir les méthodes à implémenter.



La confirmation de cet écran entraîne :

- l'ajout du nom de l'interface **AdapterView.OnItemSelectedListener** au nom de la classe de l'application comme suit :

```
18  public class MainActivity extends AppCompatActivity
19          implements View.OnClickListener, AdapterView.OnItemSelectedListener {
```

- l'intégration des 2 méthodes à développer suivantes :
  - La méthode **onItemSelected** à laquelle on va insérer la ligne d'affichage
  - La méthode **onNothingSelected**

  Permet d'avoir finalement le code suivant :

```
79      @Override
80      public void onItemSelected(AdapterView<?> adapterView, View v, int p, long l) {
81          System.out.println(spinnerDiplome.getItemAtPosition(p));
82      }
83
84      @Override
85      public void onNothingSelected(AdapterView<?> adapterView) {
86
87      }
```

Remarque :
La variable **spinnerDiplome** doit être une variable **public**.

VI- Multi-Activities et Intents

Une activité est généralement conçue pour résoudre une seule tâche, comme la saisie des données, l'affichage d'informations, etc. Dans certains cas, on désire développer une application traitant plusieurs actions. Par exemple l'ajout

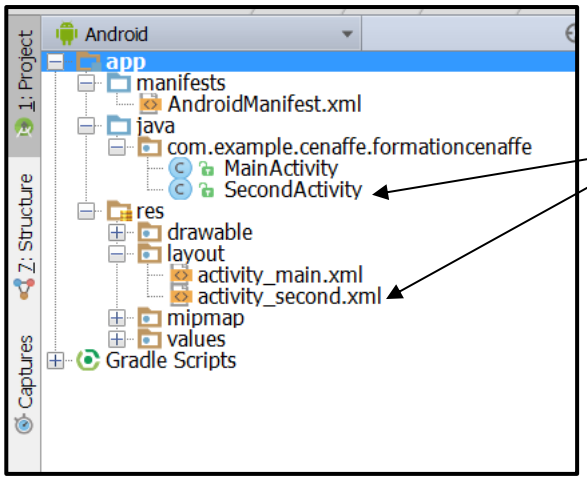
des données et l'affichage d'informations. Si tel est le cas, alors on aura besoin d'implémenter plusieurs activités.

Activité :

Ajouter une nouvelle activité à l'application précédente, puis assurer la communication entre les deux layouts à travers un clic sur un nouveau bouton ajouté au layout principal.

Solution :

- On crée une nouvelle activité :
  File → New →Activity → Empty activity
  Ensuite, attribuer un nom à la nouvelle activité. On a choisi **SecondActivity** comme nom à la nouvelle activité.



Création de la classe SecondActivity.java et le layout activity_second.xml

- Android Studio ajoute automatiquement au fichier "**AndroidManifest.Xml**" une ligne de déclaration de la nouvelle activité comme suit :
  <**activity android**:*name*="**.SecondActivity**"></**activity**>

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3       package="com.example.cenaffe.formationcenaffe">
4
5       <application
6           android:allowBackup="true"
7           android:icon="@mipmap/ic_launcher"
8           android:label="FormationCenaffe"
9           android:supportsRtl="true"
10          android:theme="@style/AppTheme">
11          <activity android:name=".MainActivity">
12              <intent-filter>
13                  <action android:name="android.intent.action.MAIN" />
14
15                  <category android:name="android.intent.category.LAUNCHER" />
16              </intent-filter>
17          </activity>
18          <activity android:name=".SecondActivity"></activity>
19      </application>
20
21  </manifest>
```

- On modifie l'activité principale :
  - Ajouter un nouveau bouton (envoyer)

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/envoyer"
    android:id="@+id/bouton3" />
```

  - Mettre en écoute ce nouveau bouton :

```
Button boutonEnvoyer = (Button)findViewById(R.id.bouton3);
boutonEnvoyer.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

    }
});
```

  - Ajouter les 2 instructions suivantes à la méthode "**onClick**" :
    Intent varIntent= new Intent(MainActivity.this, SecondActivity.class);

Les Intents permettent de gérer l'envoi et la réception de données entre les activités d'une application.
On a déclaré une nouvelle variable de type **Intent** pour assurer le transfert de données entre la classe courante "MainActivity" et celle de "SecondActivity".

