Android Studio Simple App Development



Level: Beginner

Developers: Edwin Wang, Van Nguyen, Lucas Hwang

Advisors: Professor Truong Nguyen, ECE

Project Sequence: Mobile App Development

Last Updated: 4/06/19

A NOTE FROM THE DEVELOPERS

Dear Students,

Insert Welcome text

INTRODUCTION

First, you need to install the latest version of Android Studio and learn how to program in either Java or Kotlin. These are the two primary languages for Android Studio, but this tutorial will be written with the assumption that Java will be used. After that, you will learn how to build a simple app in Android Studio. Its basic functions include a basic GUI and the ability to open the camera as well as save photos to the gallery. After setting up your app, you will learn how to integrate Firebase, a NoSQL database made for mobile apps built in Android Studio.

OVERALL LEARNING OBJECTIVES

* Java basics
* Familiarity with Android Studio
* TensorFlow Mobile

GENERAL TIMELINE:

|  |  |
| --- | --- |
| Part 1: Building the App | Challenge #1: GUI  Challenge #2: Opening Camera  Challenge #3: Saving Picture to Gallery |
| Part 2: Integrating Firebase | Challenge #4: Deploying Model  Challenge #5: Accessing From App |

REQUIRED PROJECT PARTS:

None

REQUIRED PROJECT TOOLS/EQUIPMENT:

* Laptop Computer using Windows
* Monitor
* Mouse
* Keyboard
* Optional - Phone, USB Cable

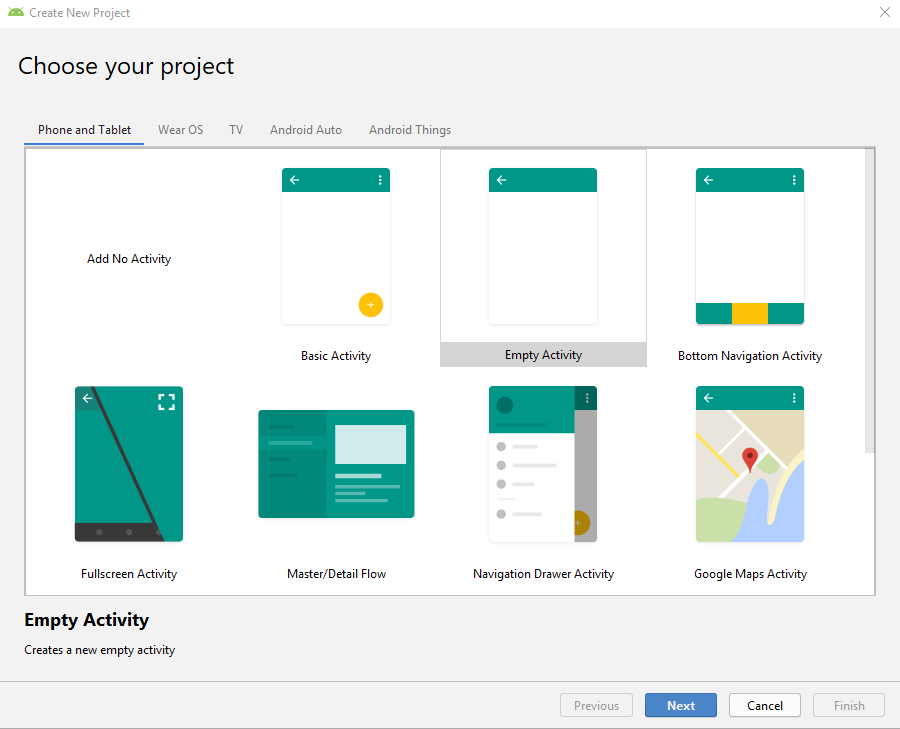
**CHALLENGE #1: GUI**

**Objective:**

You are going to create a simple GUI for your user to interact with as they use your app.

**Familiarizing Yourself with Android Studio:**

Go to <https://developer.android.com/studio> and install the latest version of Android Studio for Windows. After downloading, follow the directions in the setup tutorial and once done open up Android Studio.



After opening Android Studio, click **Start a New Project** and select **Empty Activity**.

**Configuring Android Studios:**

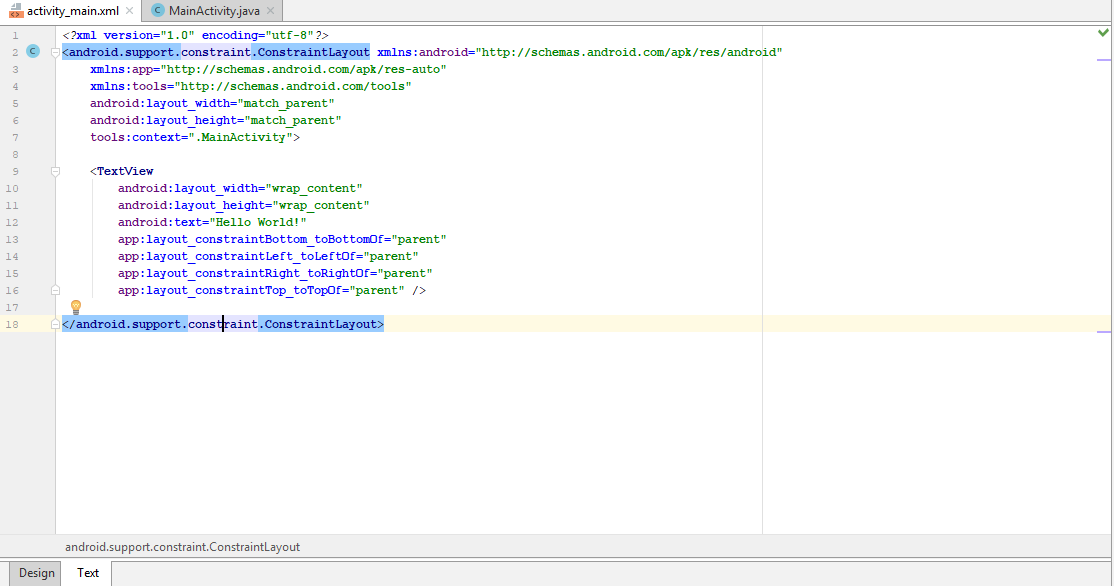
1. Create a Project Name
2. Specify the name of your Package( This package name will become your application ID, which you can change later)
3. Select Java as the Language option
4. Select a Minimum API Level which will run on a majority of devices

When you are done hit **Finish** to build your project.

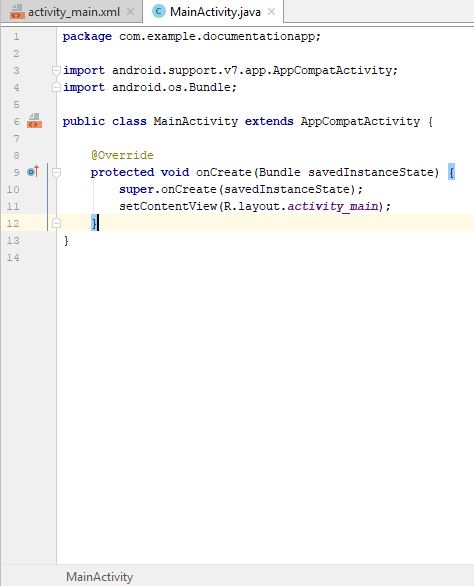
****

Design view of activity\_main.xml

After building your project, Android Studio should look like the picture above with two tabs. The first tab is your **activity\_main.xml**, which can be viewed in both **Text** and mode. This file represents the visual layout/GUI of your app. The **Text** view allows you to view the xml code which is responsible for building the GUI whereas the **Design** mode allows you to view what your app will actually look like. Additionally, any changes made to the GUI in one of the modes is reflected in the other mode. For example, if you increase the size of the button in the **Text** view, the increase in size will be reflected in the **Design** view.



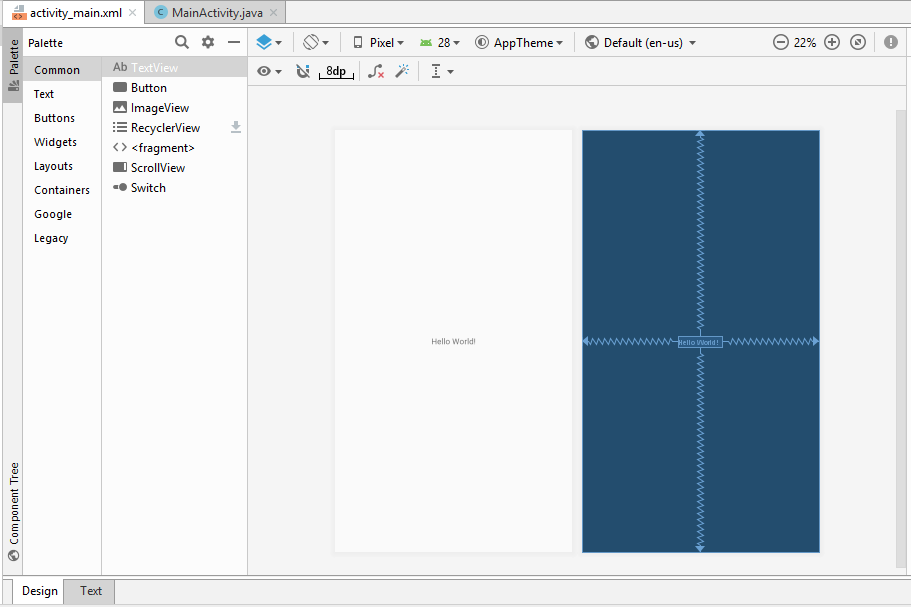
Text view of activity\_main.xml



MainActivity.java

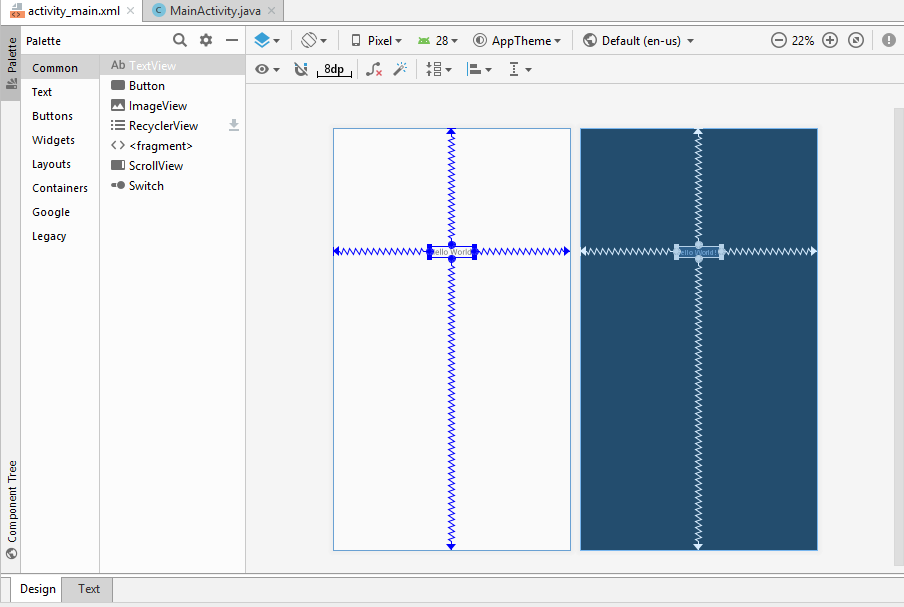
The other tab which appears at the top of your project is the **MainActivity.java** tab. This tab represents the java file which holds the code required to run any actions in your app. The starter code provided for any activity within Android Studio is the **onCreate** method. This method runs whenever an instance of your app is opened. The call to **setContentView** within **onCreate** makes it so the default screen the user sees when the app opens is **activity\_main**.

