

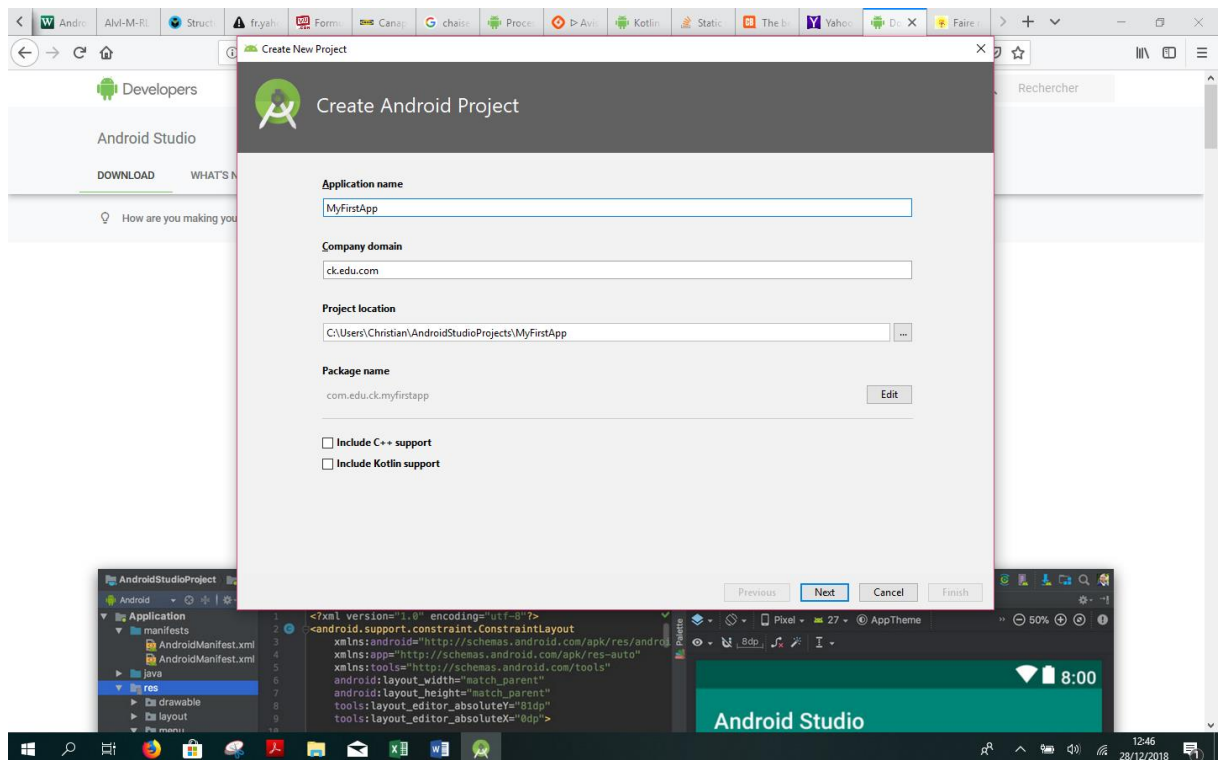
Android – A First Look

Introduction

In this document, we will introduce the different components in an Android Project step by step using Android Studio.

