Introduction au développement sous Android TP1
Année 2017-2018

Introduction

Ce premier TP est une initiation à l'environnement de développement Android Studio que vous

Utiliserez au cours de ce module. Il a pour but de vous familiariser avec cet environnement et d'illustrer les concepts du SDK Android, au travers du d´ développement d'une application mobile très simple.

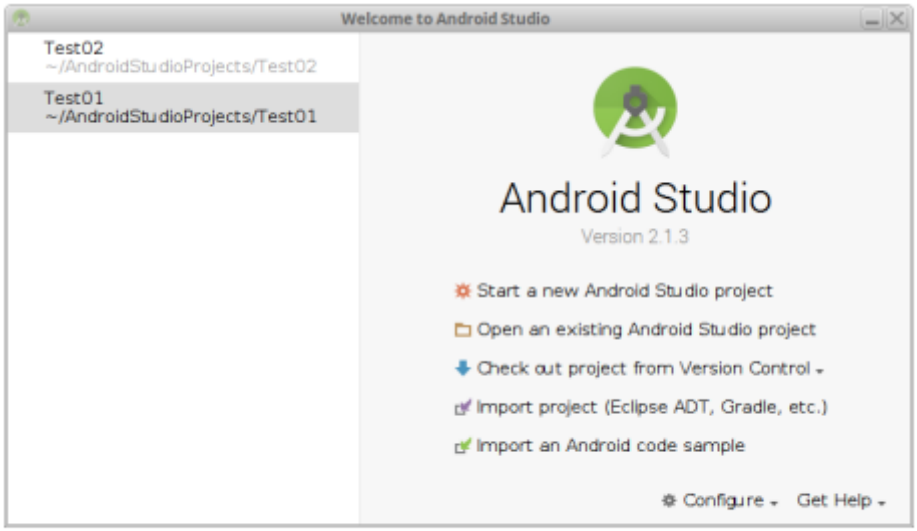


FIGURE 1 – La fenêtre de lancement initiale d'AndroidStudio

La première étape consiste ensuite à configurer votre nouveau projet, en particulier en le nommant et en spécifiant son emplacement sur votre compte. En vous inspirant de la figure 2, qui illustre la fenêtre de configuration qui apparat, complétez la configuration souhaitée :

- **Application name** : utilisez le nom Application01 qui apparaitra dans tout le reste de l'énonce.

A noter que le nom des applications doit commencer par une majuscule et qu'il faut éviter qu'il
Corresponde au nom d'une application disponible dans Google Play ;

- **Company Domain** : l'interface doit a priori vous proposer une chaine qui dépend de votre environnement. Si cette chaine n'est pas de la forme nom.univ-littoral.fr, dans laquelle nom correspond à votre login, effectuez la modification an que cela corresponde _a ce format ;

- **Package name** : l'interface vous propose un nom qui est issu des deux informations précédentes. Il n'est pas utile ici de le modifier ;

- **Project location** : par défaut, vos projets sont créés dans le dossier **AndroidStudioProjects** situes à la racine de votre compte. Il n'est pas nécessaire ici de modifier cette manière de faire.