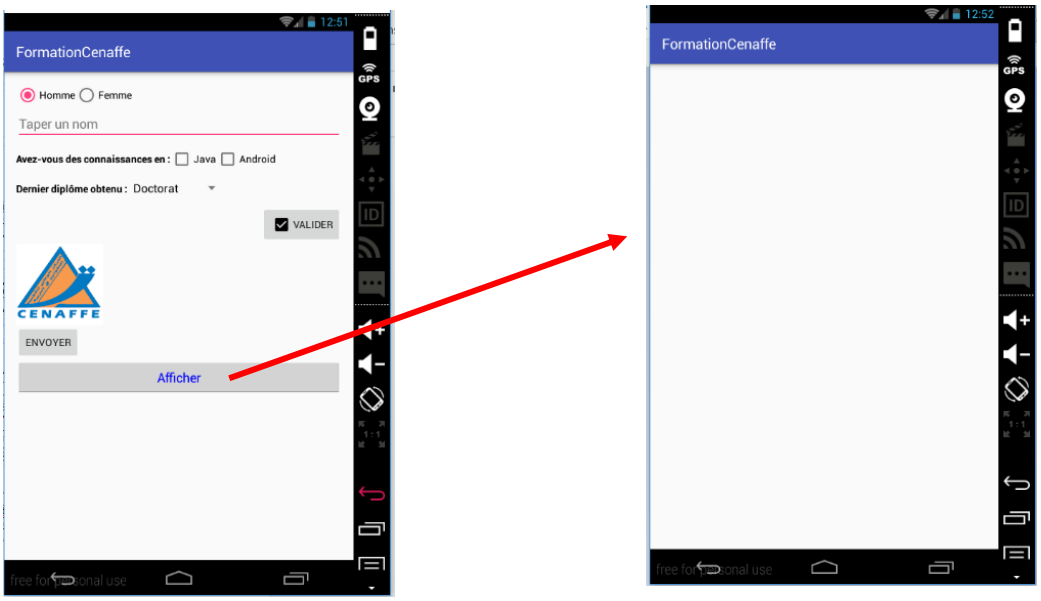
startActivity(varIntent);

Cette instruction permet de lancer la nouvelle activité.

Ce qui permet d'avoir le code final de la méthode **onClick** comme suit :

```
39      Button boutonEnvoyer = (Button) findViewById(R.id.bouton3);
40      boutonEnvoyer.setOnClickListener(new View.OnClickListener() {
41          @Override
42          public void onClick(View view) {
43              Intent intent = new Intent(MainActivity.this, SecondActivity.class);
44              startActivity(intent);
45          }
46      });
```

L'exécution de l'application permet de passer de l'activité principale vers la seconde suite au clic sur le bouton "Envoyer".



De la barre de titre de la seconde activité, on constate que :

- Le titre de la seconde activité et celui de l'application :
  Pour avoir un label approprié à une activité, on peut passer par l'attribut "label" et déclaré le fichier **AndroidManifest.xml** comme suit :

```
18      <activity android:name=".SecondActivity"
19          android:label="Seconde activité">
20      </activity>
```

  Ce qui permet d'avoir le titre comme suit :

- La seconde application ne permet pas le retour en arrière :
  Pour faire paraître la flèche de retour en arrière dans la barre des titres on doit déclarer le nom de l'activité père à celle en question et ceci dans le fichier **AndroidManifest.xml** comme suit :

```
<activity android:name=".SecondActivity"
    android:label="Seconde activité"
    android:parentActivityName=".MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

Activité :
Modifier l'application précédente afin de permettre à la seconde activité d'afficher le nom saisi dans l'activité principal.
Solution :
La méthode **putExtra** appliquée à une variable de type **Intent** permet de passer des paramètres à une autre activité.
1. On va modifier la méthode **onClick** du bouton envoyer comme suit :

