



## **Android Application Development Instructions**

Created by Bryan Van Draanen

With help from Kevin Z.

UW CSE 331 – Software Design and Implementation

Summer 2017

## Table of Contents

<a href="#">1. Downloading and Installing Android Studio</a> .....	3
<a href="#">2. Setting up Android Studio Environment – Initial Application Setup</a> .....	5
<a href="#">3. Shipping and Importing Projects into Android Studio</a> .....	9
<a href="#">4. Installing an Android Emulator to run the Application</a> .....	12
<a href="#">5. Adding a Button to your Application</a> .....	16
<a href="#">6. Adding an ImageView and making it 'Drawable' in your Application</a> .....	24
<a href="#">7. Loading and Storing Data in a ListView</a> .....	38
<a href="#">8. Conclusion and Helpful Hints</a> .....	46

# 1. Downloading and Installing Android Studio

Android Studio is the official IDE for Android application development. While previously an Eclipse plugin existed to create Android applications, this plugin is deprecated and no longer supported for future developments of Android since June 2015.

Navigate to <https://developer.android.com/studio/index.html> and download Android Studio for your appropriate OS. The Android SDK should be included with Android Studio. Make sure you do not choose an Android Studio installation that excludes the Android SDK.

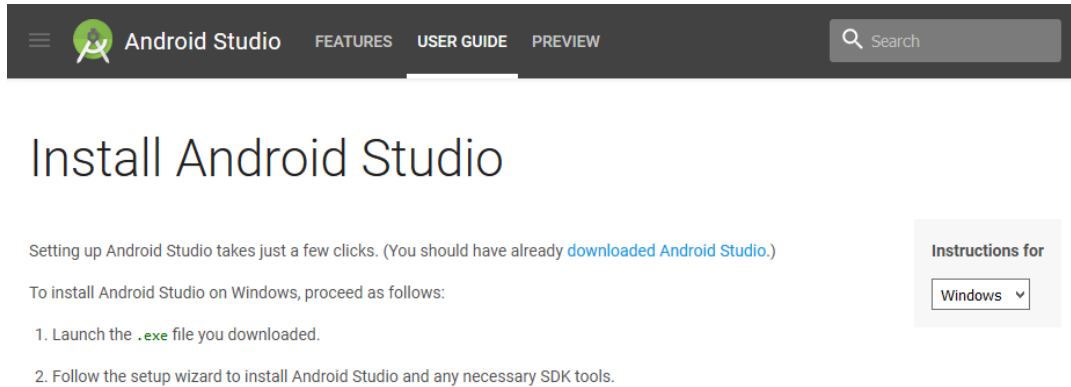
The screenshot shows the Android Studio website. On the left, there is a navigation menu with 'FEATURES', 'USER GUIDE', and 'PREVIEW'. A search bar is located in the top right. The main heading is 'Android Studio' with the subtitle 'The Official IDE for Android'. Below this, there is a description of the IDE and a 'DOWNLOAD ANDROID STUDIO' button highlighted with a red box. The button text is '2.3.3 FOR WINDOWS (1,926 MB)'. Below the button are links for 'Read the docs' and 'See the release notes'. At the bottom, there are links for 'Features', 'Latest', 'Resources', and 'Videos'. On the right side, there is a section titled 'Select a different platform' with a table of download options. The table has four columns: Platform, Android Studio package, Size, and SHA-256 checksum. The table lists options for Windows (64-bit), Windows (32-bit), Mac, and Linux. The Mac and Linux rows are highlighted with a red box.

Platform	Android Studio package	Size	SHA-256 checksum
Windows (64-bit)	<a href="#">android-studio-bundle-162.4069837-windows.exe</a> Includes Android SDK (recommended)	1,926 MB (2,020,009,280 bytes)	dc23bc968d381a5ca7fdd12bc7799b95ec0d11f1e4007387cb0de55c78b475ba
	<a href="#">android-studio-ide-162.4069837-windows.exe</a> No Android SDK	451 MB (473,299,352 bytes)	f0b72473cb94ba4bcb80eeb84f4b53364da097efa255f7cab71bcb10a28775a
	<a href="#">android-studio-ide-162.4069837-windows.zip</a> No Android SDK, no installer	468 MB (490,882,918 bytes)	b61d6f08758b5b2e6dad604d8a8d61acf549f746b07dbb0c2265daad01a7d2b7
Windows (32-bit)	<a href="#">android-studio-ide-162.4069837-windows32.zip</a> No Android SDK, no installer	467 MB (490,323,833 bytes)	db7526187d492287b6e2979249d27a67f1dd62d6e095cca7508e05edce74e272
Mac	<a href="#">android-studio-ide-162.4069837-mac.dmg</a>	463 MB (486,148,957 bytes)	da0d39221b8cb7b4b5cbe483dd4174dd1f7dc688e74f7de7c47cc8ffb6296715
Linux	<a href="#">android-studio-ide-162.4069837-linux.zip</a>	468 MB (490,782,431 bytes)	1383cfd47441e5f820b6257a1bdd683e0e980bc76c7f2027ef84dc2e6ad2f17f

Standard download option for Windows OS (above). Alternative download options for other OS (right).