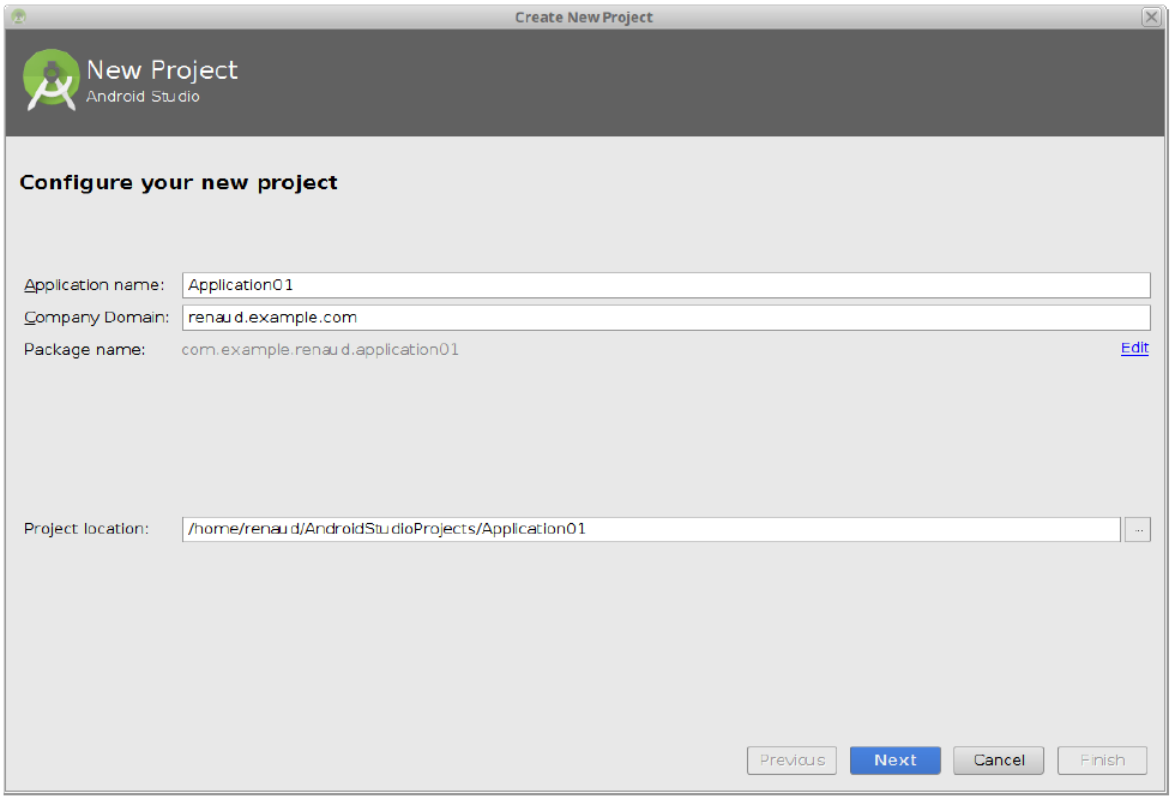


FIGURE 2 – La première étape de la création d'un nouveau projet

La seconde étape de la configuration d'un nouveau projet consiste à définir les architectures à destination desquelles vous développez votre application, ainsi que la version minimum du **SDK** à utiliser.
Comme le précise le petit texte apparaissant en figure 3, plus la version choisie est basse et plus votre application ciblera de périphériques. L'inconvénient est que les **SDK** plus récents disposent de davantage de fonctionnalités.

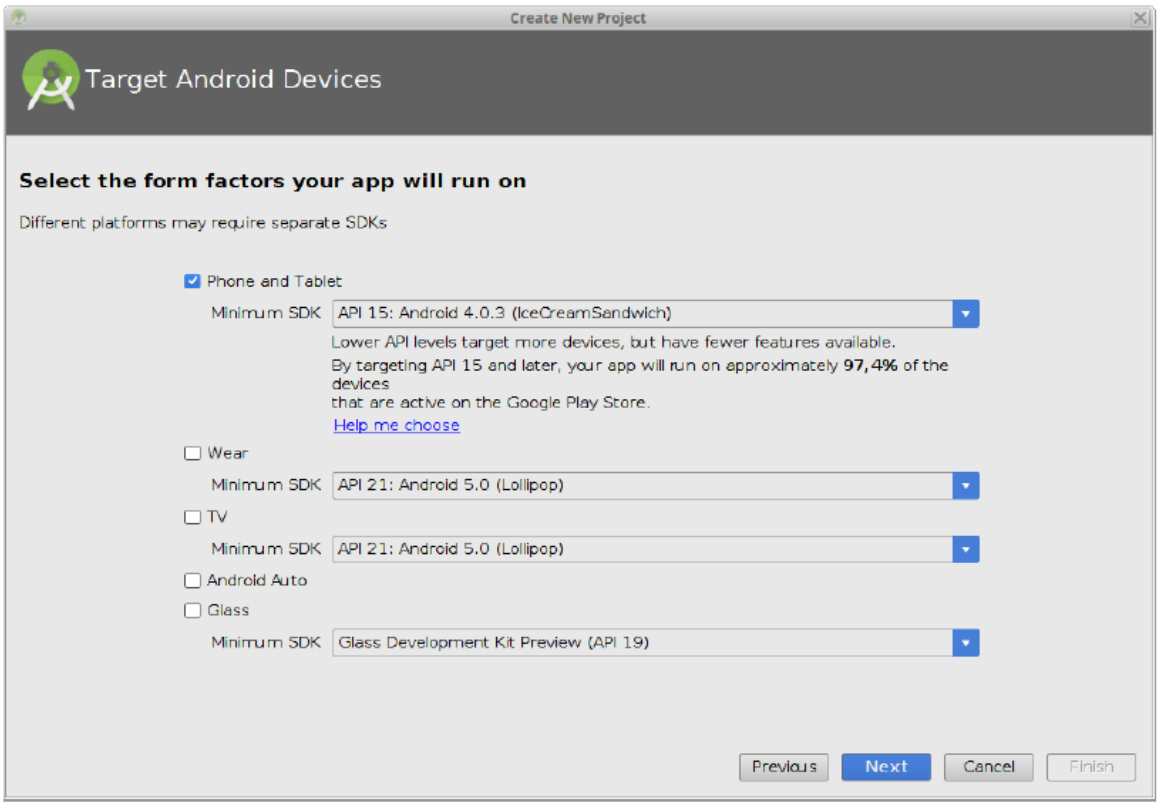


FIGURE 3 – La seconde étape de la création d'un nouveau projet

Dans le cadre de ce TP, nous choisirons l'**API 15** et ne ciblerons que les téléphones et les tablettes.
L'étape suivante consiste à sélectionner le type d'activité qui sera utilisé pour l'activité initiale de votre application. Comme vous le constatez sur la figure 4, vous disposez d'un choix assez large. Pour ce TP, nous utiliserons la **Empty Activity** _, qui correspond à un écran très simple.