Let’s make a change to the empty activity by changing the text of the string “Hello World!” to . These two files, **activity\_main.xml** and **MainActivity.java**, are the two main files which you will be editing in order to build the simple app.



**Creating a Basic UI Layout:**

Using the layout editor, one can build layouts by dragging UI elements into the visual design editor instead of typing all the code by hand in the activitymain.xml file. Along with the use of the ConstraintLayout, you can easily manipulate any Text box or Image placement on the Design View.



Layout Editor Basics

1. Palette: List of views and groups that you can drag into your layout
2. Component Tree: View Hierarchy for your layout
3. Toolbar: Buttons to configure your layout appearance in the editor and to change some layout attributes
4. Design Editor- Layout in Design or Blueprint view, or both
5. Attributes - Controls for the selected view’s attributes



Colors Basics:

Android studio uses standard RGB model. Each color is represented by a hexadecimal

number in the format of #RRGGBB. For instance #000000 is black and #FFFFFF is white. However, hexadecimal values are hard to visualize. Therefore, android studio

allows you to define your own color. This is done through editing colors.xml.

Check if you have a colors.xml file in your resource directory. If not, create one by using the following header. The starter file should look something like this:

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

<resources>

</resources>

Then you can start naming your colors by using the following pattern:

<color name="some\_color">#0123456</color>

The completed file should look something like this:

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

<resources>

<color name="some\_color">#0FAABB</color>

<color name="colorPrimary">#FFA500</color>

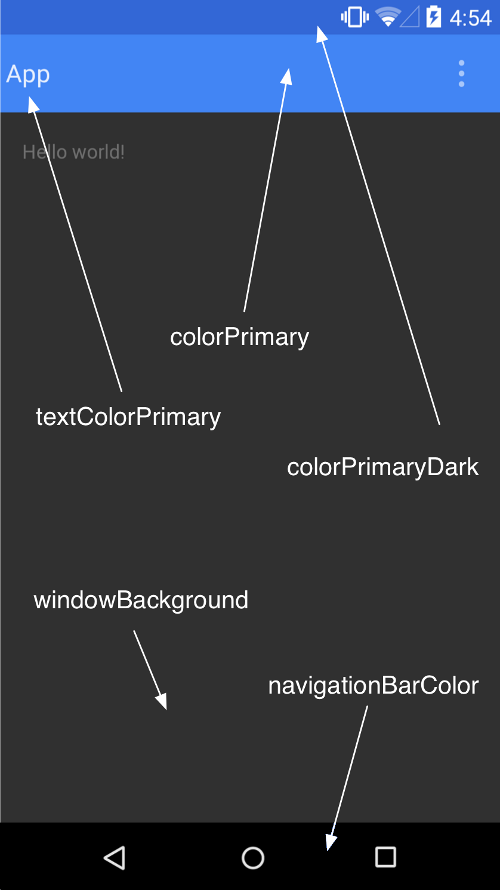
<color name="colorPrimaryDark">#D81B60</color>

<color name="colorAccent">#D81B60</color>

<color name="colorBackground">#D2691E</color>

</resources>

Note that you can name your color by your preference such as "dark\_blue" and "light\_green" or by their functionality like "menu\_color" and "button\_color". In fact, android studio already uses some color names in its default set up. Here is a picture showing the color names used by android studio.



Backgrounds:

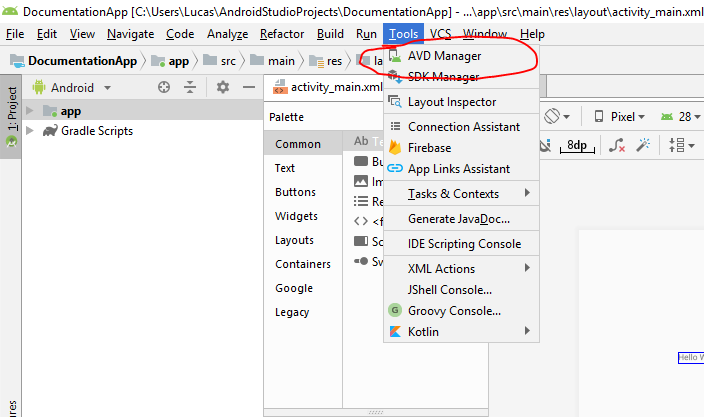
Using the settings mentioned above, you can change the background to any color of your preference. Android studio also allows you to use pictures for background. To change the look of your background, first load the image you chose for the background into the drawable directory. Then add the following line of code in the activity\_main.xml file.

android:background=”@drawable/imageName”

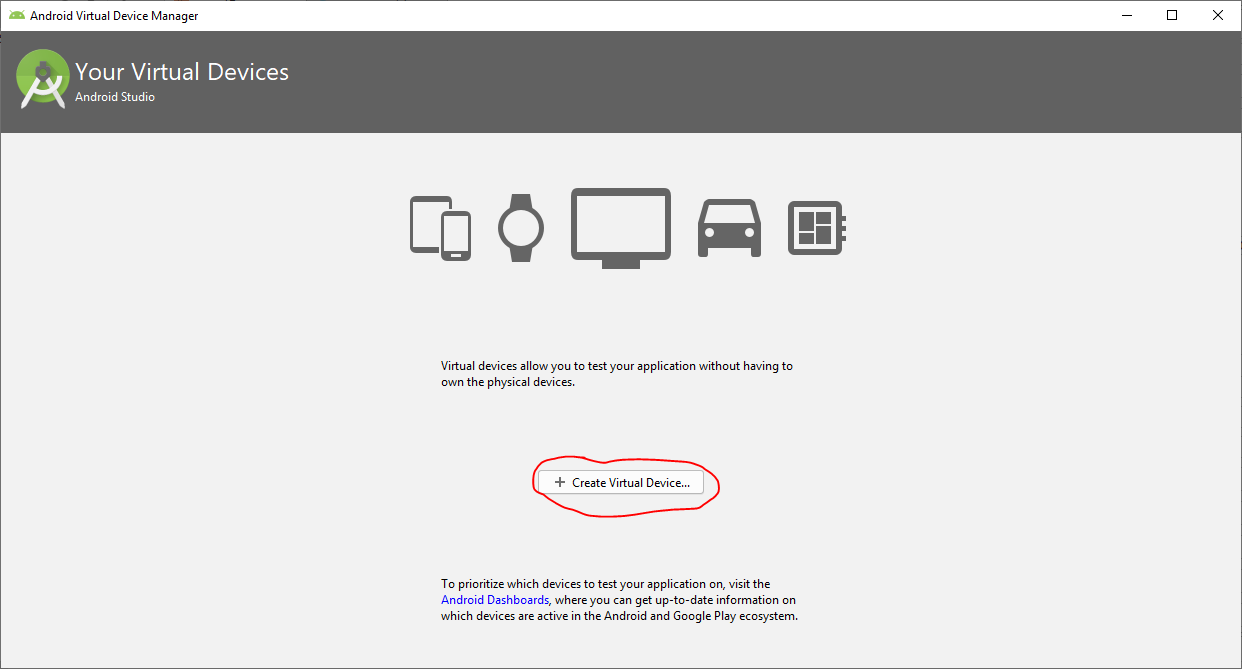
This will change the background to the image you loaded into the drawable directory.

**Using the Emulator:**

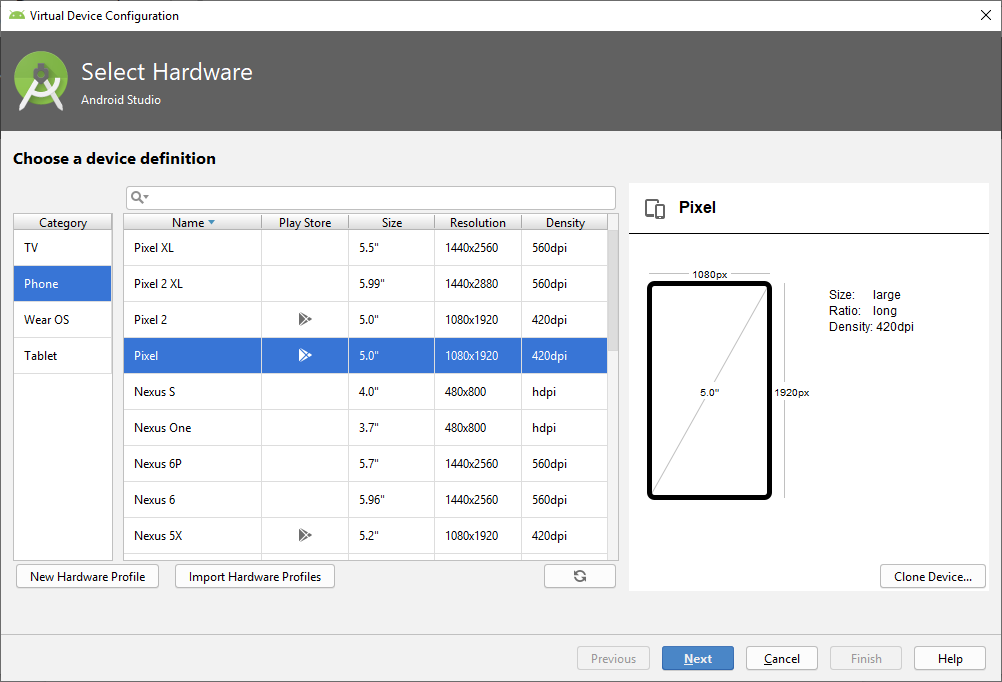
Android Studio provides two options of running your app. The first option is to plug in an Android phone and run the app on the phone. The second option is to use an emulator provided by Android Studio. In order to set up your emulator, navigate to the **Tools->AVD Manager**.