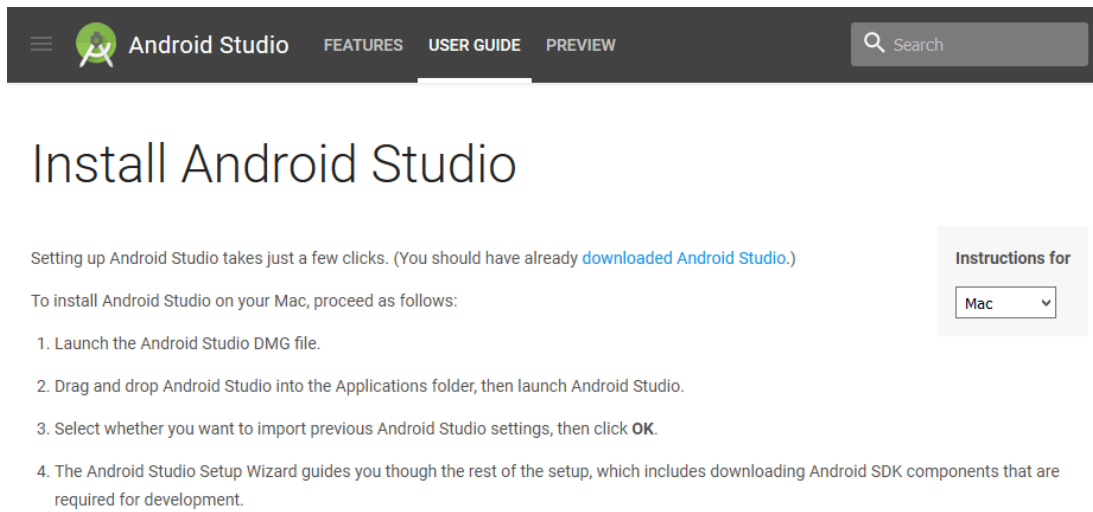
After downloading Android Studio, there are unique instructions for each individual OS on how to install the IDE (shown below). Be sure to include Android SDK in your installation.

### Windows:



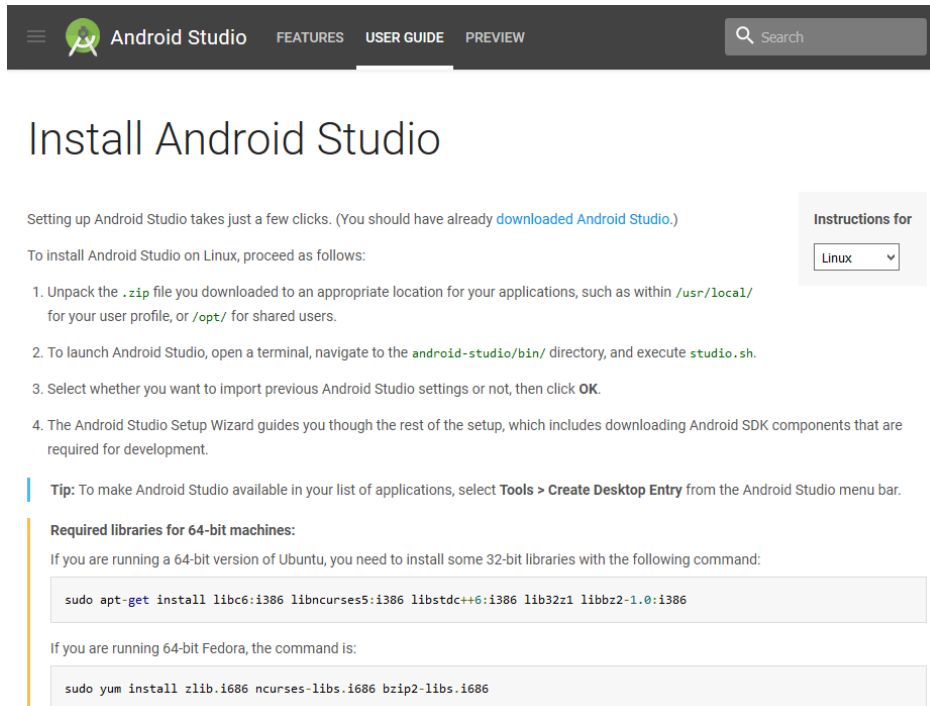
The screenshot shows the top navigation bar of the Android Studio website with the logo and links for FEATURES, USER GUIDE, and PREVIEW. A search bar is on the right. The main heading is "Install Android Studio". Below it, a text block says "Setting up Android Studio takes just a few clicks. (You should have already [downloaded Android Studio.](#))". To the right is a dropdown menu labeled "Instructions for" with "Windows" selected. The text continues: "To install Android Studio on Windows, proceed as follows:" followed by a numbered list: 1. Launch the .exe file you downloaded. 2. Follow the setup wizard to install Android Studio and any necessary SDK tools.

### Mac:



The screenshot shows the top navigation bar of the Android Studio website. The main heading is "Install Android Studio". Below it, a text block says "Setting up Android Studio takes just a few clicks. (You should have already [downloaded Android Studio.](#))". To the right is a dropdown menu labeled "Instructions for" with "Mac" selected. The text continues: "To install Android Studio on your Mac, proceed as follows:" followed by a numbered list: 1. Launch the Android Studio DMG file. 2. Drag and drop Android Studio into the Applications folder, then launch Android Studio. 3. Select whether you want to import previous Android Studio settings, then click **OK**. 4. The Android Studio Setup Wizard guides you through the rest of the setup, which includes downloading Android SDK components that are required for development.