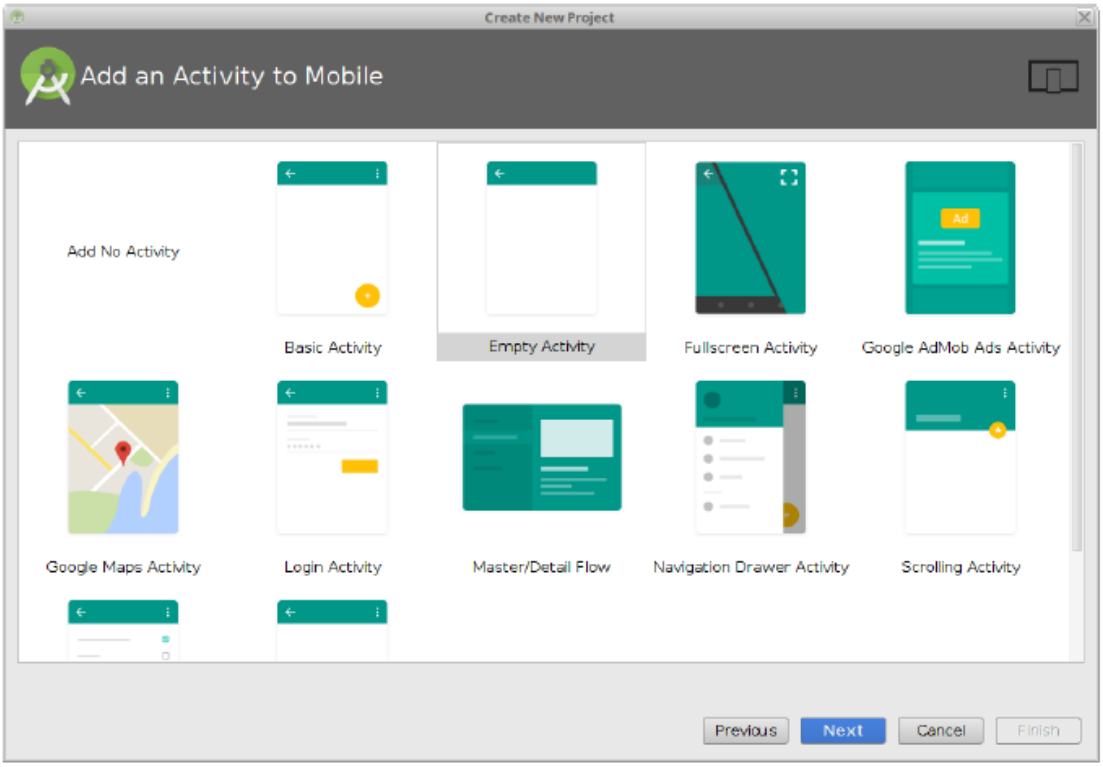


FIGURE 4 – La troisième étape de la création d'un nouveau projet

Il reste à configurer cette activité, en la nommant ainsi que ses attributs de base. Conservez pour le moment les choix par défaut qui sont proposés sur la figure 5.

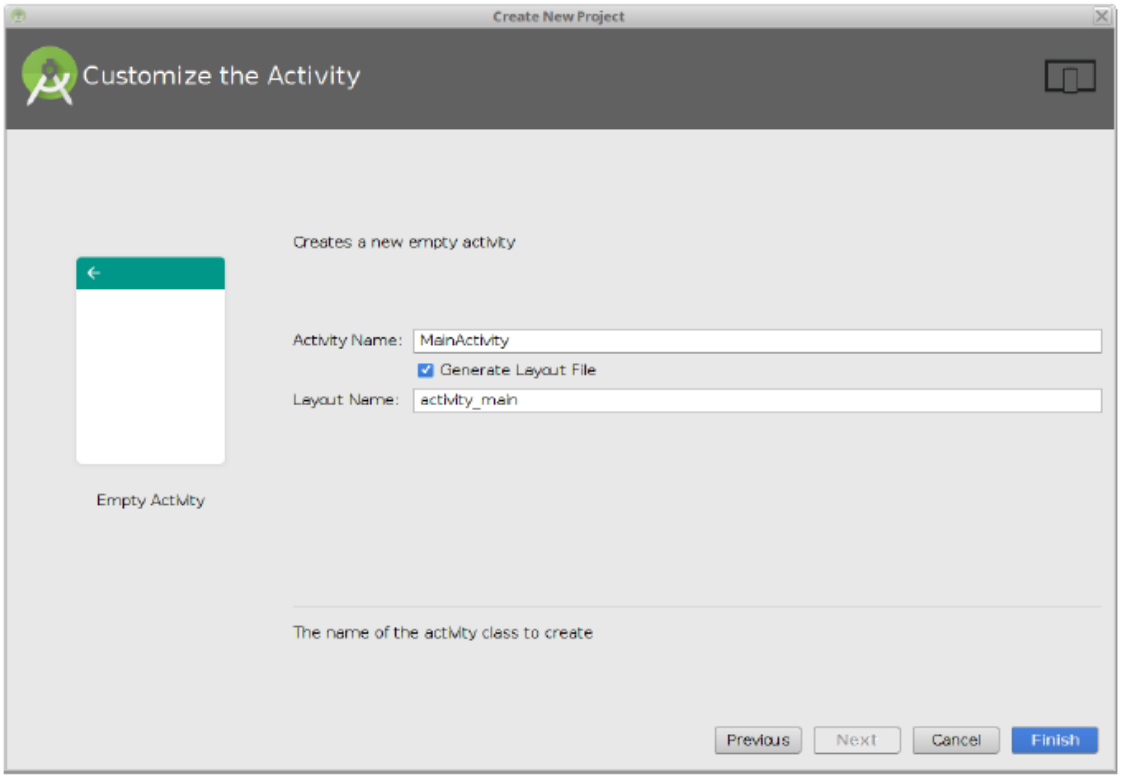


FIGURE 5 – La quatrième étape de la création d'un nouveau projet

Les différents noms à attribuer correspondent _a :

- **Activity Name** : nom de l'activité, qui correspond au nom de la classe Java générée ;
- Layout Name : le nom de l'écran (layout) associe à l'activité, qui sera repris comme nom de _chier

XML pour la configuration de ce dernier ;

Apres validation de ce dernier écran de configuration, vous voyez s'ouvrir l'IDE d'Android Studio, avec l'ensemble des fichiers génères par défaut (cf. figure 6). L'interface est initialement découpée en deux parties adjacentes, l'une pour l'éditeur (partie droite), l'autre pour l'arborescence du projet (partie gauche).
Le nombre de zones présentes dans l'**IDE** évoluera évidemment en fonction de vos actions et de vos choix.