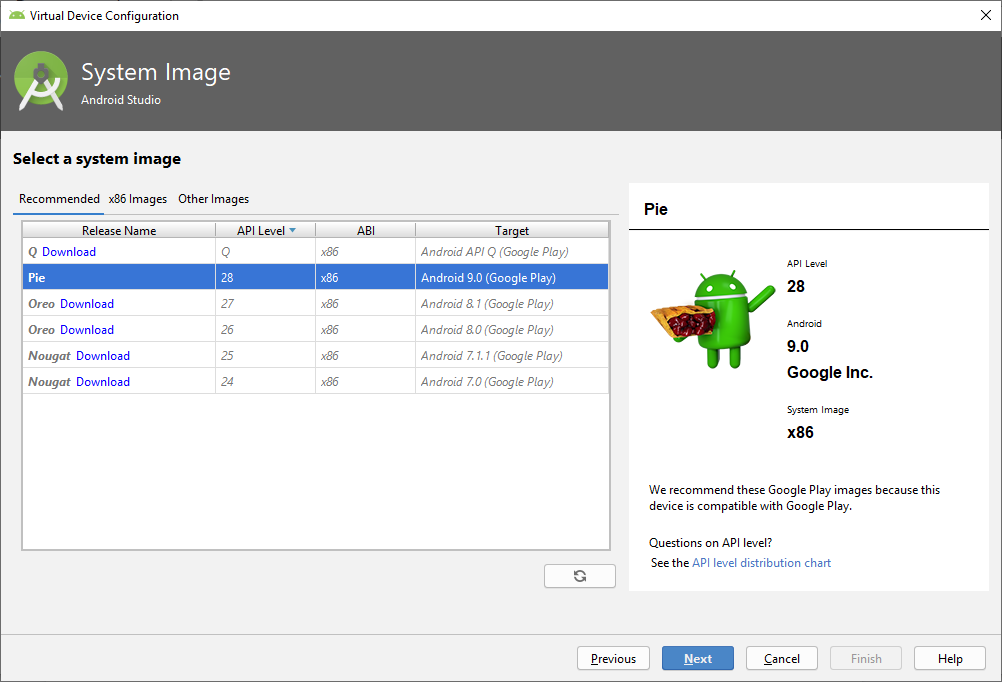
In the resulting window which pops up, click **Create Virtual Device**.



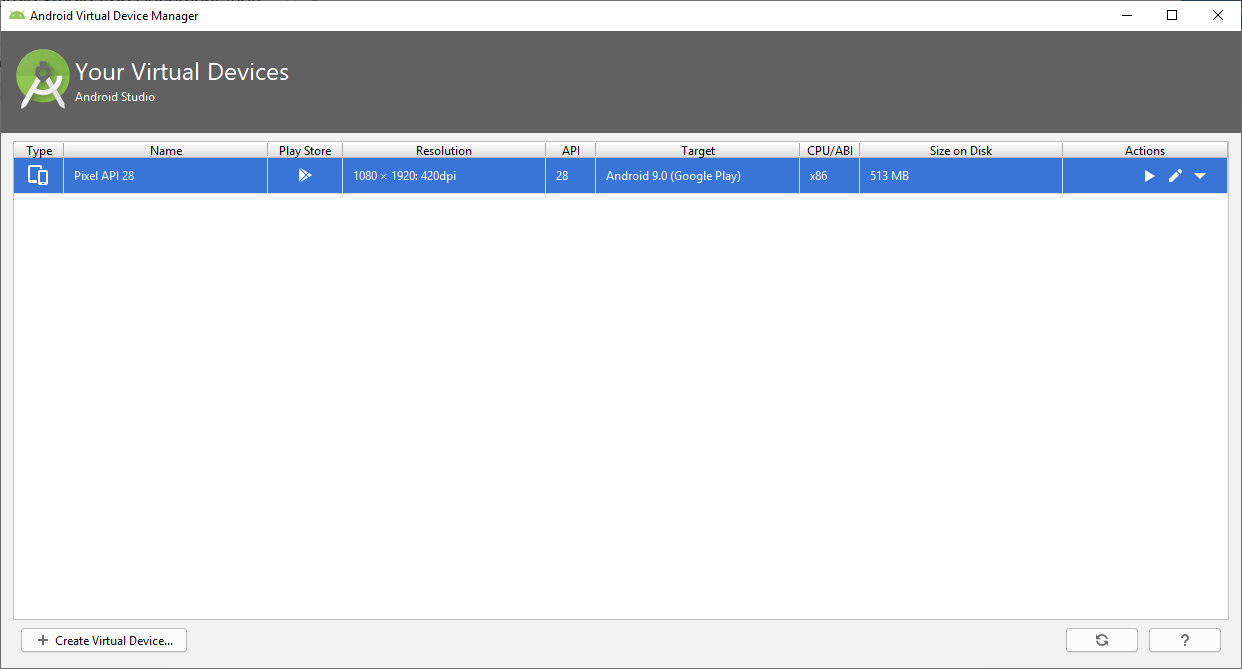
For our project we have been using the **Pixel**, but you can select any device which you want and then click **Next.**



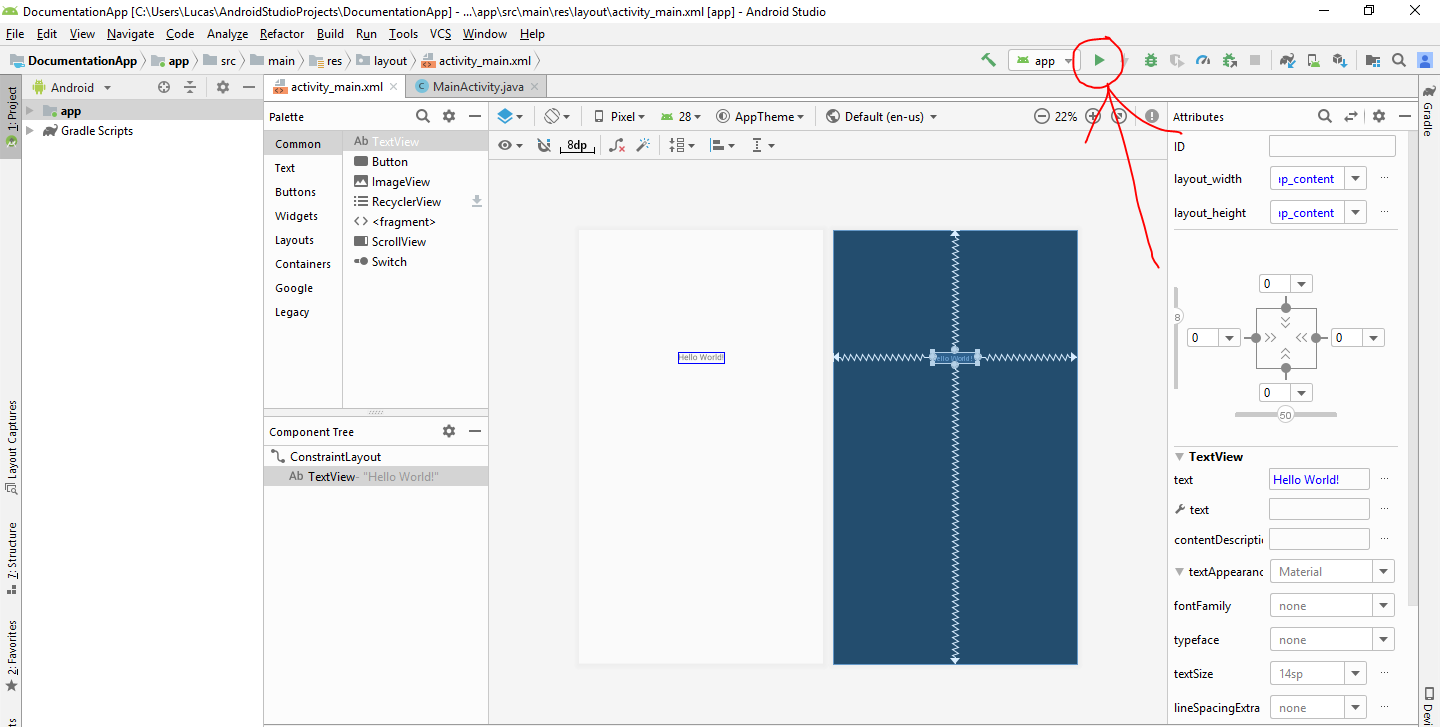
The next thing you will have to do is pick a version of Android to run on your emulated device. In this tutorial we will be using **Pie** with an **API level of 28**, but feel free to change to a different version of Android if you like. Different versions of Android are compatible with different phones so the biggest thing to remember is to make sure your version of Android is compatible with whatever phones you eventually want to port the app to. After you select your version of Android, hit **Next**.

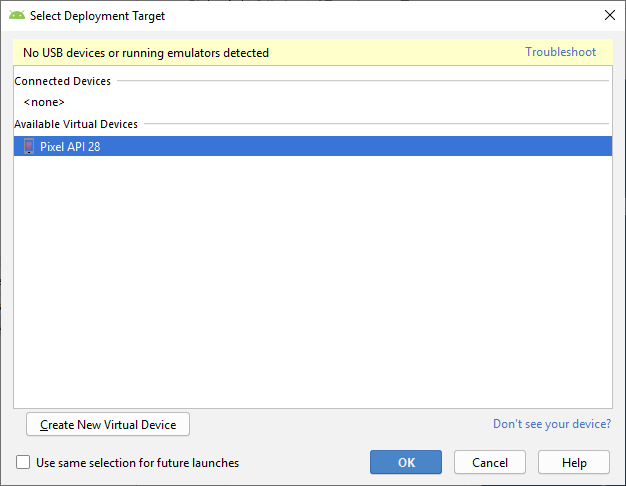


On the next page hit **Finish** and you should see your new device now under your list of devices.

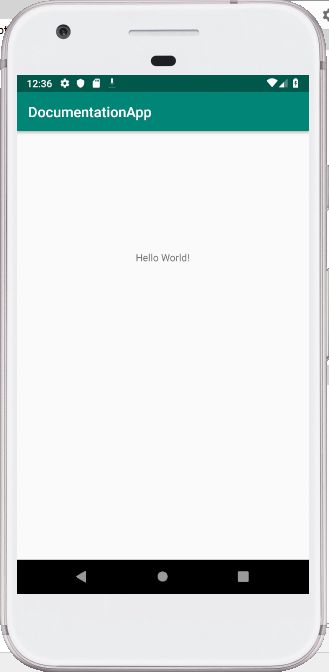


You can now close the AVD Manager window and go back to your Android Studio project. In order to emulate your app, hit the play button in the top right portion of the window and select a device from either your connected devices or one of the virtual devices which you have set up.