### Linux:

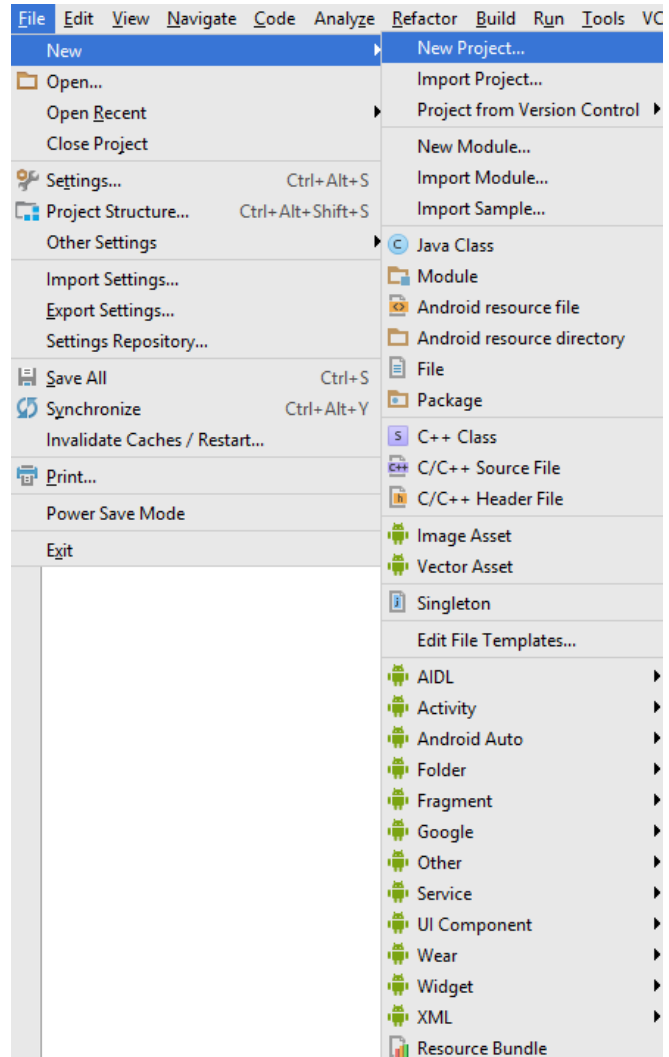


The screenshot shows the top navigation bar of the Android Studio website. The main heading is "Install Android Studio". Below it, a text block says "Setting up Android Studio takes just a few clicks. (You should have already [downloaded Android Studio.](#))". To the right is a dropdown menu labeled "Instructions for" with "Linux" selected. The text continues: "To install Android Studio on Linux, proceed as follows:" followed by a numbered list: 1. Unpack the .zip file you downloaded to an appropriate location for your applications, such as within /usr/local/ for your user profile, or /opt/ for shared users. 2. To launch Android Studio, open a terminal, navigate to the android-studio/bin/ directory, and execute studio.sh. 3. Select whether you want to import previous Android Studio settings or not, then click **OK**. 4. The Android Studio Setup Wizard guides you through the rest of the setup, which includes downloading Android SDK components that are required for development. Below the list is a tip: "Tip: To make Android Studio available in your list of applications, select **Tools > Create Desktop Entry** from the Android Studio menu bar." Then, under "Required libraries for 64-bit machines:", it says "If you are running a 64-bit version of Ubuntu, you need to install some 32-bit libraries with the following command:" followed by a code block: `sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386 lib32z1 libbz2-1.0:i386`. Then it says "If you are running 64-bit Fedora, the command is:" followed by another code block: `sudo yum install zlib.i686 ncurses-libs.i686 bzip2-libs.i686`

## 2. Setting up Android Studio Environment – Initial Application Setup

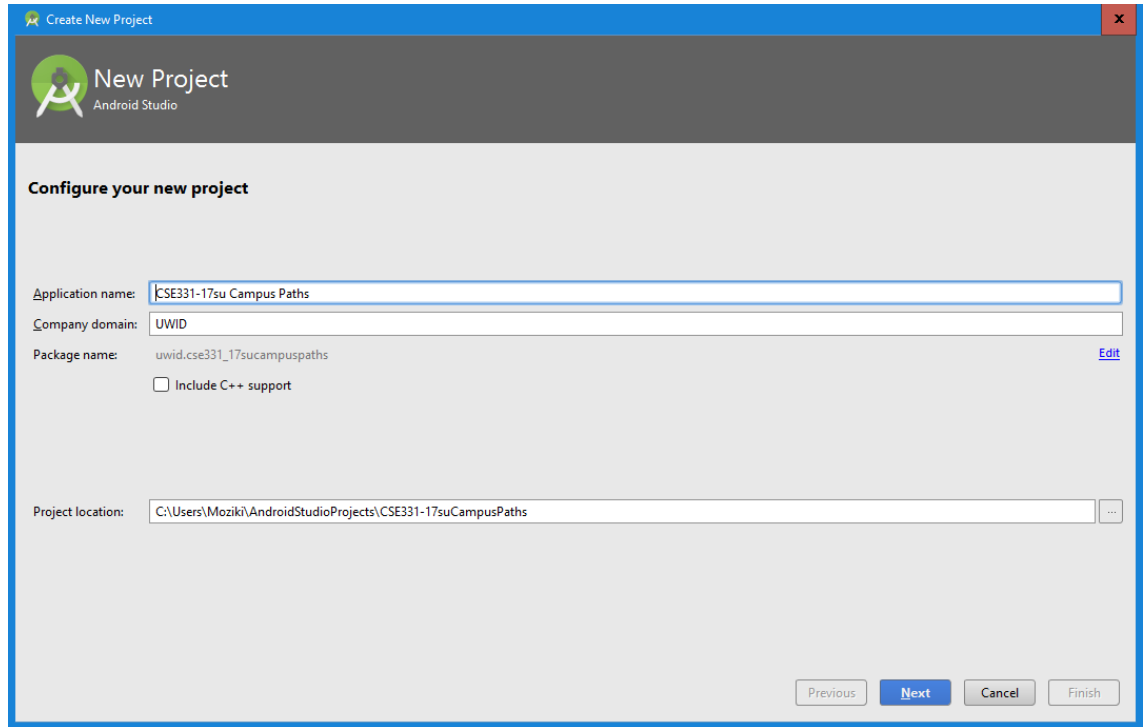
Android Studio uses the IntelliJ IDE which while different from Eclipse, should feel very similar to Eclipse in terms of coding, project layout, and various buttons to run and debug your program.

To create a new Android Studio project, navigate to File -> New -> New Project...