Creating The Project

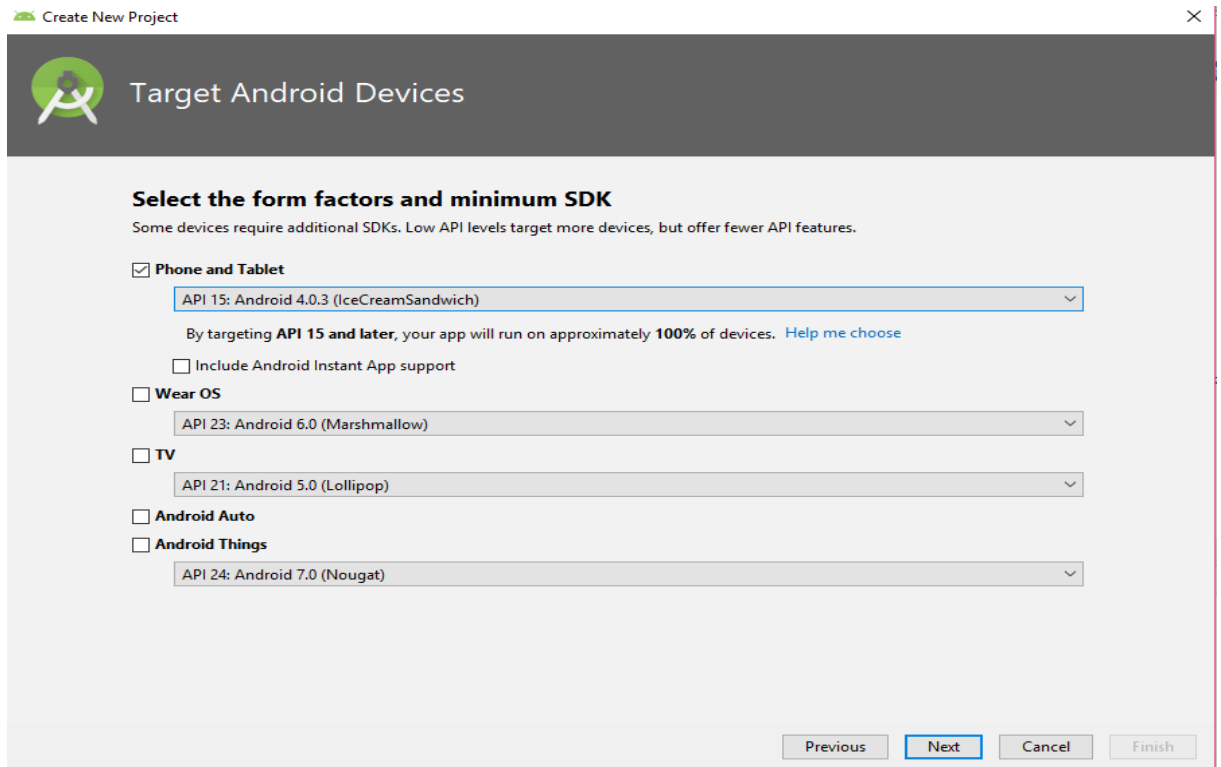
1. Download Android Studio from <https://developer.android.com/studio/>
2. Run it
3. Create a new project (follow the screenshots below)



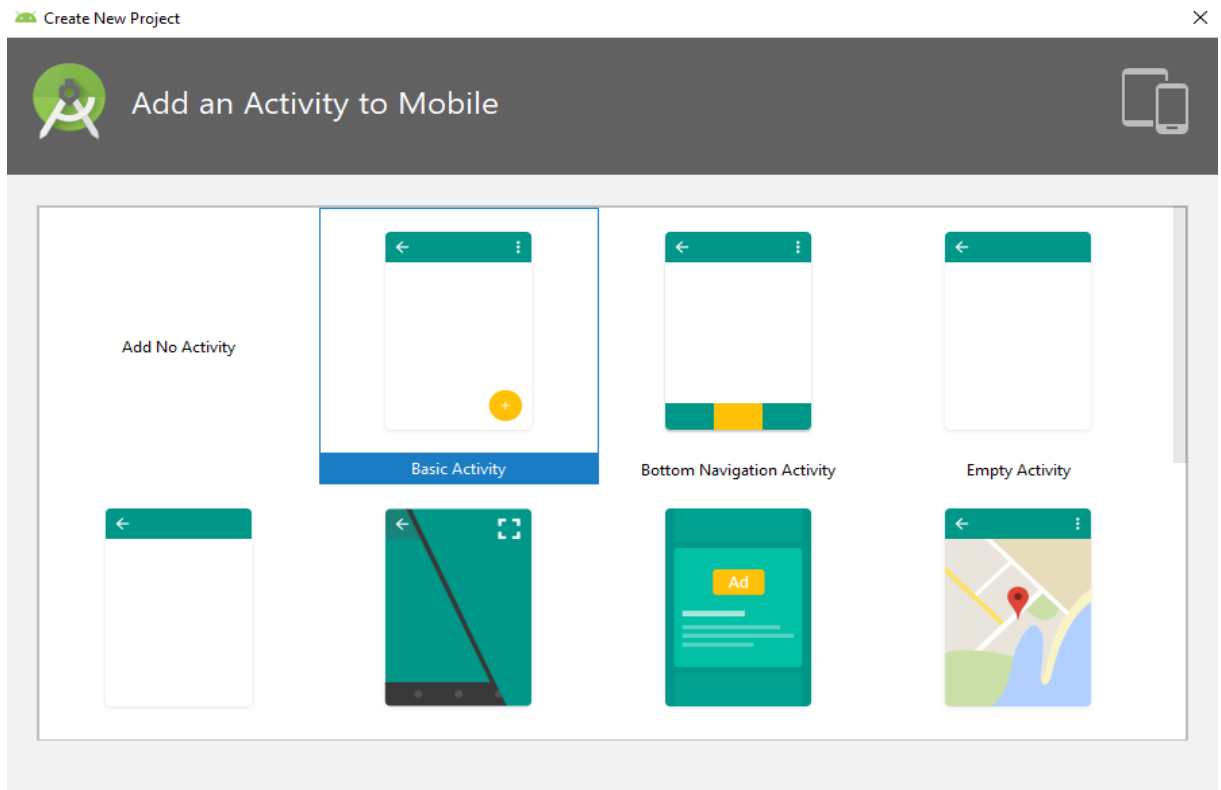
The company name is used as a prefix to the package name for the classes that are generated.