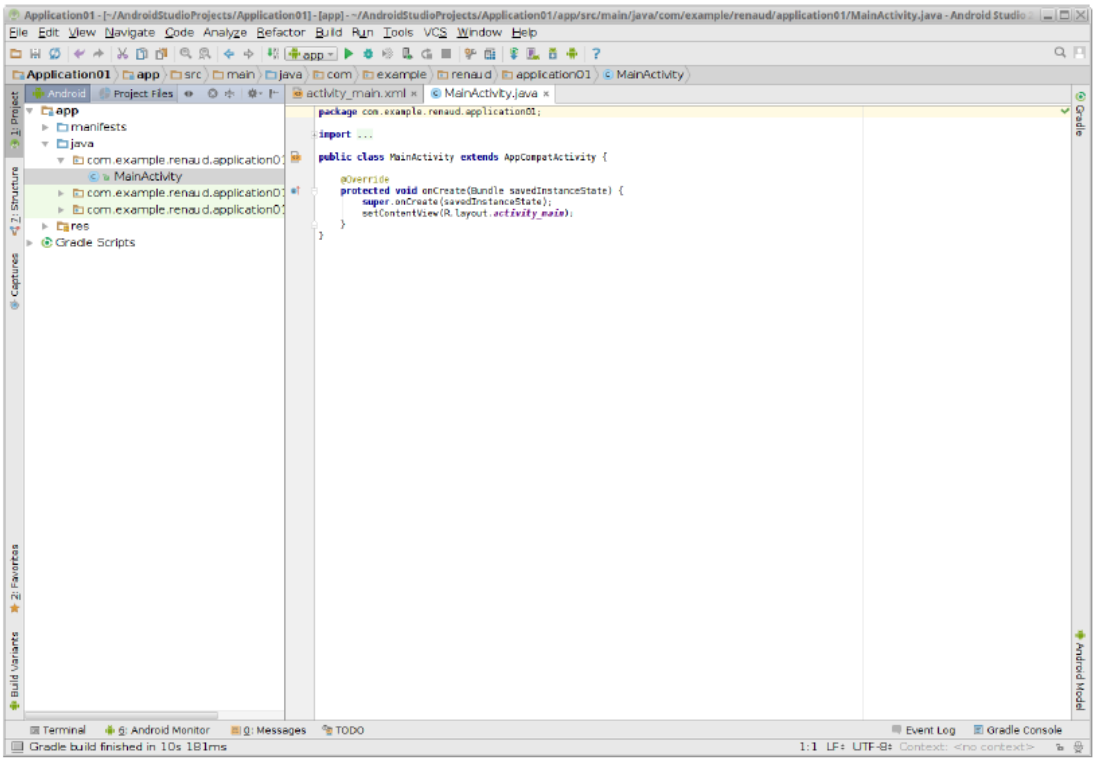


FIGURE 6 – Vue initiale de l'IDE après création du nouveau projet.

Compilation et lancement de l'application

La création d'un nouveau projet via Android Studio génère une application par défaut, compilable et exécutable. Son exécution produit la création d'une seule activité et de son _écran associe, et produit l'affichage du texte **Hello world !** ...

Compilation

Pour compiler uniquement votre application, plusieurs choix s'offrent à vous :

- Utiliser le menu **Build->Make Project** ;
- Utiliser le raccourci Ctrl+F9 ;
- Utiliser le bouton de compilation.

Choisissez l'option qui vous convient et compilez l'application. Vous devez voir apparaitre dans la barre de notification de la fenêtre (en bas ...) le message Gradel build running durant la phase de compilation, qui peut prendre quelques secondes.

Exécution

Pour lancer votre application, vous avez également plusieurs choix :

- Utiliser le menu **Run->Run app** ;
- Utiliser le raccourci **Maj+F10** ;

- Utiliser le bouton de lancement.

Notez que si l'application n'a pas été compilée, une demande d'exécution lancera préalablement la phase de compilation…

Choisissez l'option qui vous convient et lancez l'exécution de cette application. Android Studio ouvre alors une fenêtre vous permettant de choisir le périphérique sur lequel doit s'exécuter l'application (cf. Figure 7).

- **Connected Devices** : si un périphérique physique (tablette, smartphone) est connecte à votre ordi- nateur il apparat dans la partie supérieure et vous pouvez y lancer l'exécution de votre application.

Dans le cas contraire (qui correspond à ce qui est présente sur la figure 7, un message vous indique qu'aucun périphérique n'est connecte ;

- **Available Emulators** : Si un émulateur d'un périphérique physique a été installé, vous pouvez lancer l'exécution de l'application sur celui-ci, divers périphériques virtuels étant disponibles selon la configuration d'Android Studio dont vous disposez.