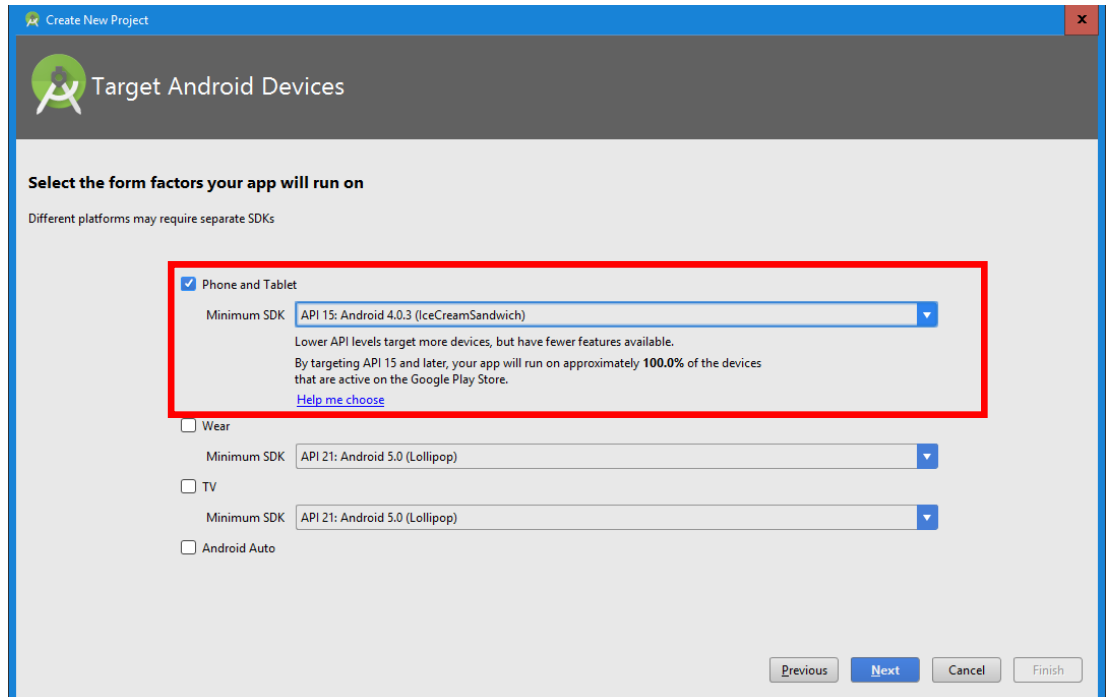
The following prompting panes on the subsequent page will open:

Name your application something appropriate like “CSE331-17su Campus Paths”. For company domain, you can simply use your UW student ID.



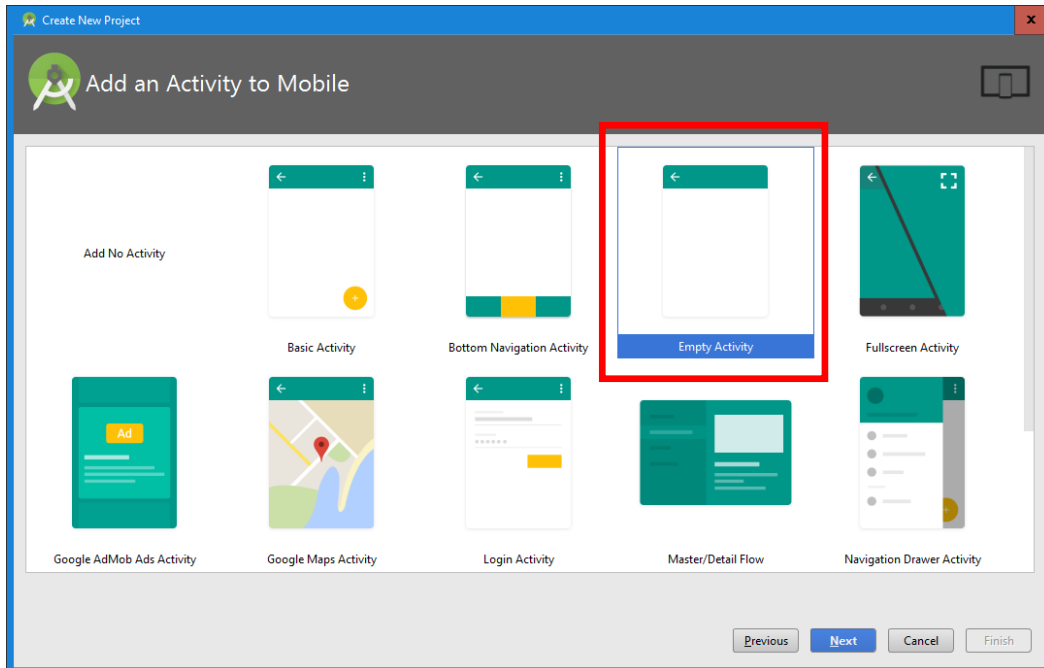
Click “Next” and you will be brought to the following screen:

Make sure “Phone and Tablet” is selected.



For the Campus Paths assignment, using “API 15: Android 4.0.3 (IceCreamSandwich)” as the minimum SDK will suffice for the Campus Paths assignment. Then click “Next” to continue.