Package name = ck.edu.com.myfirstapp

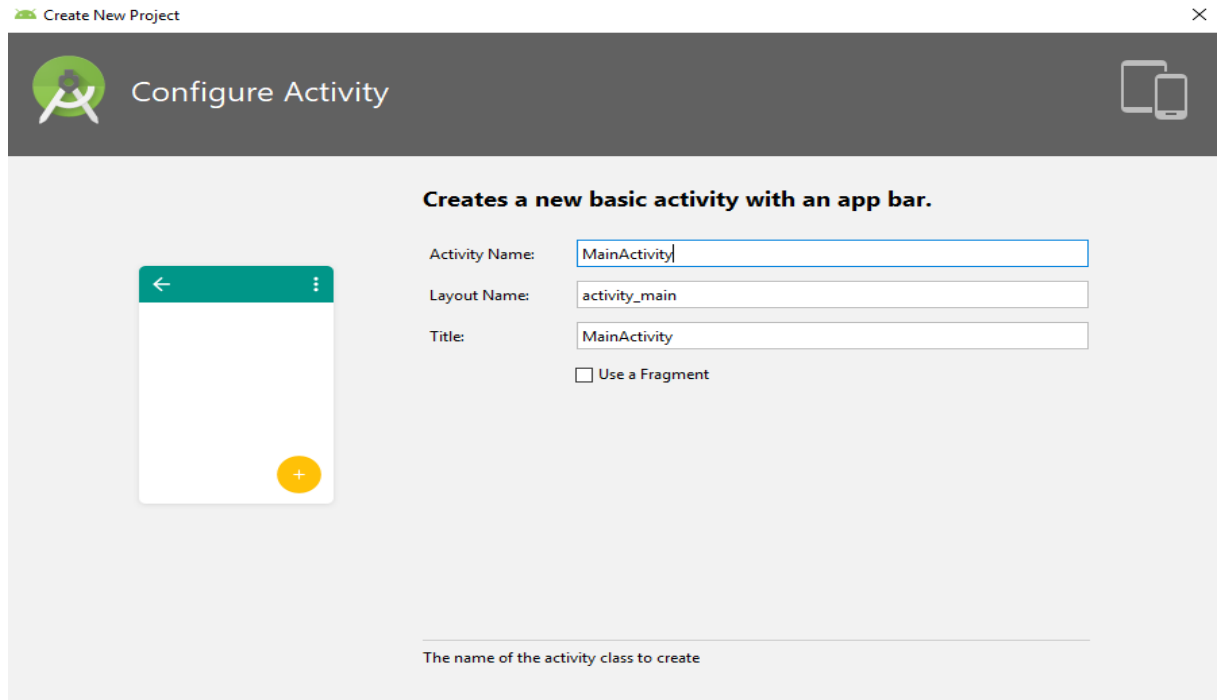
This is also going to be by default the application Id (found in build.gradle file after generating the project); the latter uniquely identifies an application on the device it's running on and on google play store.



This defines the minimal API that can be used to run the application. Obviously, new APIs can execute the application too (look inside the build.gradle file for this information after generating the project).



Choose an basic activity. The activity simply put in Android terms is a screen or an interface containing widgets. An application can contain multiple activities. In this case, we're just specifying the screen that shows first upon running the application.



Create New Project

Configure Activity

Creates a new basic activity with an app bar.

Activity Name: MainActivity

Layout Name: activity_main

Title: MainActivity

☐ Use a Fragment

The name of the activity class to create

The «Activity Name » is used as a class name.