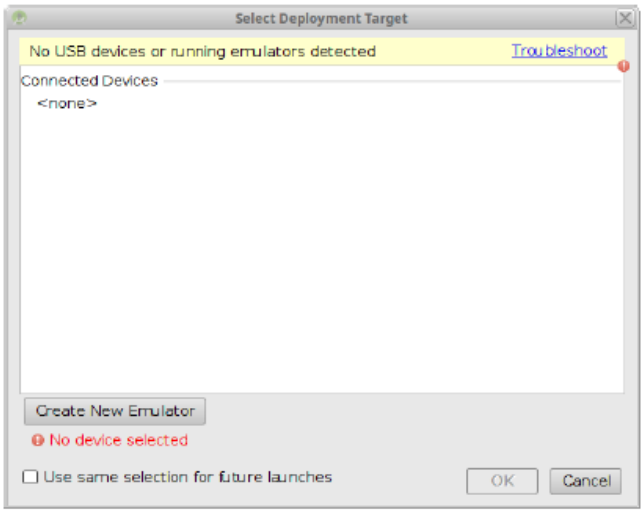
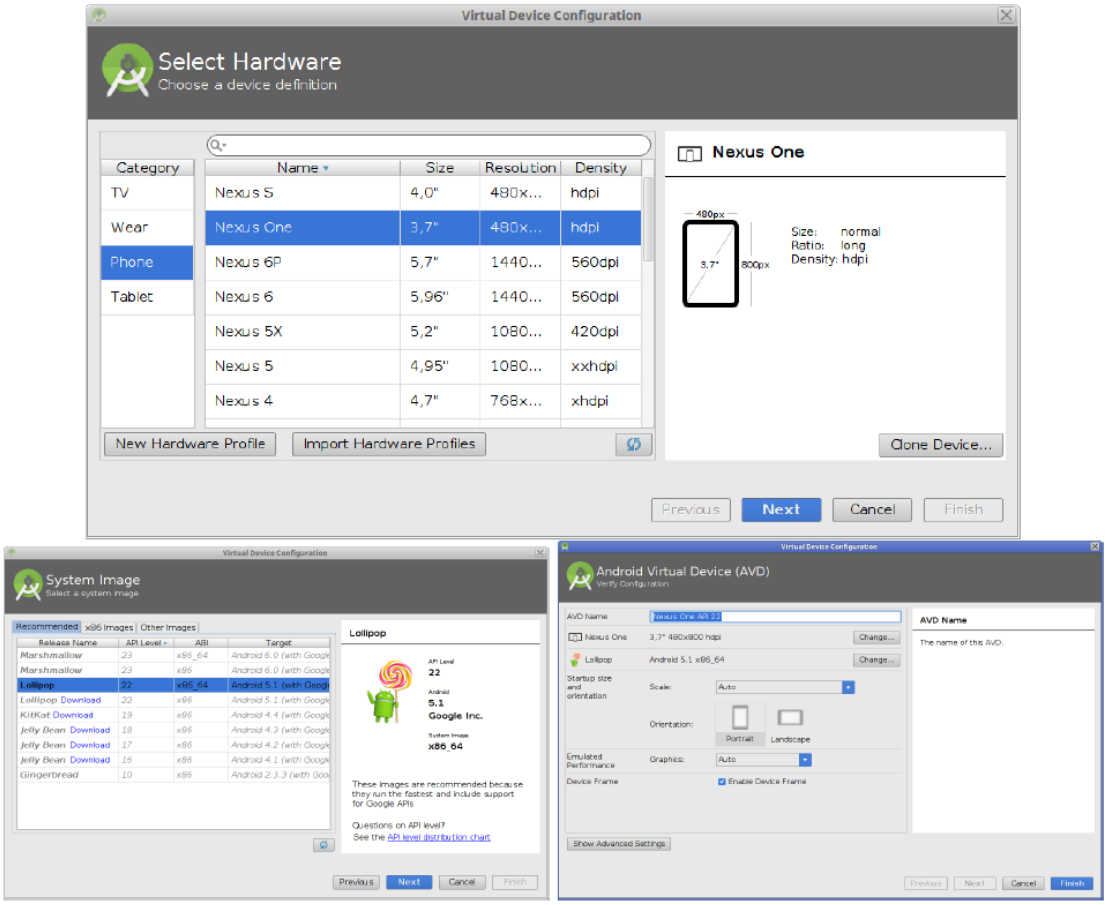


FIGURE 7 – Vue de la fenêtre de choix du périphérique à utiliser.

Dans le cadre de ce premier TP, aucun émulateur n'a été installé. Pour pallier cette absence, utilisez le bouton **Create New Emulator** pour créer un nouvel émulateur qui sera installé sur votre compte.
Choisissez l'émulateur du NEXUS One, puis validez votre choix en suivant les écrans qui apparaissent dans les captures d'écrans de la figure 8. Ne modifiez rien pour le moment au paramétrage du périphérique.

Notez également que vous pouvez à tout moment accéder à l'interface de gestion/création de périphériques virtuels via l'AVD **Manager** (**Android Virtual Devices Manager**), en utilisant le bouton.



Apres validation du périphérique à utiliser, une console d'exécution s'ouvre dans la partie inférieure de l'IDE (cf. figure 9) et après un temps plus ou moins long, l'émulateur s'ouvre (figure 10a). Vous disposez alors d'un smartphone virtuel, que vous pouvez utiliser comme un périphérique physique via la souris de votre machine.