```
40          Button boutonEnvoyer = (Button) findViewById(R.id.bouton3);
41          boutonEnvoyer.setOnClickListener(new View.OnClickListener() {
42              @Override
43              public void onClick(View view) {
44                  Intent intent = new Intent(MainActivity.this, SecondActivity.class);
45                  intent.putExtra("nom", ((EditText) findViewById(R.id.editText)).getText().toString());
46                  startActivity(intent);
47              }
48          });
```

Dans la ligne 45, on a passé à la "secondeActivity" un paramètre intitulé "nom" et contenant la valeur du nom saisi.
2-On va dresser une zone de texte dans le layout de la seconde activité pour avoir finalement le contenu suivant :

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:tools="http://schemas.android.com/tools"
4       android:layout_width="match_parent"
5       android:layout_height="match_parent"
6       android:paddingBottom="16dp"
7       android:paddingLeft="16dp"
8       android:paddingRight="16dp"
9       android:paddingTop="16dp"
10      tools:context="com.example.cenaffe.formationcenaffe.SecondActivity">
11      <TextView
12          android:layout_width="wrap_content"
13          android:layout_height="wrap_content"
14          android:id="@+id/textView" />
15  </LinearLayout>
```

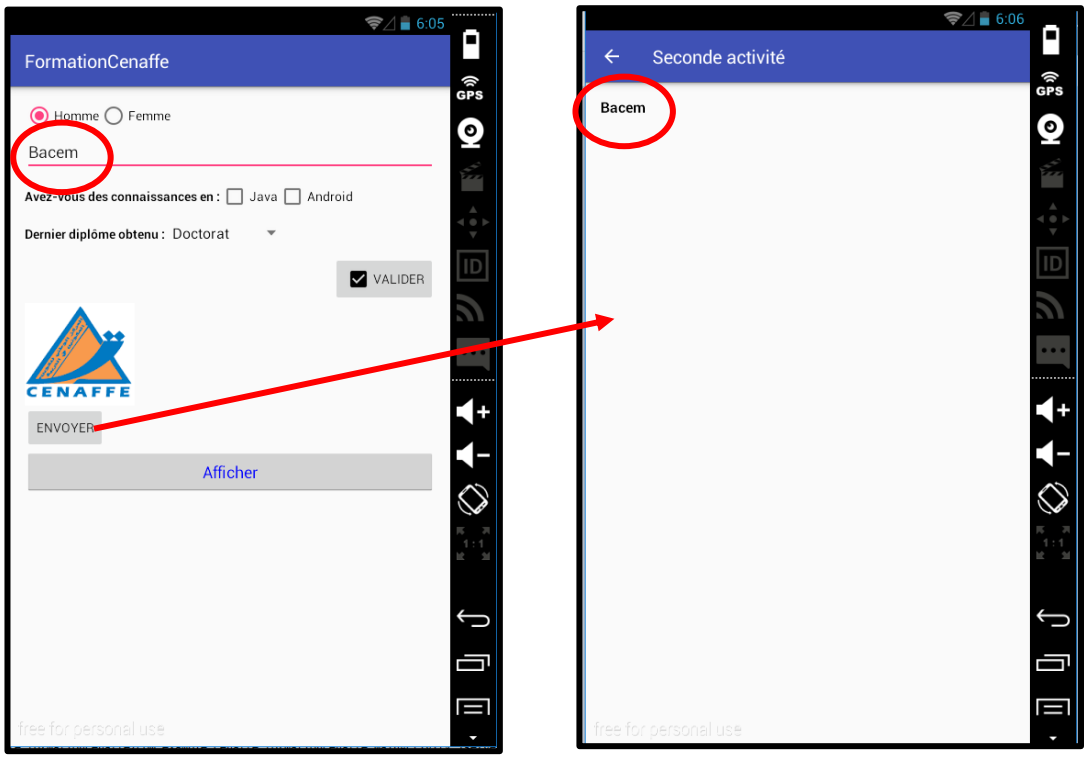
3-On récupère le paramètre "**nom**" passé avec la méthode **putExtra** par le biais d'une méthode intitulée **getStringExtra** comme suit :

Intent intent = **getIntent**;
String nom =**intent.getStringExtra**("**nom**");