Once you have selected a device hit, **OK**. Depending on your computer, it will take some time for the emulator to load so just be patient with it. Additionally, if your app crashes the first time you launch it, just try launching it again and it should work.



**Challenge 2: Opening The Camera**

**Objective:** Learn how to access the Android Studio built-in camera API and be able to take photos using your app.

**Create a button to open camera:**

In order to open the camera, the first thing which needs to be done is to create a way for users to do so. In this tutorial, we will be using a button to open the camera, however this can be done in a number of ways depending on the app you are trying to create. In order to create a button, go to the **Palette** in the **Design** view for your **activity\_main.xml** and drag a button of your choice onto the app design. You can then click on the button and in the **Attributes** window edit the text of the button to have it relate to the action you are trying to do. In this case, since we are trying to open the camera, the button will be labeled “take picture”.

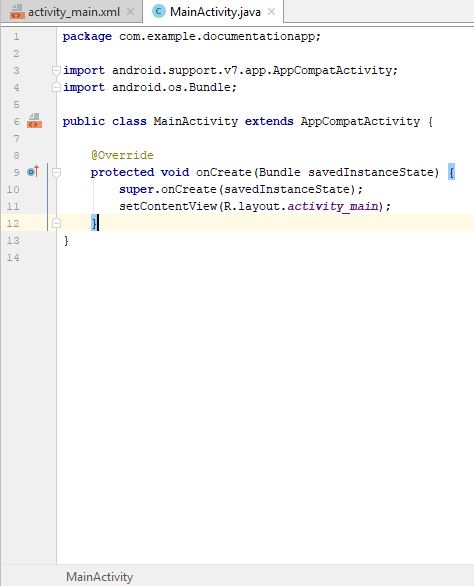
Constraints and establishing a layout margin is necessary for all buttons and Text boxes placed onto the design screen. To establish the layout margins of the button, click on the **blue plus signs** surrounding the square to the right of the design screen(as shown in the figure below). Clicking on each of the four plus signs will allow you to establish the button’s location with respect to the four sides of the design screen. Failure to establish a layout margin for each button/text-box will result in an error warning next to your button in the component tree.( Ignore the warning about the @string text after the button name is created).



Now that you have created a way for the user to open the camera, we need to program the button so it actually does what we want it to do. Click on the **MainActivity.java** tab in the toolbar towards the top of your screen to view the code which drives your app. Here is where we will be writing the function to open the camera.

**Writing your first function in Android Studio:**

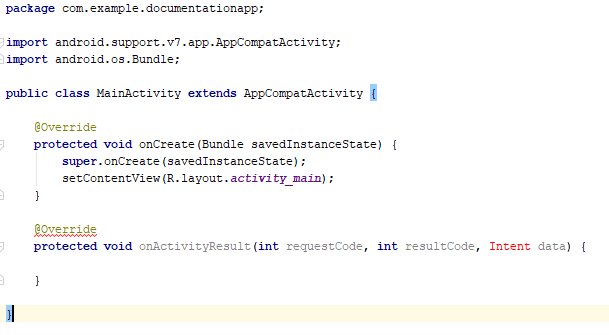
When writing the code to open the camera, we will be using many methods from the Camera API provided here <https://developer.android.com/guide/topics/media/camera>. While we will be explaining a little bit about what each method does, in order to get a better grasp and understanding of the code, feel free to reference the link above while you are following these instructions. The only piece of code which should be in your **MainActivity.java** file is the **onCreate** method. If there are any other methods, delete them before continuing. Again your code should look like this before you start writing anything.( Note: package name “documentationapp” should be replaced with your package name)



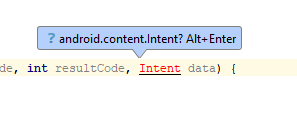
The first method that we will implement from the Camera API is the **onActivityResult** method. Essentially, this method is called after an activity/action occurs within your app in order to make something else happen. In our case, after the button is pressed, we want the camera to open. This method takes three parameters which we will now explain in order to give you a better idea of what this method does:

1. requestCode - specifies the kind of request/what the request is for
2. resultCode - identifies whether or not the activity was performed successfully
3. resultIntent - what you want to happen now that the activity has occurred

Now that we have explained the method, create an empty **onActivityResult** method with just the method header so your code looks like this.



Notice that the word Intent is in red. This is because we have not imported the Intent class yet and cannot make use of any method/objects from that class. In order to import the Intent class, hover over the keyword and hit **‘Alt+Enter’**. A general rule of thumb is that if you hover over the keyword in red and a prompt like the one below shows up allowing you to import the class, you should do so.



After importing the class you should see that an import statement has been added to the top of your code and it should now look like this.



The first thing which needs to be done in order to write the method is to write a series of if statements. The purpose of these if statements is to make sure that the method is being accessed from the right activity. There may be many places in your code where onActivityResult is called and we want to know through which activity this method is being called, if the activity executed successfully, and what the method is supposed to do now after the activity has occurred. Copy and paste the following if statements into your code:

**if**(resultCode == ***RESULT\_OK***) { *//checks whether or not the activity executed successfully*

**if**(requestCode == ***CAPTURE\_IMAGE\_REQUEST***) { *//if the request is to open the camera*

}

}

**else** {

*//displays message on the screen if the activity did not execute successfully*

Toast.*makeText*(getBaseContext(), **"error has occurred"**, Toast.***LENGTH\_LONG***).show();

}

Within the else brackets, this statement makes use of the Toast class which causes temporary messages to appear on the screen. Throughout the rest of the documentation, we will be making use of this class in order to debug and look for errors. Once this is done, declare a final static int named CAPTURE\_IMAGE\_REQUEST at the beginning of your code right before your onCreate method. The statement should look like this:

**static final int *CAPTURE\_IMAGE\_REQUEST*** = 1;

This constant will be used to specify that the camera is supposed to be opened. Afterwards, copy and paste this code inside the if statement which checks if requestCode is equal to CAPTURE\_IMAGE\_REQUEST:

*//to convert File object to Bitmap object*

**bitmap** = BitmapFactory.*decodeFile*(**photoFile**.getAbsolutePath());

**imageView**.setImageBitmap(**bitmap**);

*//creates an image which can be displayed*

MediaStore.Images.Media.*insertImage*(getContentResolver(), **bitmap**, **"title"** , **"description"**);

Make sure to resolve all import errors and then declare

File **photoFile** = **null**;

at the beginning of your code. Make sure the File class is imported.

While we won’t go into too much depth about what the two statements above do, just know that these statements basically convert a File object to a Bitmap object which can then be turned into an image that is able to be saved on an Android phone. If you are interested in the exact details of what this method does, then please feel free to look at the Android Studio API which has documentation on all of these classes and methods.

Now you must test the code that you have just written. Add these lines of code to your **onCreate** method in order to add a listener to the button so it calls the onActivityResult method when clicked. Remember to resolve any import errors.

*//represents button object in design*

Button button = findViewById(R.id.***button***);

*//adds listener for camera button*

button.setOnClickListener(**new** View.OnClickListener() {

@Override

**public void** onClick(View view) {

*//creates the intent to access camera*

Intent takePictureIntent = **new** Intent(MediaStore.***ACTION\_IMAGE\_CAPTURE***);

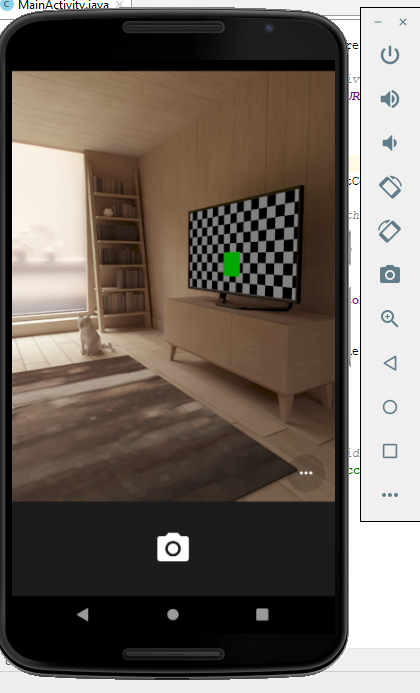
*//starts activity which implicitly calls onActivityResult*

startActivityForResult(takePictureIntent, ***CAPTURE\_IMAGE\_REQUEST***);

}

});

Hit the play button in the top right corner and select the device which you want to run your app from. When the app opens, click on the button and hit allow for any prompts which come up. The camera should then open and show you a screen like this if you are using the emulator.



Within this view you can walk around the room by holding Alt and using WASD as well as rotate your field of view by holding Alt and using your mouse. In order to take a picture you can click the white camera at the bottom of the screen and in order to exit without taking a picture you can hit the triangle facing left at the bottom of the screen. At this point in the documentation, you have successfully opened the camera and have completed the second challenge! Just know that if you do try and save the photo you take, the app will crash as we have not yet written the code to properly save photos which have been taken. Choosing not to save the photo will bring you back to the original home screen and a Toast message should pop up saying an error has occurred.

**Challenge 3: Saving Picture to Gallery**

**Objective:** Learn how to interact with other apps from your app in Android Studio and how to save data which lasts outside of your app’s runtime

**Writing a simple debug method:**

Before we get into saving the picture from your camera to the gallery on your phone, we want to write a simple debug method which should help you out if you run into any issues. The point of this method is to display a message in text on the screen whenever an event happens/doesn’t happen in order to help you identify whether or not your app is working correctly. In this tutorial, the method is called displayMessage, however you can title it as you want as long as you remember to use your own method name instead of ours in all occurrences which follow for the rest of the tutorial.

*//method to display messages using toast*

**private void** displayMessage(Context context, String message)

{

Toast.*makeText*(context,message,Toast.***LENGTH\_LONG***).show();

}

This method makes use of the Toast class which essentially revolves around displaying temporary messages on the screen. The two options you can select for how long the message is displayed are LENGTH\_LONG and LENGTH\_SHORT. Usually, it is easier to see the message when it lasts on the screen for a longer time so we will be using LENGTH\_LONG for debugging purposes.

**Writing a method to create an image file:**

At this point, you should be able to open the camera to take a picture, but not save a picture. Therefore, the first function we will write in Challenge #3 will allow us to save that picture. In order to save a picture, you must create an image file for the picture to be stored in. The method to do that is written as follows:

*//method which creates the image file*

**private** File createImageFile() **throws** IOException {

*// Create an image file name*

String timeStamp = **new** SimpleDateFormat(**"yyyyMMdd\_HHmmss"**).format(**new** Date());

String imageFileName = **"JPEG\_"** + timeStamp + **"\_"**;

File storageDir = getExternalFilesDir(Environment.*DIRECTORY\_PICTURES*);

File image = File.*createTempFile*( imageFileName,**".jpg"**,storageDir);

*// Save a file: path for use with ACTION\_VIEW intents*

**mCurrentPhotoPath** = image.getAbsolutePath();

**return** image;

}

Remember to import all necessary classes and then declare mCurrentPhotoPath at the beginning of your code like this:

String **mCurrentPhotoPath**;

mCurrentPhotoPath does not need to be initialized as the only time we will be using it is within the createImageFile method where we are guaranteed to initialize it. This method is pretty straightforward in terms of code, we are basically creating a file name for the image which consists of the date and then we are specifying that the location of the file is the external storage directory in the DIRECTORY\_PICTURES, which is essentially where all photos are saved on an Android phone. Additionally, the reason why we have a return statement at the end of the file is because we will be using this file we have created in another method.