Remarque importante : Gardez l'émulateur ouvert (ou réduit) après l'avoir lancé, de manière à ne pas avoir de délais d'attente longs dès que vous voulez tester une nouvelle modification de votre application.

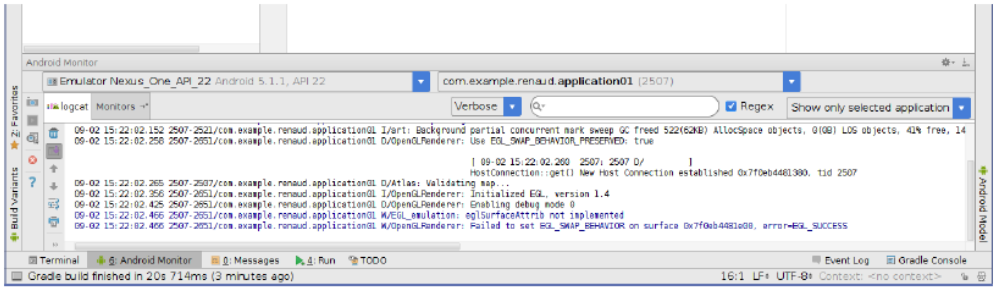
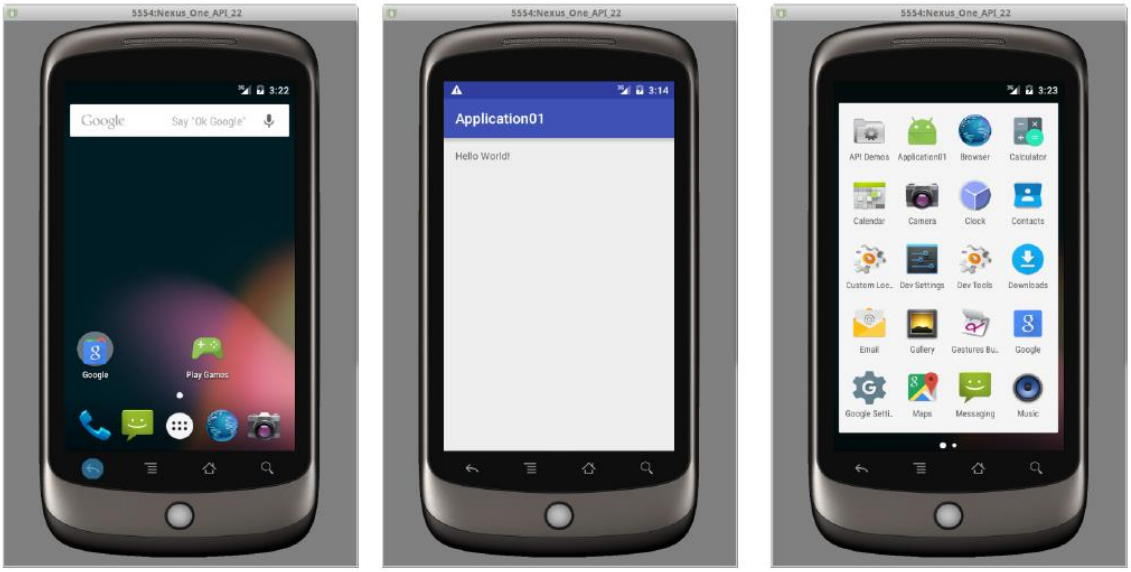


FIGURE 9 – Vue de la partie inférieure de l'IDE, avec la console de l'émulateur.



(a) Vue de l'émulateur après initialisation d'Android.

(b) Vue de l'application.

(c) L'application dans la liste des applications.

FIGURE 10 – Différentes vues de l'émulateur (ici celui du NEXUS 5).

Après avoir éventuellement _débloque _votre périphérique, vous voyez apparaitre votre application, comme l'illustre la figure 10b. Notez que votre application est disponible dans la liste des applications présentes sur le périphérique virtuel, avec une icône par défaut (cf. figure 10c).

Exercice 1

Modifiez le nom de la classe de l'activité principal en renommant sous **ActivitePrincipale**.

Notez que le nom de la classe et le nom du fichier Java doivent être modifies en conséquence. Vous pouvez effectuer ces modifications soit dans le fichier Java, soit en changeant le nom du chier Java et en utilisant les fonctions de re-factoring disponibles dans l'IDE :

- Petite icone _ampoule _qui apparat dans le cas d'une modification directe du nom de la classe ;
- Click sur le bouton droit de la souris sur le nom du fichier Java présent dans l'arborescence en cas de modification du nom du fichier, suivi de **Refactor->Rename**.