4-On place le contenu récupéré dans la zone de texte d'identifiant "textView" pour avoir finalement le code suivant de la méthode **onCreate** de la seconde activité :

```
8  public class SecondActivity extends AppCompatActivity {
9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.activity_second);
14         Intent intent = getIntent();
15         String nom =intent.getStringExtra("nom");
16         TextView varText = (TextView)findViewById(R.id.textView);
17         varText.setText(nom);
18     }
19 }
```

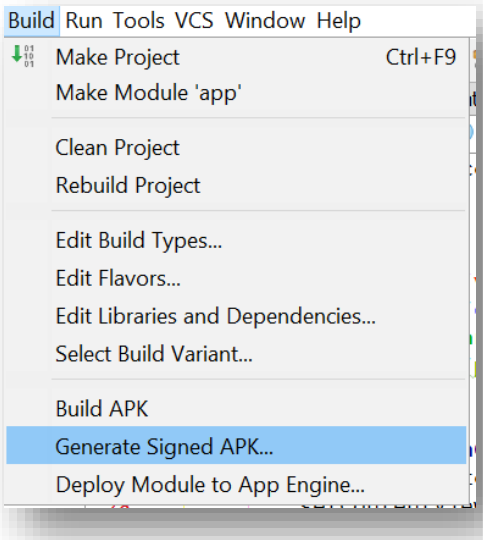
L'exécution de cette application permet d'avoir les aperçus suivants :



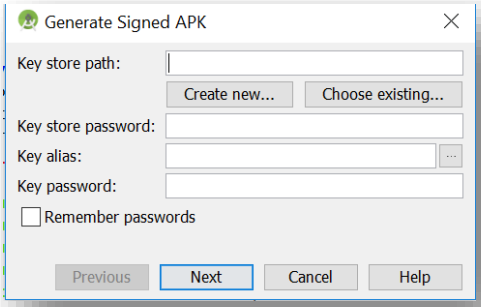
VII- Génération de l'APK de l'application

Toute application Android développée doit être exécutée sur des appareils téléphoniques d'où on a besoin de générer un fichier APK qui sera installé.
Pour générer le fichier APK d'une application en utilisant Android Studio on procède comme suit :

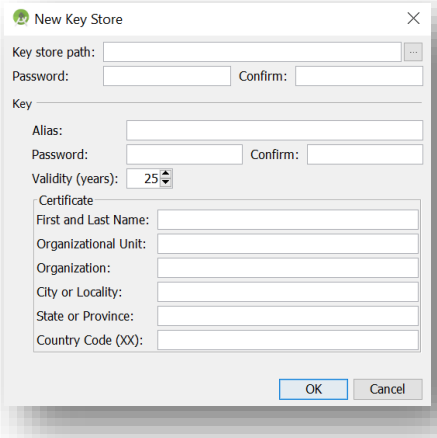
1- Choisir la commande "Generate Signed APK" du menu "Build"