**Checking permissions:**

The next method we will write involves checking the permissions on the phone. In order to open the camera and save to the gallery, the user of the phone must allow the app to access these functions. The method below checks to make sure the appropriate permissions are granted and then calls the captureImage() method which we will write next.

*//checks to see if permissions are granted*

@Override

**public void** onRequestPermissionsResult(**int** requestCode, String[] permissions, **int**[] grantResults) {

**if** (requestCode == 0) {

**if** (grantResults.**length** > 0 && grantResults[0] == PackageManager.***PERMISSION\_GRANTED***

&& grantResults[1] == PackageManager.***PERMISSION\_GRANTED***) {

captureImage();

}

}

}

**Capturing the Image:**

First create a method header named **public void** captureImage() . Then, create an if statement which checks if the permissions are granted by calling the method we wrote:

**if** (ContextCompat.*checkSelfPermission*(**this**, Manifest.permission.***CAMERA***) != PackageManager.***PERMISSION\_GRANTED***) {

ActivityCompat.*requestPermissions*(**this**, **new** String[] { Manifest.permission.***CAMERA***, Manifest.permission.***WRITE\_EXTERNAL\_STORAGE*** }, 0);

}

**else**

{

//to be written later

}

The main portion of the code will be written in the else statement. This is because there is not much for the app to do if the permissions have not been granted. Next, copy and paste this piece of code into your else statement.

*//creates the intent to access camera*

Intent takePictureIntent = **new** Intent(MediaStore.***ACTION\_IMAGE\_CAPTURE***);

**if** (takePictureIntent.resolveActivity(getPackageManager()) != **null**) {

*// Create the File where the photo should go*

**try** {

**photoFile** = createImageFile();

displayMessage(getBaseContext(),**photoFile**.getAbsolutePath());

Log.*println*(Log.***ASSERT***,**"photoFile path: "**,**photoFile**.getAbsolutePath());

*// Continue only if the File was successfully created*

**if** (**photoFile** != **null**) {

Uri photoURI = FileProvider.*getUriForFile*(**this**,

**"com.example.savepictogallery.fileprovider"**,

**photoFile**);

takePictureIntent.putExtra(MediaStore.***EXTRA\_OUTPUT***, photoURI);

startActivityForResult(takePictureIntent, ***CAPTURE\_IMAGE\_REQUEST***);

}

}

**catch** (Exception ex) {

*// Error occurred while creating the File*

displayMessage(getBaseContext(),ex.getMessage().toString());

}

What this piece of code does is basically invoke a try and catch. The point of the try and catch is to attempt to create the image file while accounting for possible errors which may come up. The try statement attempts to create the file and if an error occurs, instead of crashing the program, the program jumps to the catch statement which handles the error. In this case our most likely error when creating the image file is that we have not specified a proper directory for it and it cannot be properly instantiated. If any error occurs, the catch statement is responsible for printing out the error message so you know what went wrong. Additionally, there is another if statement which checks whether or not the photoFile is null. If the file is not null, then the function will make a call to the startActivityForResult method which will process the request to take a picture and save it. After your else statement paste this else statement which tells us whether or not an error occurred:

*//displays error message*

**else** {

displayMessage(getBaseContext(),**"Error occurred"**);

}

Your code should look like this now:

*//method to invoke the camera*

**public void** captureImage()

{

*//makes sure permissions are granted*

**if** (ContextCompat.*checkSelfPermission*(**this**, Manifest.permission.***CAMERA***) != PackageManager.***PERMISSION\_GRANTED***) {

ActivityCompat.*requestPermissions*(**this**, **new** String[] { Manifest.permission.***CAMERA***, Manifest.permission.***WRITE\_EXTERNAL\_STORAGE*** }, 0);

}

**else**

{

*//creates the intent to access camera*

Intent takePictureIntent = **new** Intent(MediaStore.***ACTION\_IMAGE\_CAPTURE***);

**if** (takePictureIntent.resolveActivity(getPackageManager()) != **null**) {

*// Create the File where the photo should go*

**try** {

**photoFile** = createImageFile();

displayMessage(getBaseContext(),**photoFile**.getAbsolutePath());

Log.*println*(Log.***ASSERT***,**"photoFile path: "**,**photoFile**.getAbsolutePath());

*// Continue only if the File was successfully created*

**if** (**photoFile** != **null**) {

Uri photoURI = FileProvider.*getUriForFile*(**this**,

**"com.example.documentationapp.fileprovider"**,

**photoFile**);

takePictureIntent.putExtra(MediaStore.***EXTRA\_OUTPUT***, photoURI);

startActivityForResult(takePictureIntent, ***CAPTURE\_IMAGE\_REQUEST***);

}

}

**catch** (Exception ex) {

*// Error occurred while creating the File*

displayMessage(getBaseContext(),ex.getMessage().toString());

}

}

*//displays error message*

**else** {

displayMessage(getBaseContext(),**"Error occurred"**);

}

}

}

Now that you have completed the captureImage method, go back to the onCreate method and delete these two lines:

*//creates the intent to access camera*

Intent takePictureIntent = new Intent(MediaStore.*ACTION\_IMAGE\_CAPTURE*);

*//starts activity which implicitly calls onActivityResult*

startActivityForResult(takePictureIntent, *CAPTURE\_IMAGE\_REQUEST*);

Instead, replace them with a call to capture image. Previously, we were informally opening the camera which did not allow us to actually save the image. The captureImage() method allows the user to actually save the image to their device instead of just taking a picture.

Now that the code is taken care of, we will be making a few changes to some manifest files and creating a file of our own.

The first step is to change your Android\_Manifest.xml file by copying and pasting this code to replace your current file:

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

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

**package="com.example.documentationapp"**>

<**uses-feature**

**android:name="android.hardware.camera"**

**android:required="true"** />

<**uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"** />

<**uses-permission android:name="android.permission.CAMERA"** />

<**uses-feature android:name="android.hardware.camera.autofocus"** />

<**application**

**android:allowBackup="true"**

**android:icon="@mipmap/ic\_launcher"**

**android:label="@string/app\_name"**

**android:roundIcon="@mipmap/ic\_launcher\_round"**

**android:supportsRtl="true"**

**android:theme="@style/AppTheme"**>

<**activity android:name=".MainActivity"**>

<**intent-filter**>

<**action android:name="android.intent.action.MAIN"** />

<**category android:name="android.intent.category.LAUNCHER"** />

</**intent-filter**>

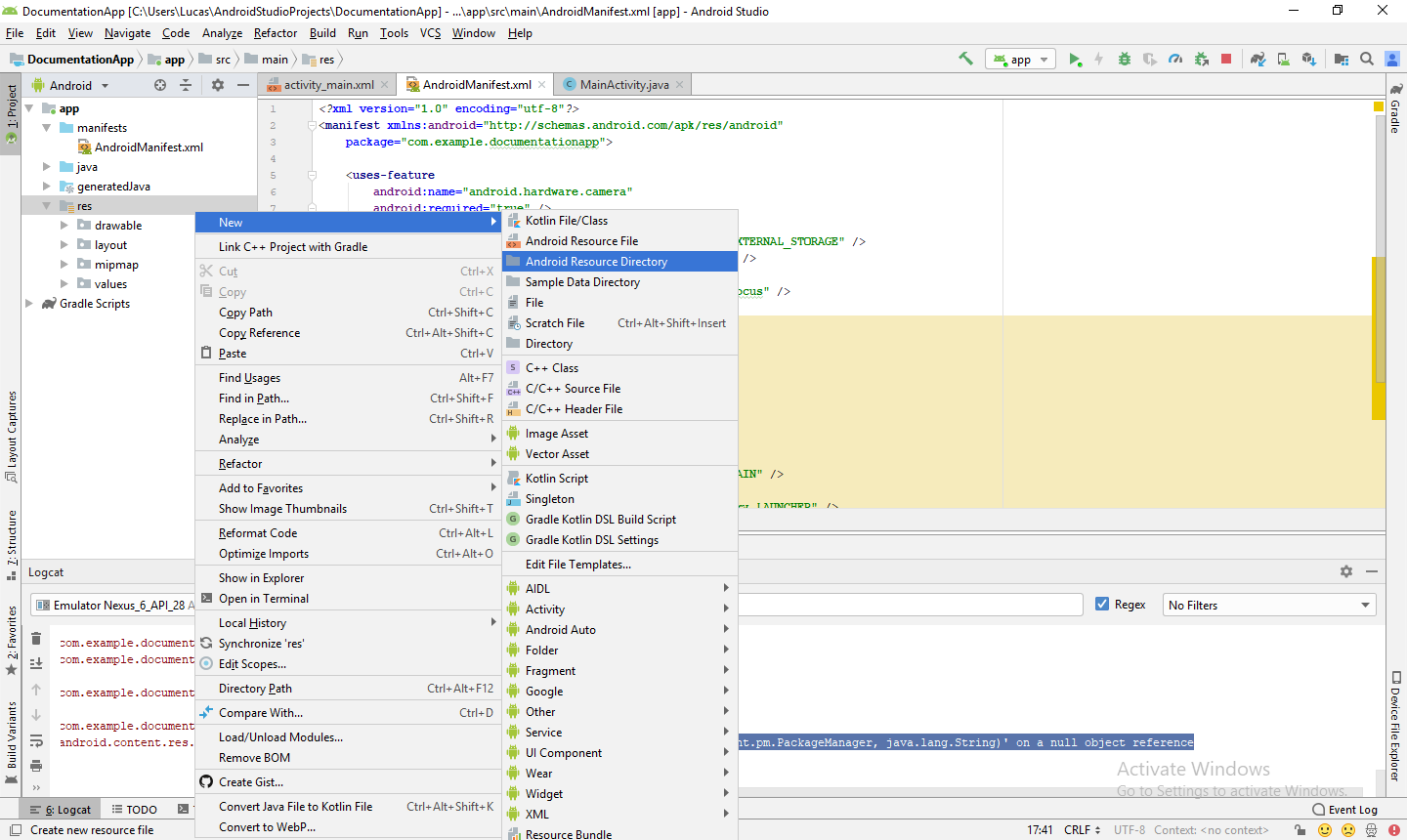
</**activity**>

</**application**>

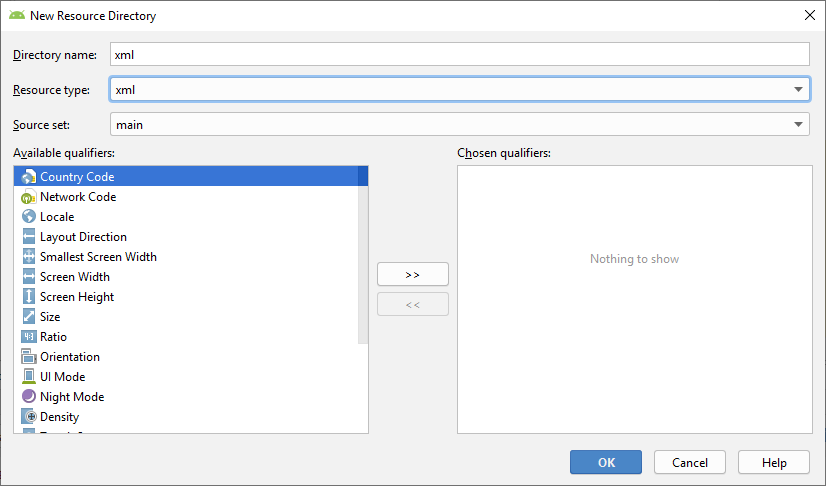
</**manifest**>

**IMPORTANT:** Remember to change the package name at the top of your file to match your own package name! Unless your project is also named DocumentationApp, then you should change it! The package name is highlighted above.

Once you are finished with the manifest file, navigate to your **res** directory under your project directory/tree in the upper left of your screen. Then right-click **res** and select **New**, then select **Android Resource Directory**:



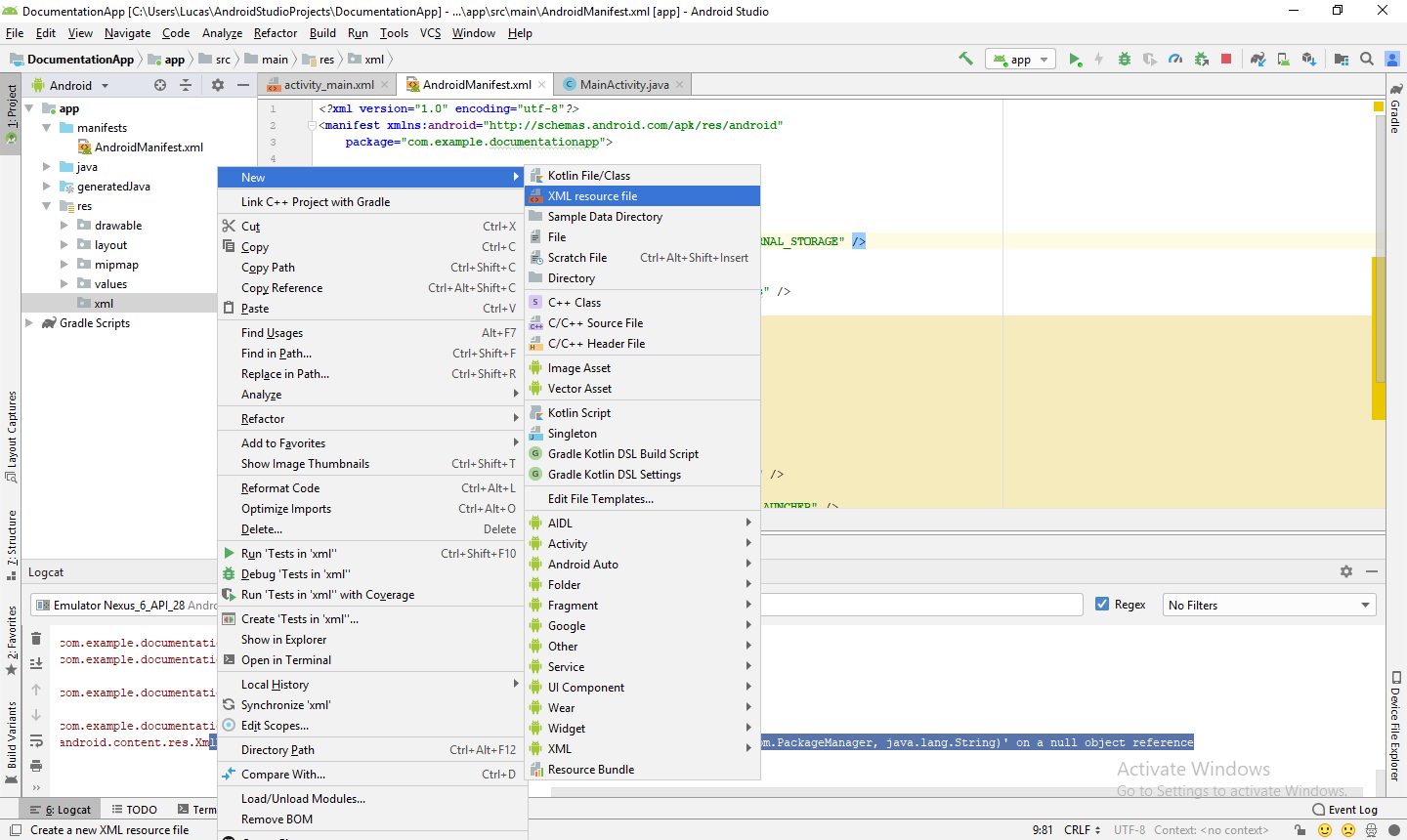
That will bring you to this window:



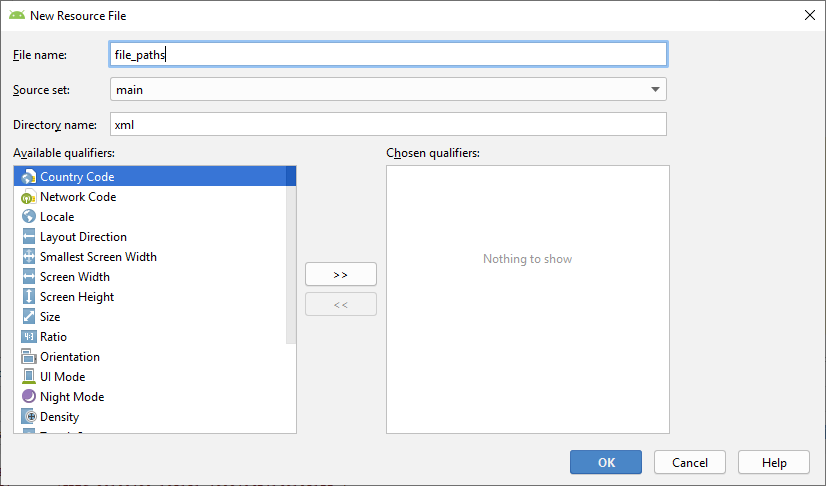
Make the directory name **xml** and the resource type also **xml**, then hit **OK**. Once that is done you should now have a new folder in your project directory entitled xml.



After your xml folder is created, **right-click on it** and then select **New**, then select **XML resource file**:



The name of the file should be **file\_paths**:



Hit **OK** and then switch to the **Text** instead of the Design view for the file\_paths.xml file. Replace the existing code in the file with this:

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

<**paths xmlns:android="http://schemas.android.com/apk/res/android"**>

<**external-path name="my\_images" path="Android/data/com.example.savepictogallery/files/Pictures"** />

</**paths**>

Additionally, go to your Android\_Manifest.xml file and copy and paste this in:

<**provider**

**android:name="android.support.v4.content.FileProvider"**

**android:authorities="com.example.documentationapp.fileprovider"**

**android:exported="false"**

**android:grantUriPermissions="true"**>

<**meta-data**

**android:name="android.support.FILE\_PROVIDER\_PATHS"**

**android:resource="@xml/file\_paths"** />

</**provider**>

Again, remember to switch the package name to what you titled your project/package.

Now you should be able to correctly save a photo.

**Choosing Photo from Gallery**

To allow the user to also have the option of choosing an image from the gallery, first declare a final static int named GALLERY\_REQUEST at the beginning of your code right before your onCreate method, under CAPTURE\_IMAGE\_REQUEST. The statement should look like this:

**static final int *GALLERY\_REQUEST*** = 0;

Inside the **onActivityResult** function, add this if statement under the if statement for the camera image request.

**if**(requestCode == ***GALLERY\_REQUEST***) {

Uri uri = data.getData();

**try** {

**bitmap** = MediaStore.Images.Media.*getBitmap*(getContentResolver(), uri);

imageView.setImageBitmap(**bitmap**);

displayMessage(getBaseContext(), **"successful"**);

}

**catch**(Exception e) {

e.printStackTrace();

}

}

We will also need a function to open up the gallery within the **onCreate** function.

//method which opens gallery, gallery button

**public void** openGallery(View view) {

Intent intent = **new** Intent();

intent.setType(**"image/\*"**);

intent.setAction(Intent.***ACTION\_GET\_CONTENT***);

startActivityForResult(Intent.*createChooser*(intent, **"Select Picture"**), ***GALLERY\_REQUEST***);

}

**CHALLENGE #4: Deploying Model**

**Objective:**

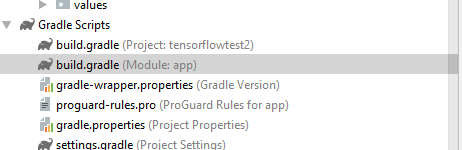
Integrate TensorFlow Mobile model within Android Studio to allow your app to make predictions based on a model.

In this tutorial we will assume that you already have a TensorFlow Mobile model set up and we will be responsible for teaching you how to integrate it with Android Studio. The following article is a link to a more general TensorFlow Mobile tutorial if you do not have it already set up on your computer:

<https://heartbeat.fritz.ai/deploying-pytorch-and-keras-models-to-android-with-tensorflow-mobile-a16a1fb83f2>

Please feel free to reference this article should our tutorial not provide enough details or if you are more curious about TensorFlow.