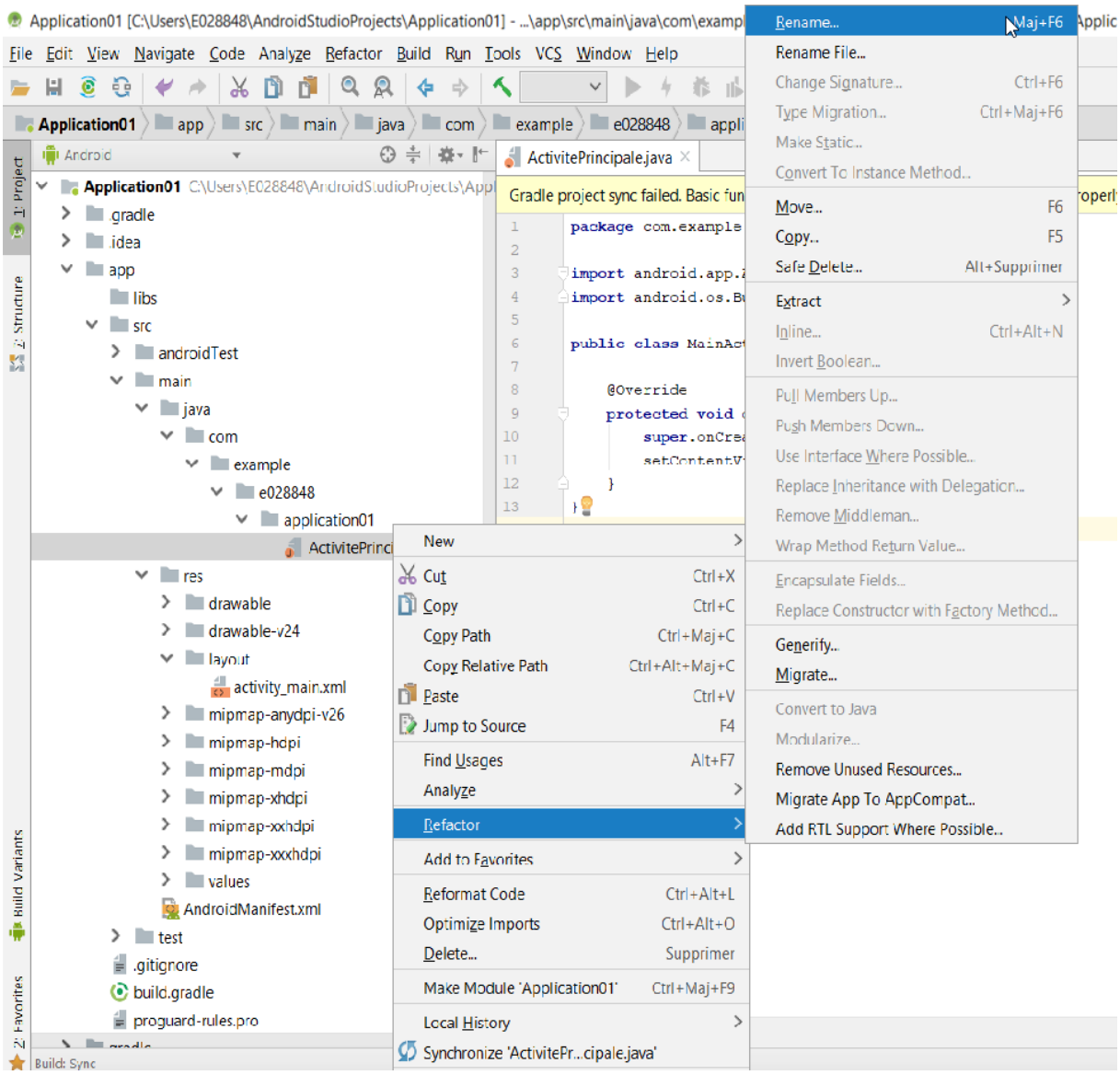


Figure 10.1 : Refactor ficher

Notez qu'il est également nécessaire de modifier le fichier **AndroidManifest**.xml pour tenir compte du changement de nom si vous utilisez la première option ...

Testez les deux possibilités en variant à chaque fois que toutes les modifications sont correctes, puis compilez et testez votre application.

Exercice 2

Le message qui apparait lors de l'exécution de l'application. Dans le fichier main activity.xml, le message **Hello world !** Apparat dans la balise **TextView**. Vous pouvez des lors modifier son contenu, par votre Prénom et nom Example : Cédric Le Belge.

La manière de gérer et modifier le contenu de ce texte va cependant cacher les possibilités offertes par Android de gérer plusieurs langues, en centralisant par exemple les chaines constantes dans le fichier strings.xml et ses dérivées. Il est donc plus judicieux de déclarer cette chaine de texte dans le fichier strings.xml et d'établir un lien entre la variable ainsi dénie dans le fichier main activity.xml.

1. Créez une variable app message dans le fichier strings.xml, en l'initialisant avec le texte Coucou le Monde ! ;
2. Modifiez le **TextView** présent dans le Fichier activity_main.xml, pour établir un lien vers cette variable. Compilez et testez.

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="application_name">My Application</string>
    <string name="application_description">This is my first Android application!</string>