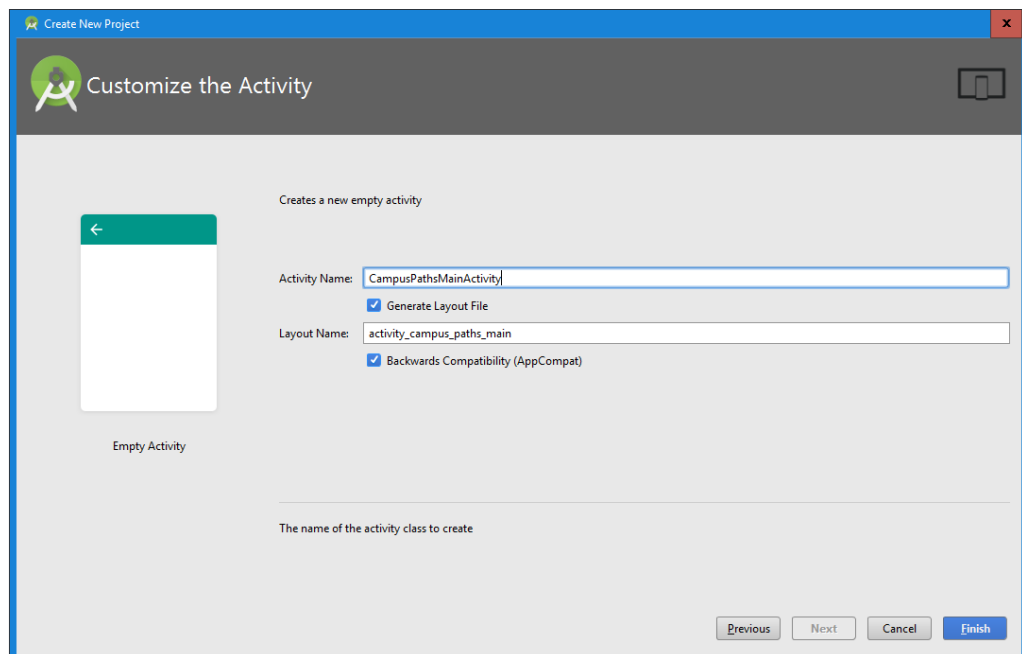
For the initial environment setup, you'll want to start with an "Empty Activity" and press "Next."



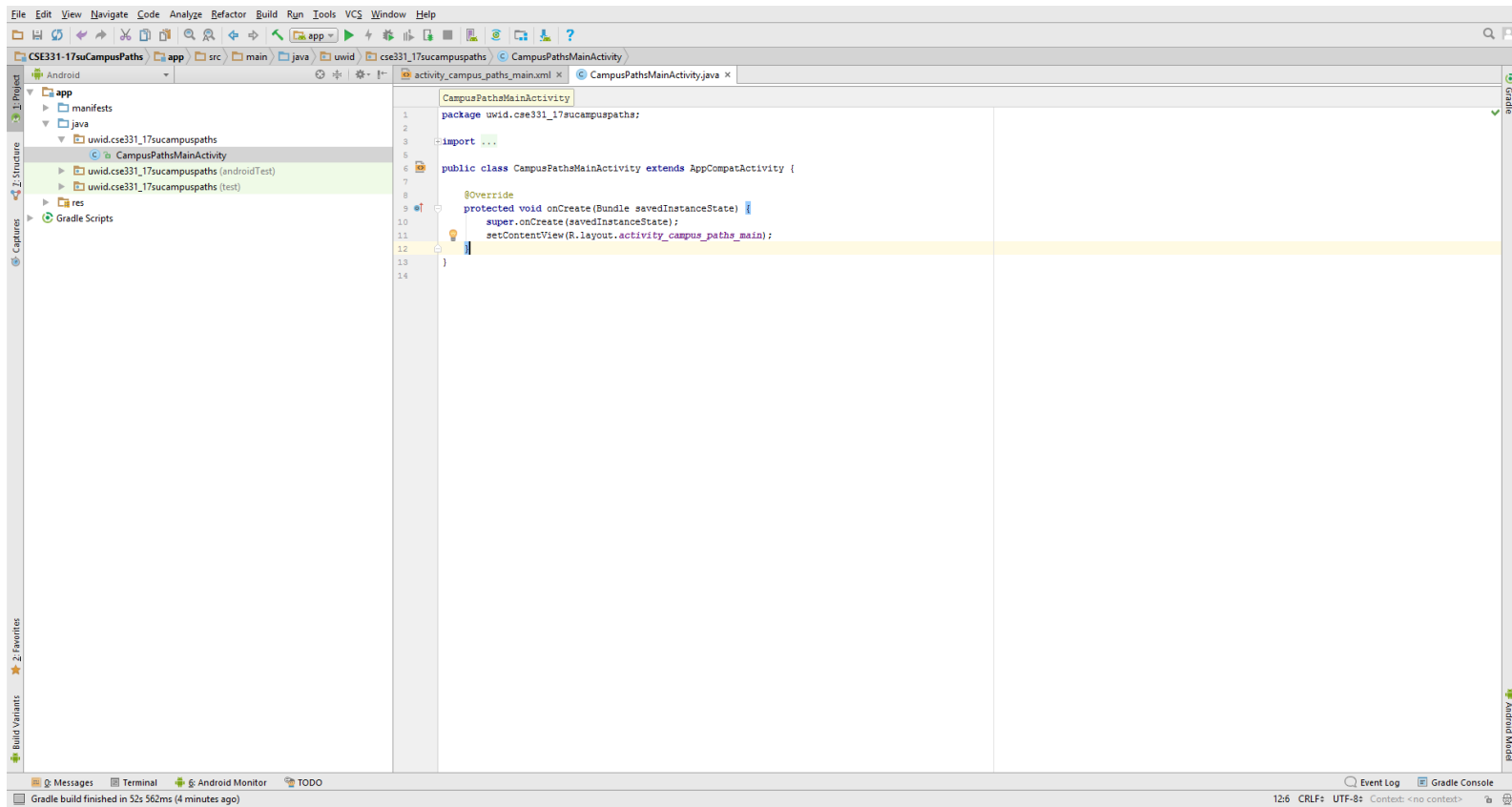
This prepares you with the initial infrastructure to execute the application along with a means of designing the layout and manipulating added components (i.e. Button) later in Java.

The "Main Activity" is the main entry of execution for your Android Application. Furthermore, since the Main Activity automatically pairs to the layout of your application, this will likely be the main mechanism of control for your application (i.e. setting up Buttons, Callbacks, etc.).