The first file we will be editing in this challenge is the app level **build.gradle** file. Here is a picture of the project hierarchy so you can see which **build.gradle** file we are talking about:



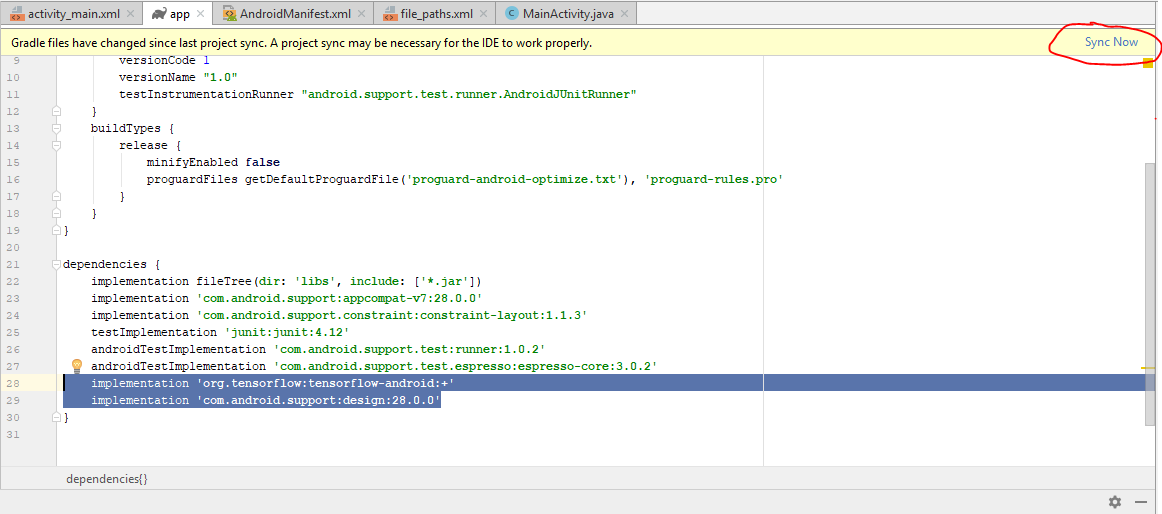
**NOTE:** There are two build.gradle files. One is your project level build.gradle file and the other is the app level build.gradle file. Make sure you are editing the app level build.gradle file! It should say “(Module:app)” next to it.

Add the following two lines at the end of your dependencies block in your build.gradle file:

implementation **'org.tensorflow:tensorflow-android:+'**

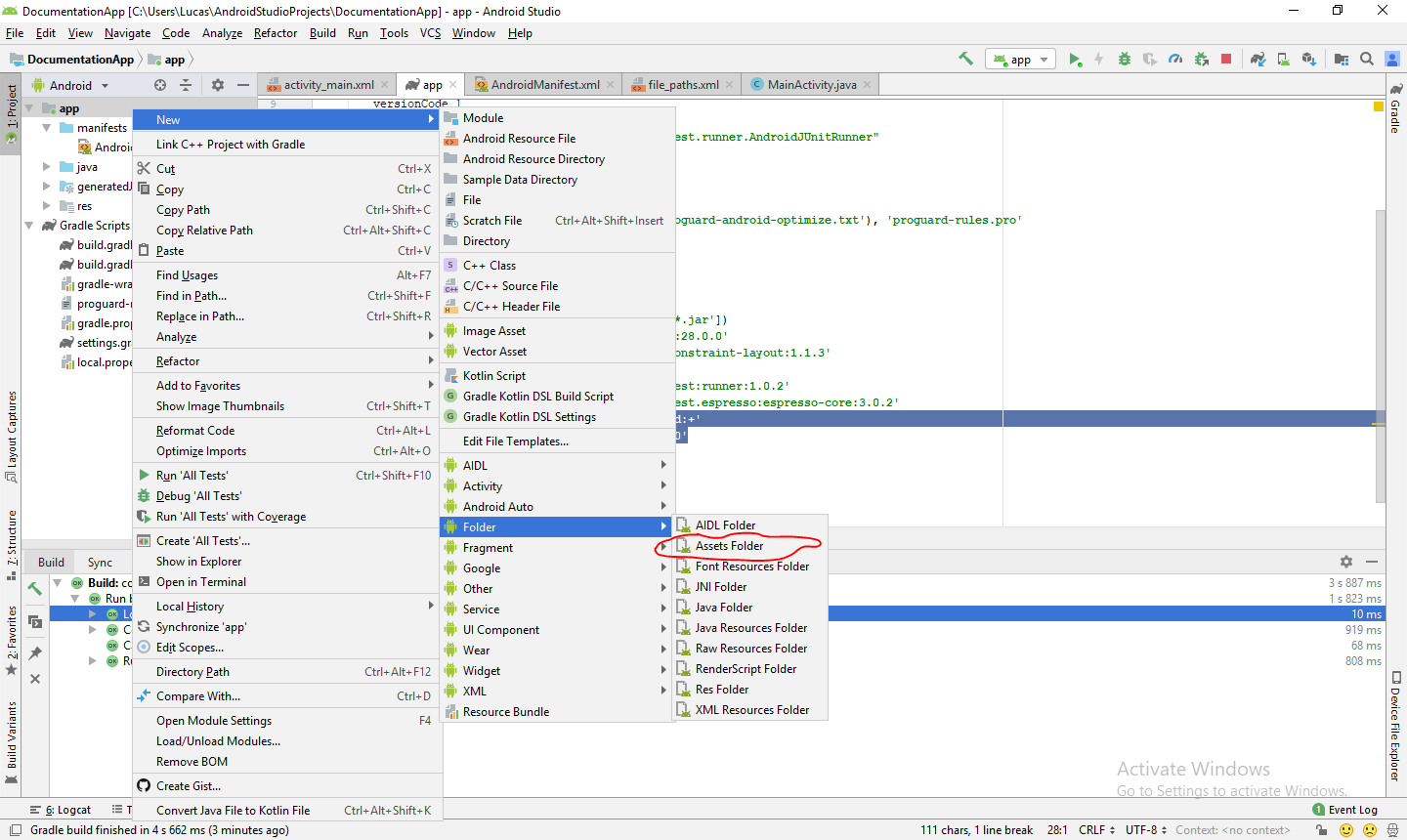
implementation **'com.android.support:design:28.0.0'**

Your file should now look like this:

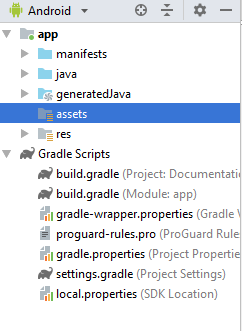


Once you are done, click on **Sync Now** in order to sync your gradle files. (Upper-right hand corner)

At this point we will need to create another folder in our project similar to the one we created for our file\_paths.xml file. Right click on **app** then go to **New->Folder->Assets Folder**.

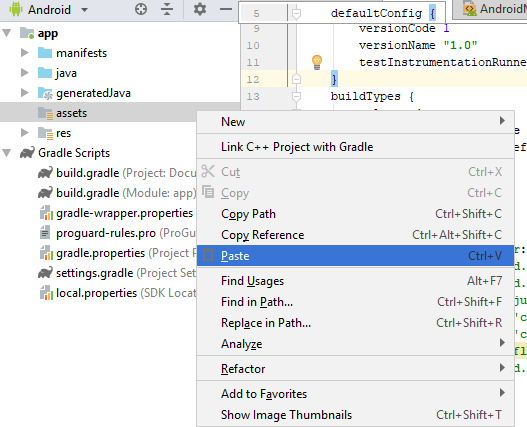


If a prompt comes up asking about a root source directory or something similar, just hit **Finish** and the new **assets** folder should appear in your project.

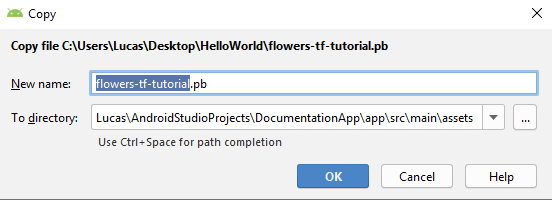


The next thing you should do is locate your model file which should end in .pb and then copy and paste it into the assets folder.

**NOTE:** There have been problems right-clicking assets then adding a new file so please right click your .pb file and then hit copy and then right click assets and hit paste.



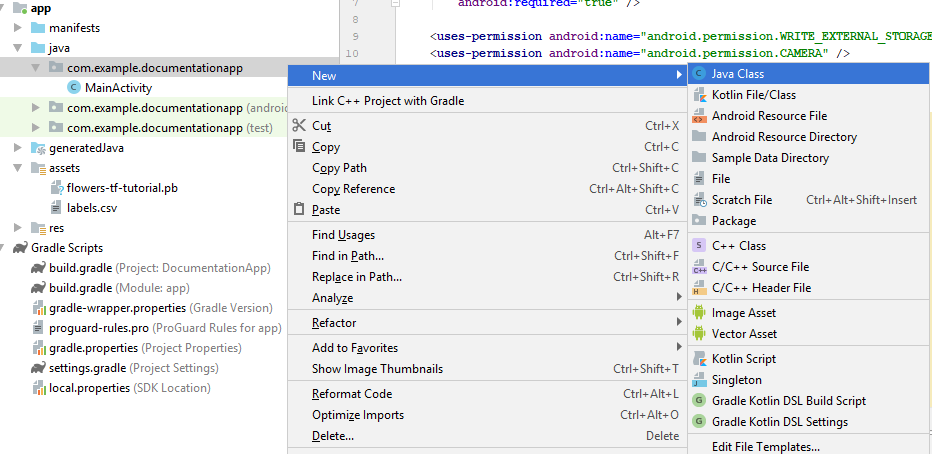
After clicking paste you should get a prompt which looks like this:

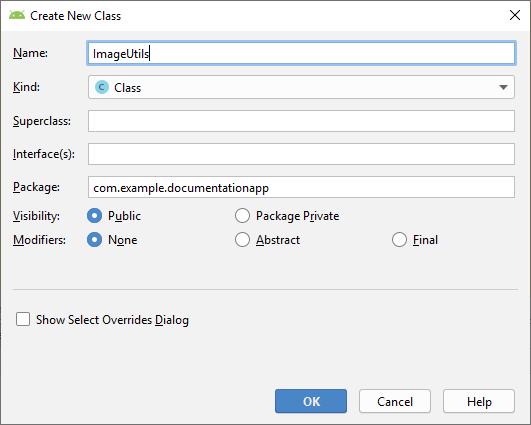


Hit **OK** and then you should see your model file within the **assets** folder.

The next thing you need to do is add your .csv or similar file (.json or anything like those also work) into your **assets** folder if you need this to run your model. Our model, which is named **flowers-tf-tutorial.pb** comes with a **labels.csv** file so we will be adding that to the folder. Repeat the same process of copying and pasting that file into the folder. At this point you have imported all the necessary assets and we can begin to write some code which will help you analyze images using the provided model.

The first thing to do is create a new class called ImageUtils. Right click on the java folder under app in the project directory portion of your screen. Go to **New->Java Class** and call it **ImageUtils**.