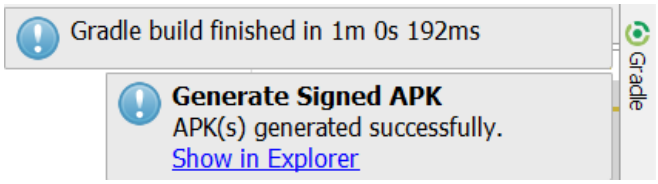
2- Créer un nouveau Key Store



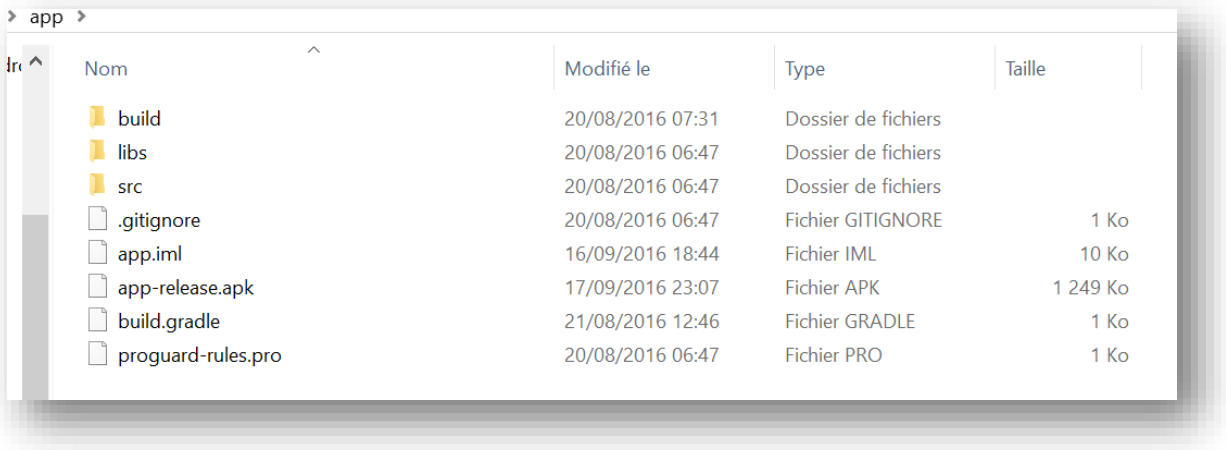
3- Enregistrer les données relatives au développeur



4- Une fois l'exécutable est prêt, un message indiquant la réussite de l'opération ainsi que le lien menant à l'emplacement du fichier APK généré.



5- Le fichier "**app-release.apk**" est le fichier généré à installer sur les appareils téléphoniques.



Section II :

Activités d’auto-formation

Activités d'auto-formation

Les cinq journées dédiées à la formation sont nécessaires pour présenter aux enseignants les notions de base du langage Android. Toutefois et afin de maîtriser ce langage orienté objet, il est nécessaire d'approfondir les connaissances acquises dans des activités plus complexes. La continuité de la formation est assurée à travers des activités d'auto-formation présentées ci-après.

Exercice 1 :

Développer une application Android permettant de remplir une liste par les noms des élèves hébergée sur un serveur distant. Ajouter une icône à chaque nom tout en lui développant un évènement d'écoute qui permet d'afficher les données appropriées.

Exercice 2 :

Développer une application Android permettant de remplir une liste par les données des personnes, ensuite envoyer un SMS à chaque élément sélectionné.

Exercice 3 :

Développer une application Android permettant d'ajouter un menu à une activité.

Conclusion

A travers le module de formation **«Programmation Android Niveau 1»** et dans la **première Section,** on a essayé de présenter les notions de base dans la programmation Android afin de permettre au formé une bonne maîtrise des savoirs de référence en relation avec le thème de la formation et ceci à travers :

- un exposé structuré et pertinent des connaissances de base et des méthodes de référence,
- des activités d'application et des travaux dirigés.

Dans la **deuxième Section**, des exercices plus complexes ont été proposées permettant au formé de développer ses capacités dans la programmation Android.

Suite à cette formation de niveau 1, le formé sera en mesure de développer des outils didactiques simples qu'il pourra exploiter en classe à travers les tablettes et les smartphones. Toutefois, ce contenu reste insuffisant et ne permet pas au formé de développer des applications Android et des outils didactiques complexes. Une formation de niveau 2 demeure obligatoire.

Bibliographie

1- Dawn Griffiths & David Griffiths. Head First Android Development : O'Reilly Media, 2015, 734 p.

2- Damien Guignard, Julien Chable, Emmanuel Robles. Programmation Android de la conception au déploiement avec le SDK Google Android 2 : Groupe Eyrolles, 2011, 486 p.

3- Jean-Francois Lalande. Développement sous Android [**en ligne**] : December 2015, Version 2.3, 105 p. Format Xml. Disponible sur : < http://www.univ-orleans.fr/lifo/Members/Jean-Francois.Lalande/enseignement/android/cours-android.pdf >

Webographie :

1- Tutos-android. Disponible sur http://www.tutos-android.com/maitriser-gradle-partie-1.

2- Android Studio The official IDE for Android. Disponible sur : https://developer.android.com/index.html.

3- Pluralsight. Disponible sur : http://app.pluralsight.com/courses/android-layout-fundamentals.

4- https://www.youtube.com/watch?v=QAbQgLGKd3Y&list=PL6gx4Cwl9DGBsvRxJJOzG4r4k_zLKrnxl

5- Android Application Developement. Disponible sur : http://amitandroid.blogspot.com/