Name this Activity something descriptive like "CampusPathsMainActivity" and press "Finish."



Some initialization will occur in until Android Studio creates the new project and looks like this:



*If an error appears in your project stating that “Cannot resolve symbol R” this is simply the IntelliSense catching up – start typing “R.layout” or simply open your activity\_campus\_paths\_main.xml tab and it should fix the false positive.*

You are now ready to start ship and import your previous homework into Android Studio to prepare for the Campus Paths project!



### 3. Shipping and Importing Projects into Android Studio

In order to build off the previous projects you have worked on throughout the quarter, you'll have to "ship" and import your projects into Android Studio using a Jar.

Android Studio has functionality to "Import Projects," however, this ends up creating an entirely new project based on the directory structure of your previous projects. As you can see with the activity created previously, the directory structure for an Android application is very unique. Therefore, by importing the project using Android's import settings, you will not be able to create an Activity (and thus not be able to launch any application in Android) because it will not know how to reconfigure the directory structure.

So, you'll have to import your previous homework assignments as a library – similar to how you would ship code and libraries you produced to clients who want to use it in industry. We'll accomplish this by packaging the Java homework assignments into a Jar. Make sure you have the most recent commit in your local copy of the homework too in Eclipse – the build.xml file will package homework 5-8 into a single Jar file to be used as the library.

*Note that once you package your homework assignments into a Jar, it will exclude the testing files that you wrote when initial writing your classes. Make sure your programs are tested thoroughly and you are confident they all work as expected before you create your Jar – you won't be able to edit the files in Android Studio unless you create an entirely new Jar then reimport the libraries!*



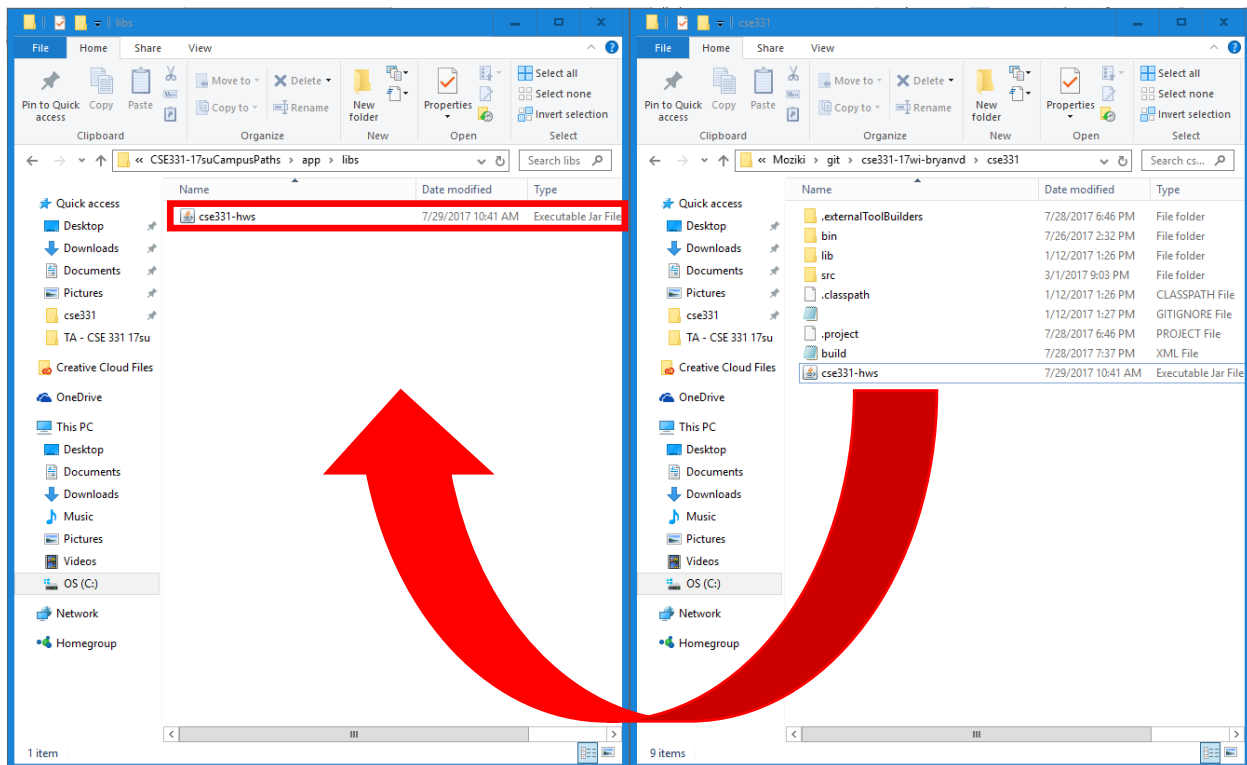
*As of Summer 2017, Android Studio does not support Java 8. Attempting to compile and build your projects using JDK 1.8 (Java 8) will result in your application not being able to launch despite the libraries importing seemingly alright! As a result, since a majority of CSE 331 is completed under the assumption of Java 8, the following section will guide you through installing the previous JDK, JDK 1.7 (Java 7) which Android Studio is able to support.*

Refer to the other guide (separate from this one) about building your project and creating a Jar in JDK 1.7 (Java 7) before continuing.

Once the Jar file has been created, navigate to your Android Studio Project's "libs" folder.