</resources>
```

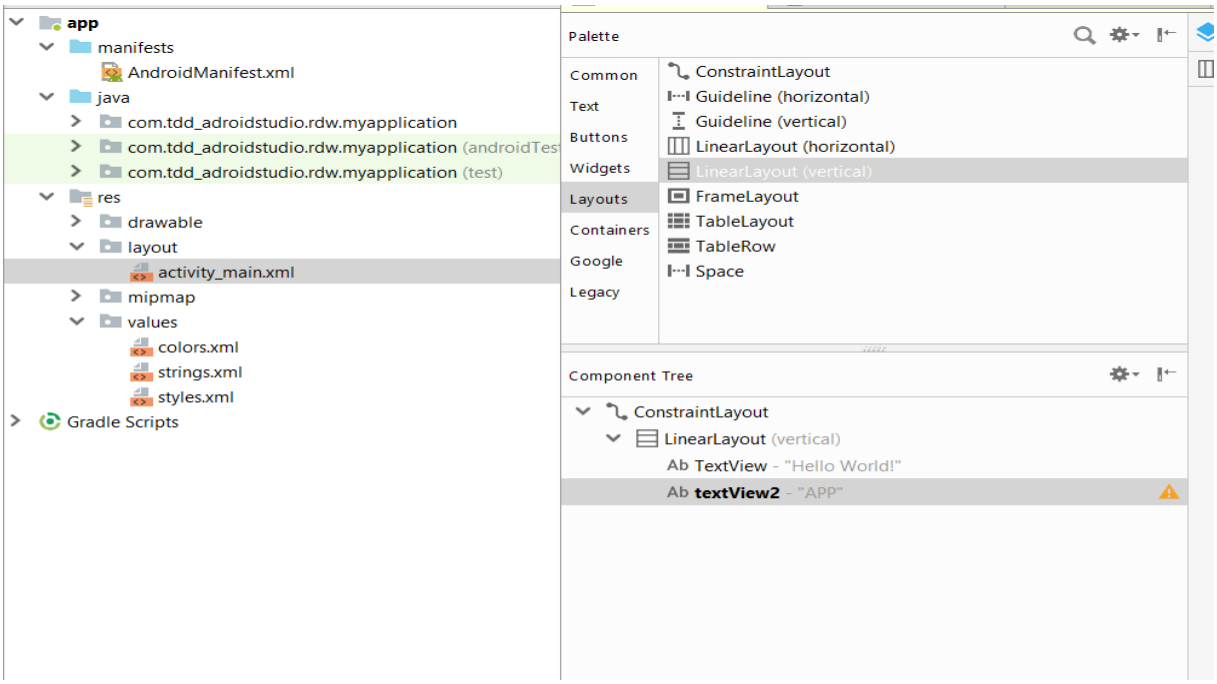


Figure 10.2 Exemple String.xml

Exercice 3

Vous allez à présent internationaliser votre application, en proposant 2 versions du texte qui y apparaît, l'une en anglais, l'autre en français. On rappelle que les textes, libelles, etc. sont définis dans le fichier **strings.xml** qui est utilisé par défaut. Ce fichier se trouve dans le dossier valu de votre application. Lorsque plusieurs versions linguistiques sont présentes, ce fichier doit être dupliqué et place dans un dossier portant le nom values-xx, le xx étant remplacé par un indentant représentant la langue (**fr** pour le français, **en** pour l'anglais, etc.)

Exercice 4

On souhaite modifier l'icône de lancement de l'application. Dans un premier temps, récupérez l'image disponible avec le sujet et rangez-la quelque part sur votre compte elle représente une icône **Android en 3D**, avec une résolution importante.

Sous Android Studio, cliquez sur le bouton droit sur le dossier app et sélectionnez l'option **New->Image Asset**. Cela a pour effet d'ouvrir un petit éditeur qui vous permet de sélectionner une image, d'y appliquer quelques effets simples et de générer des icones avec différents niveaux de détails (cf. figure 11). Lorsque vous êtes content de votre icône, il ne vous reste plus qu'`à la sauvegarder, en la renommant Android par exemple (pour éviter d'écraser l'icône précédente). Notez que l'image de départ doit avoir une résolution suffisante pour que l'éditeur d'icone puisse générer les déférents niveaux de détails. En cas de résolution insuffisante, les niveaux hauts seront plus ou moins pixélisés …

FIGURE 11 – Vue de la fenêtre de l'éditeur d'icônes en cours d'utilisation.

Il vous reste à modifier le fichier AndroidManifest.xml pour utiliser la nouvelle icone, compiler et enfin tester. A noter que vous pouvez supprimer toutes les icones non utilisées

en cliquant droit sur le dossier qui les contient et en sélectionnant l'option délecte. Effectuer cette opération sur

ic-launcher.png et notez au passage que l'IDE vérifie, avant suppression, que ces icones ne sont plus utilisées dans votre application.

Exercice 5

Vous allez à présent ajouter à votre application (fichier ActivitePrincipale.java) différentes méthodes du cycle de vie de l'activité et tester leur appel

- Ajoutez un affichage dans la console lorsque la méthode onCreate () est appelée ; vous pouvez utiliser pour ce faire un appel à **System.out.println**. Vérifiez qu'au lancement de votre application, cet affichage est bien génère ;
- Ajoutez la méthode **onStart**() suivante et de même insérez-y un affichage dans la console permettant de vérifier l'appel de cette méthode.

  protected void **onStart**(){
  super.onStart()
  }

  Compilez et testez ;
- Faites de même avec les méthodes **onStop**(), **onRestart** (), **onResume**() et **onPause**() ;
- Complétez votre application avec l'ajout et le test des méthodes :
  1. void **onSaveInstanceState**(Bundle outState)
  2. void **onRestoreInstanceState**(Bundle savedInstanceState).

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.content_main);
        Toast.makeText(getApplicationContext(), "First onCreate() calls", Toast.LENGTH_SHORT).show();
       }

    @Override
    protected void onStart()
   {
        super.onStart();
       Toast.makeText(getApplicationContext(),"Now onStart() calls", Toast.LENGTH_LONG).show(); //onSt
       }

}
```

Figure 11.1 : exemple méthode onStar()