Copy and paste this into your file:

**package** com.example.documentationapp;

**import** android.content .res.AssetManager;

**import** android.graphics.Bitmap;

**import** android.graphics.Canvas;

**import** android.graphics.Matrix;

**import** android.os.Environment;

**import** java.io.File;

**import** java.io.FileOutputStream;

**import** java.io.InputStream;

**import** org.json.\*;

*/\*\**

*\* Utility class for manipulating images.*

*\*\*/*

**public class** ImageUtils {

*/\*\**

*\* Returns a transformation matrix from one reference frame into another.*

*\* Handles cropping (if maintaining aspect ratio is desired) and rotation.*

*\**

*\** ***@param srcWidth*** *Width of source frame.*

*\** ***@param srcHeight*** *Height of source frame.*

*\** ***@param dstWidth*** *Width of destination frame.*

*\** ***@param dstHeight*** *Height of destination frame.*

*\** ***@param applyRotation*** *Amount of rotation to apply from one frame to another.*

*\* Must be a multiple of 90.*

*\** ***@param maintainAspectRatio*** *If true, will ensure that scaling in x and y remains constant,*

*\* cropping the image if necessary.*

*\** ***@return*** *The transformation fulfilling the desired requirements.*

*\*/*

**public static** Matrix getTransformationMatrix(

**final int** srcWidth,

**final int** srcHeight,

**final int** dstWidth,

**final int** dstHeight,

**final int** applyRotation,

**final boolean** maintainAspectRatio) {

**final** Matrix matrix = **new** Matrix();

**if** (applyRotation != 0) {

*// Translate so center of image is at origin.*

matrix.postTranslate(-srcWidth / 2.0f, -srcHeight / 2.0f);

*// Rotate around origin.*

matrix.postRotate(applyRotation);

}

*// Account for the already applied rotation, if any, and then determine how*

*// much scaling is needed for each axis.*

**final boolean** transpose = (Math.*abs*(applyRotation) + 90) % 180 == 0;

**final int** inWidth = transpose ? srcHeight : srcWidth;

**final int** inHeight = transpose ? srcWidth : srcHeight;

*// Apply scaling if necessary.*

**if** (inWidth != dstWidth || inHeight != dstHeight) {

**final float** scaleFactorX = dstWidth / (**float**) inWidth;

**final float** scaleFactorY = dstHeight / (**float**) inHeight;

**if** (maintainAspectRatio) {

*// Scale by minimum factor so that dst is filled completely while*

*// maintaining the aspect ratio. Some image may fall off the edge.*

**final float** scaleFactor = Math.*max*(scaleFactorX, scaleFactorY);

matrix.postScale(scaleFactor, scaleFactor);

} **else** {

*// Scale exactly to fill dst from src.*

matrix.postScale(scaleFactorX, scaleFactorY);

}

}

**if** (applyRotation != 0) {

*// Translate back from origin centered reference to destination frame.*

matrix.postTranslate(dstWidth / 2.0f, dstHeight / 2.0f);

}

**return** matrix;

}

**public static** Bitmap processBitmap(Bitmap source,**int** size){

**int** image\_height = source.getHeight();

**int** image\_width = source.getWidth();

Bitmap croppedBitmap = Bitmap.*createBitmap*(size, size, Bitmap.Config.***ARGB\_8888***);

Matrix frameToCropTransformations = *getTransformationMatrix*(image\_width,image\_height,size,size,0,**false**);

Matrix cropToFrameTransformations = **new** Matrix();

frameToCropTransformations.invert(cropToFrameTransformations);

**final** Canvas canvas = **new** Canvas(croppedBitmap);

canvas.drawBitmap(source, frameToCropTransformations, **null**);

**return** croppedBitmap;

}

**public static float**[] normalizeBitmap(Bitmap source,**int** size,**float** mean,**float** std){

**float**[] output = **new float**[size \* size \* 3];

**int**[] intValues = **new int**[source.getHeight() \* source.getWidth()];

source.getPixels(intValues, 0, source.getWidth(), 0, 0, source.getWidth(), source.getHeight());

**for** (**int** i = 0; i < intValues.**length**; ++i) {

**final int** val = intValues[i];

output[i \* 3] = (((val >> 16) & 0xFF) - mean)/std;

output[i \* 3 + 1] = (((val >> 8) & 0xFF) - mean)/std;

output[i \* 3 + 2] = ((val & 0xFF) - mean)/std;

}

**return** output;

}

**public static** Object[] argmax(**float**[] array){

**int** best = -1;

**float** best\_confidence = 0.0f;

**for**(**int** i = 0;i < array.**length**;i++){

**float** value = array[i];

**if** (value > best\_confidence){

best\_confidence = value;

best = i;

}

}

**return new** Object[]{best,best\_confidence};

}

**public static** String getLabel( InputStream jsonStream,**int** index){

String label = **""**;

**try** {

**byte**[] jsonData = **new byte**[jsonStream.available()];

jsonStream.read(jsonData);

jsonStream.close();

String jsonString = **new** String(jsonData,**"utf-8"**);

JSONObject object = **new** JSONObject(jsonString);

label = object.getString(String.*valueOf*(index));

}

**catch** (Exception e){

}

**return** label;

}

}

In your MainActivity file put the following in before your onCreate method

*//Load the tensorflow inference library*

**static** {

System.*loadLibrary*(**"tensorflow\_inference"**);

}

*//PATH TO OUR MODEL FILE AND NAMES OF THE INPUT AND OUTPUT NODES*

**private** String **MODEL\_PATH** = **"file:///android\_asset/my\_tf\_model.pb"**;

**private** String **INPUT\_NAME** = **"input\_1"**;

**private** String **OUTPUT\_NAME** = **"dense\_2/Softmax"**;

**private** TensorFlowInferenceInterface **tf**;

**private** Bitmap **bitmap**;

*//ARRAY TO HOLD THE PREDICTIONS AND FLOAT VALUES TO HOLD THE IMAGE DATA*

**float**[] **PREDICTIONS** = **new float**[1000];

**private float**[] **floatValues**;

**private int**[] **INPUT\_SIZE** = {224,224,3};

ImageView **imageView**;

TextView **resultView**;

Snackbar **progressBar**;

NOTE: The MODEL\_PATH variable will be different for every person. Simply replace the my\_tf\_model.pb with your own file name.

Put the following in your onCreate method:

*//initialize tensorflow with the AssetManager and the Model*

**tf** = **new** TensorFlowInferenceInterface(getAssets(),**MODEL\_PATH**);

**imageView** = (ImageView) findViewById(R.id.***imageView***);

**resultView** = (TextView) findViewById(R.id.***results***);

**progressBar** = Snackbar.*make*(**imageView**,**"PROCESSING IMAGE"**,Snackbar.***LENGTH\_INDEFINITE***);

**final** FloatingActionButton predict = (FloatingActionButton) findViewById(R.id.***predict***);

predict.setOnClickListener(**new** View.OnClickListener() {

@Override

**public void** onClick(View view) {

**try**{

**progressBar**.show();

predict(bitmap);

}

**catch** (Exception e){

Log.*println*(Log.***ASSERT***, **"image test"**, **"not work"**);

}

}

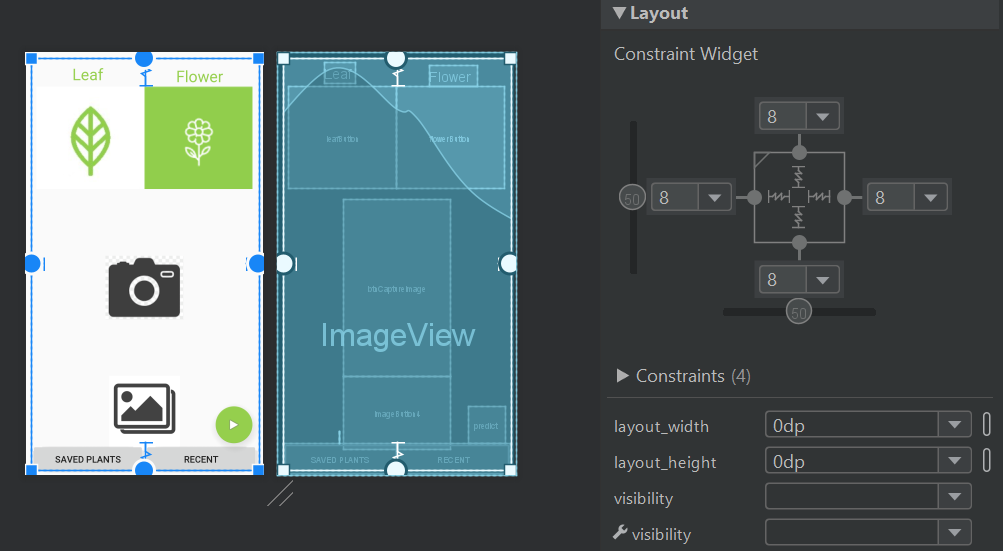
});

There will be a few errors in your code after doing this. In order to resolve these, create an ImageView, TextView and FloatingActionButton in the GUI portion of your project, activity\_main.xml, and title them “imageView”, “results”, and “predict”. You can change these names if you want, but you will have to change them within the code as well.

**ImageView “imageView”**

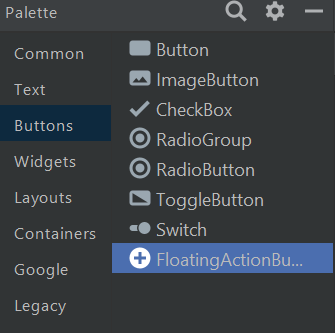
This will display your image after taking a picture or choosing it from the gallery.

Use the Constraint Widget to adjust where you want the image to appear.



Leave the actual image blank so the user can decide the image.

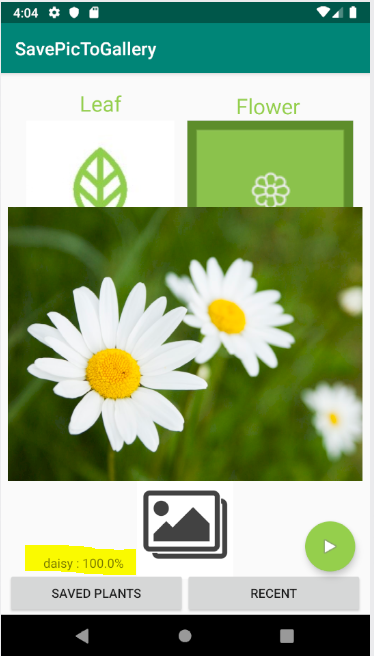
**FloatingActionButton “predict”**



This will be a button located in the bottom right-hand corner of your screen that will classify your image and display it through TextView “results”.

You can put any image or text you want inside to make it user-friendly. 

**TextView “results”**



This shows the user the name of the flower and the confidence percentage.

Leave the actual text blank so the predict function can fill it in.

After resolving these errors you should be able to make a prediction based on your model.