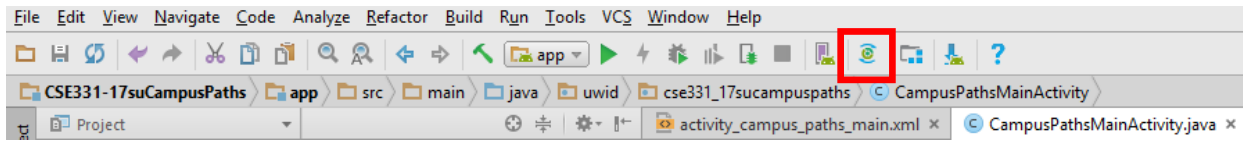
More specifically, navigate to: AndroidStudioProjects -> CSE331-17suCampusPaths -> app -> libs

Move/Copy the created "cse331-hws" Jar into the libs folder of your project.

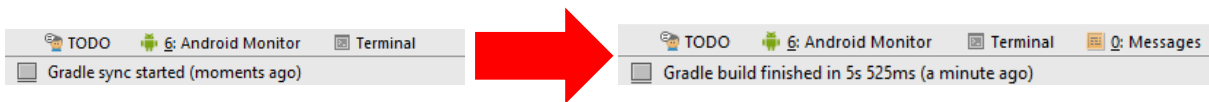


Your Android Studio project should now include all the projects you worked on previously throughout the quarter! Simply let a Gradle Build run and try importing one of your homework assignments.

Gradle Builds tend to run in the background and on their own in Android Studio, however, to manually start a Gradle Build, simply press the icon at the top of Android Studio shown below which performs a “Sync Project with Gradle Files.”



You’ll be able to tell the build is running because at the bottom of Android Studio, it’ll say “Gradle Sync started (moments ago)” in the bottom left. Wait for the files to stop synchronizing and Android Studio to now report: “Gradle build finished in...”



Usually Gradle builds take much longer than 5 seconds, don’t worry if yours takes longer!

With the Gradle Build complete and your homework projects successfully imported as a new library, you should be able to import the individually homework files similar to how you have always been working in Eclipse.

For instance, say I wanted to create a new Graph inside my MainActivity “onCreate” method (which would be a very poor idea – exposing how your Campus paths is represented – and should be removed after testing this when you actually go to make your Android Application).

I simply include the line, “import hw5.\*” just like in Eclipse then create my instance of Graph like so:

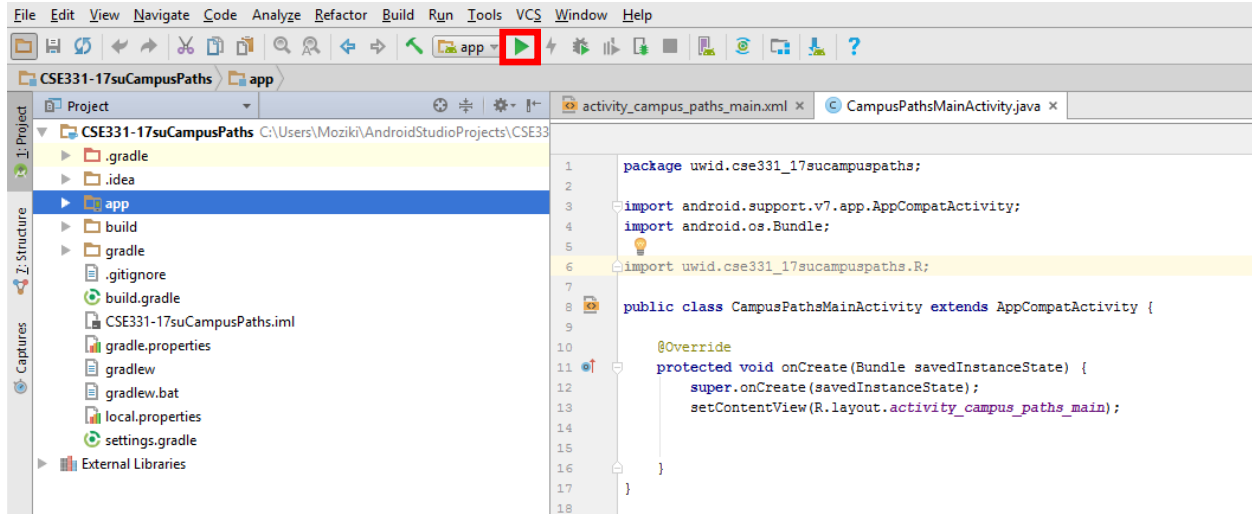
```
activity_campus_paths_main.xml x CampusPathsMainActivity.java x
CampusPathsMainActivity onCreate()
1 package uwid.cse331_17sucampuspaths;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5
6 import uwid.cse331_17sucampuspaths.R;
7
8 import hw5.*;
9
10 public class CampusPathsMainActivity extends AppCompatActivity {
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_campus_paths_main);
16
17         Graph graph = new Graph();
18
19     }
20 }
21
```

Your project should now be able to access all your previous homework assignments seamlessly through a process similar to shown above! The next step is to start creating and adding widgets, images, and more to your application!