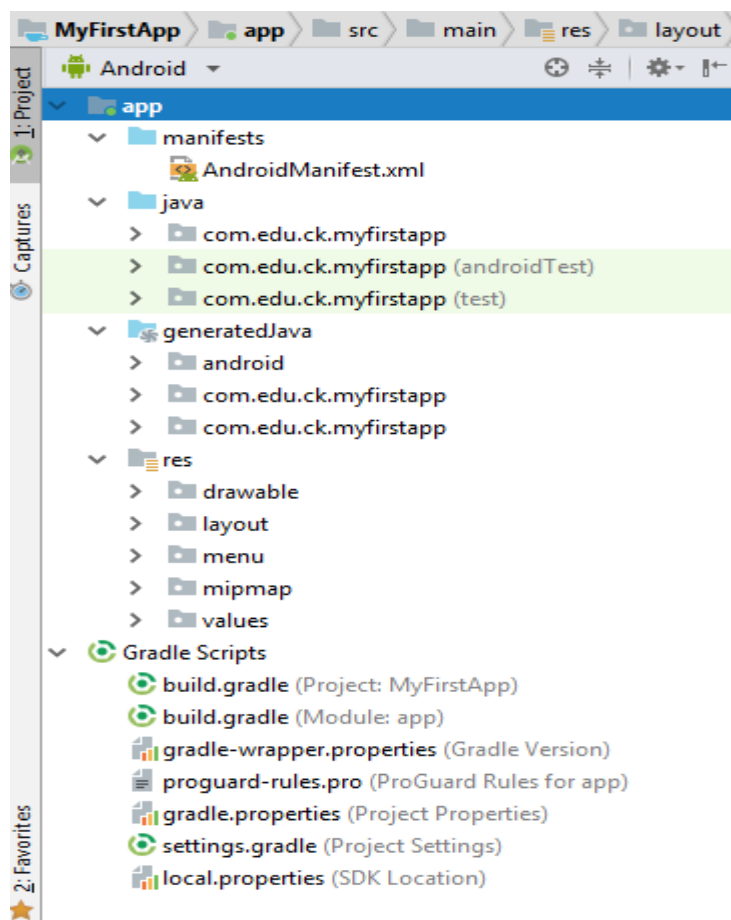
The « Layout Name » is used to define the content of the screen./activity/interface that is going to be defined.

The « Title » is used to name the application on the device (look it up once you install it) and to label the screen too once it shows on the device.

On the next screen, click on « Finish » and the project is going to be generated.

The general structure of an Android project is as follows :

1. Manifest part : general public information about the app
2. Code part (java in this case) : what the app is going to run in order to react to events
3. Generated/immutable part : the link between the xml parts (static) and the dynamic ones (the code)
4. Res stands for resources and contains all possible static resources (menus, logos, strings, dimensions, colours, ...)
5. Gradle scripts to build the app properly



1. Manifest file

It describes essential information about your app to the Android build tools, the Android operating system, and Google Play.

Among many other things, the manifest file is required to declare the following:

- The app's package name, which usually matches your code's namespace.

- The components of the app, which include all activities, services, broadcast receivers, and content providers. Each component must define basic properties such as the name of its Java class. It can also declare capabilities such as which device configurations it can handle, and intent filters that describe how the component can be started.
- The permissions that the app needs in order to access protected parts of the system or other apps. It also declares any permissions that other apps must have if they want to access content from this app.
- The hardware and software features the app requires, which affects which devices can install the app from Google Play.

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

PS : Namespace (xmlns attribute) android used (we find it as a prefix to all tags in the xml document)

The activity name gives the class name of the GUI class that has been generated, in this case MainActivity (the one we gave in one of the previous screens).

In the Manifest, there are some constants that are used and that have the following (simplified) syntax :

```
@<resource_type>/<resource_name>
```

Ex : @string/app_name

Look inside the res folder and inside the values part for « strings.xml » !

Here is the Java code that goes with the only activity we asked the system to generate, namely MainActivity.

```
12 public class MainActivity extends AppCompatActivity {
13
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.activity_main);
18         Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
19         setSupportActionBar(toolbar);
20
21         FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
22         fab.setOnClickListener((view) -> {
23             Snackbar.make(view, text: "Replace with your own action", Snackbar.LENGTH_LONG)
24                 .setAction(text: "Action", listener: null).show();
25         });
26     }
27
28     @Override
29     public boolean onCreateOptionsMenu(Menu menu) {
30         // Inflate the menu; this adds items to the action bar if it is present.
31         getMenuInflater().inflate(R.menu.menu_main, menu);
32         return true;
33     }
34
35     @Override
36     public boolean onOptionsItemSelected(MenuItem item) {
37         // Handle action bar item clicks here. The action bar will
38         // automatically handle clicks on the Home/Up button, so long
39         // as you specify a parent activity in AndroidManifest.xml.
40         int id = item.getItemId();
41
42         //noinspection SimplifiableIfStatement
43         if (id == R.id.action_settings) {
44             return true;
45         }
46
47         return super.onOptionsItemSelected(item);
48     }
49 }
50
51
52 }
```

It inherits from AppCompatActivity (lookup the class in the documentation). This class implements the default screen and lets us override some of its functions. The

latters are the ones called by the graphical engine when a state is reached by an object of this class. Beware that when the app runs, an object of this class is created as stated in the Manifest file.

Line 16

As its name suggests, `OnCreate` is called upon creation of an object of this class. It takes as an argument a bundle/package called `savedInstanceState`. We'll see later how this is used, but you can already foresee its usage for saving the content of a screen and then be able to restore it.

Line 17

`setContentView` is a function that populates the activity using an object identified by « `R.layout.activity_main` ». This identifier can be found under the « `generatedJava` » part of the project and references a file under the « `res` » part (resources). The content of this file is the following :

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".MainActivity">
8
9      <android.support.design.widget.AppBarLayout
10         android:layout_width="match_parent"
11         android:layout_height="wrap_content"
12         android:theme="@style/AppTheme.AppBarOverlay">
13
14         <android.support.v7.widget.Toolbar
15             android:id="@+id/toolbar"
16             android:layout_width="match_parent"
17             android:layout_height="?attr/actionBarSize"
18             android:background="?attr/colorPrimary"
19             app:popupTheme="@style/AppTheme.PopupOverlay" />
20
21     </android.support.design.widget.AppBarLayout>
22
23     <include layout="@layout/content_main" />
24
25     <android.support.design.widget.FloatingActionButton
26         android:id="@+id/fab"
27         android:layout_width="wrap_content"
28         android:layout_height="wrap_content"
29         android:layout_gravity="bottom|end"
30         android:layout_margin="16dp"
31         app:srcCompat="@android:drawable/ic_dialog_email" />
32
33 </android.support.design.widget.CoordinatorLayout>
```

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      app:layout_behavior="android.support.design.widget.AppBarLayout$ScrollingView..."
8      tools:context=".MainActivity"
9      tools:showIn="@layout/activity_main">
10
11      <TextView
12          android:layout_width="wrap_content"
13          android:layout_height="wrap_content"
14          android:text="Hello World!"
15          app:layout_constraintBottom_toBottomOf="parent"
16          app:layout_constraintLeft_toLeftOf="parent"
17          app:layout_constraintRight_toRightOf="parent"
18          app:layout_constraintTop_toTopOf="parent" />
19
20  </android.support.constraint.ConstraintLayout>

```

It describes the content of the screen. Generally, to describe the contents of an interface, we need the following :

1. Define a tree of elements to show, the inner nodes being containers and the leaves being simple widgets
2. Define the layout of all elements inside every container (specify the layout manager)
3. Define functions to react to events happening on the interface

In the above files, described using XML, there's the following tree of containers (layout managers in general with their attributes) and their (position, size, other attributes) tuple inside their corresponding higher level container.

```

CoordinatorLayout  -> AppBarLayout -> ToolBar
|
➔ ConstraintLayout -> TextView
|
➔ FloatingActionButton

```

Look up these layout managers and widgets in the android documentation.

Globally, « setContentView » reads the content of this STATIC description and instantiates the objects accordingly and positions them on the interface. Upon running this function, the interface is visible as defined to the user.

What is still missing in our description is how events are handled in our interface !

Line 21

We want to react to a click on the FloatingActionButton. Since this object has been instantiated from the XML description, there has to be a way to define its handler inside code. But to do that, we need to get hold of the reference to this specific object. And that's where the « findViewById » function comes in handy as it enables the programmer to get the reference to the object using a special unique identifier defined inside the XML document (R.id.fab in the code and android:id in XML).

Inside the XML file, each identifiable object has to declare its unique id by using the following syntax :

```
android:id= « @+id/myname »
```

This creates a new reference that identifies the object. This reference can be used elsewhere inside XML by omitting the « + » sign.

Line 22

We subscribe a listener to the click event on the special widget (this is usual java code for handling events)

Running The App

Either connect your android mobile (enable usb debugging) or create a virtual device (use AVD for that – search for it in the) and then run the app.

<https://developer.android.com/studio/run/managing-avds>