## 4. Installing an Android Emulator to run the Application

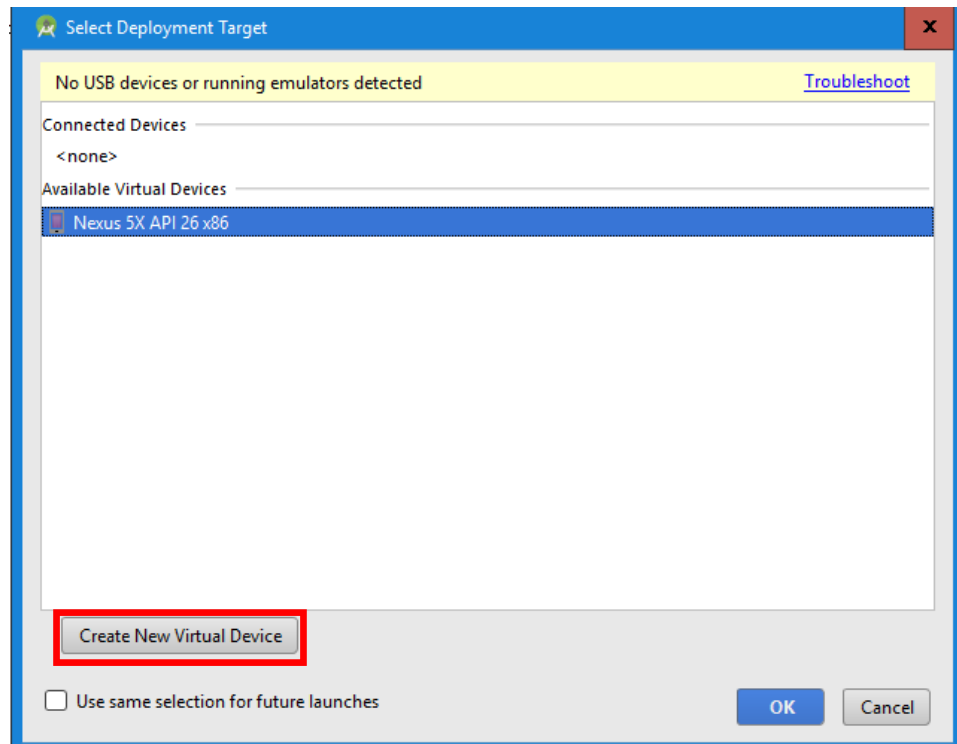
In order to run and test your application on an Android device, you'll have to install an Android emulator on your computer.

To do this easily, simply try running your application – pressing the “Run” button at the top of Android Studio.



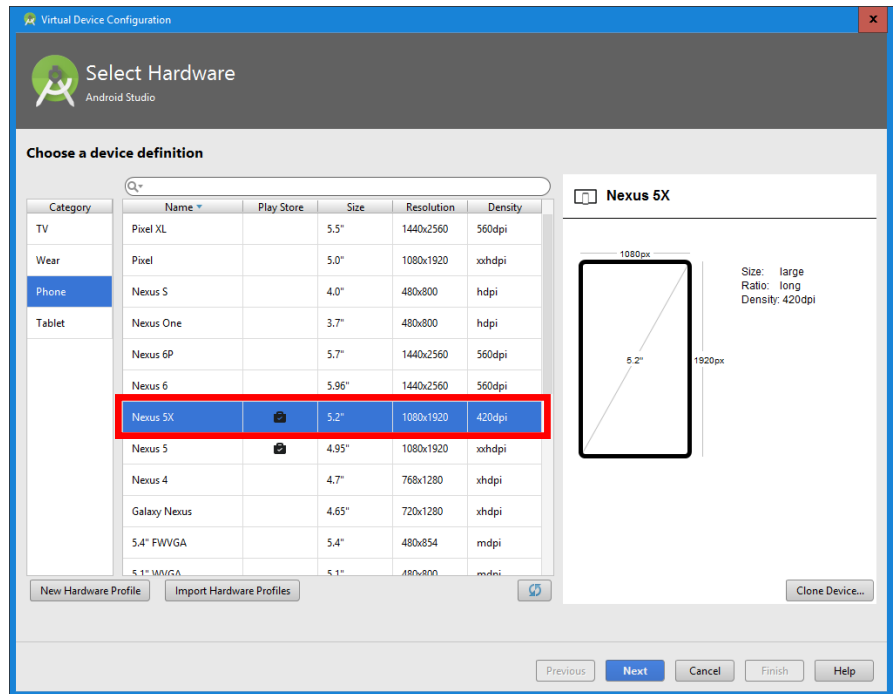
This will bring up a pane prompting you to choose which virtual device you would like to run your application on. Simply click “Create New Virtual Device” to start installing a new Android Emulator.

*Note that I already have an Android Emulator installed when writing this guide so our “Select Deployment Target” panels may look slightly different!*

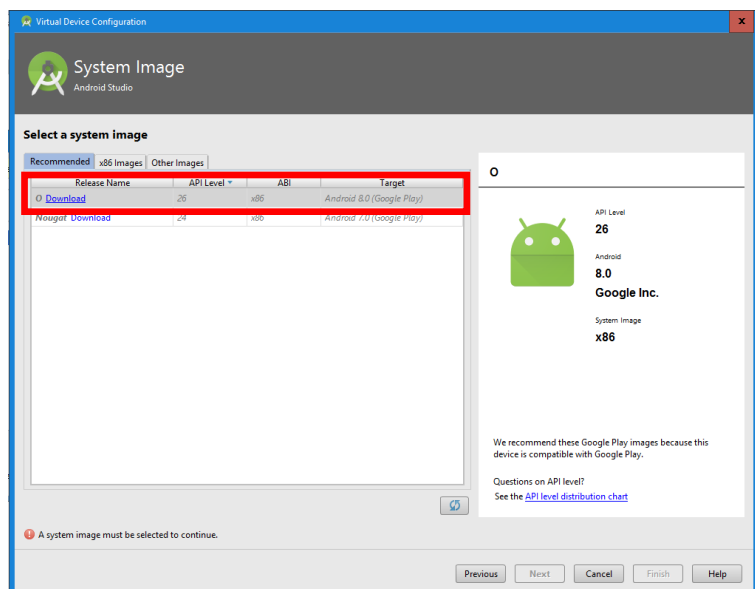


For grading this assignment, we'll be running your Android Application on a Nexus 5X, however, if you want to try running your application on other Android devices, this is strongly encouraged! When producing applications for clients and users, you'll want to make sure it looks good, if not the same, on all devices. Furthermore, it is easy to install multiple different emulators all in one place with Android Studio.

Select the Nexus 5X, or your other device of choice and click "Next."



You'll have to install an Android System Image to run the emulator. We'll be using the highest Android API level for our emulator software in this assignment. To download the "O" system image, simply click the adjacent "Download" button next to this selection in the "Recommended" tab.

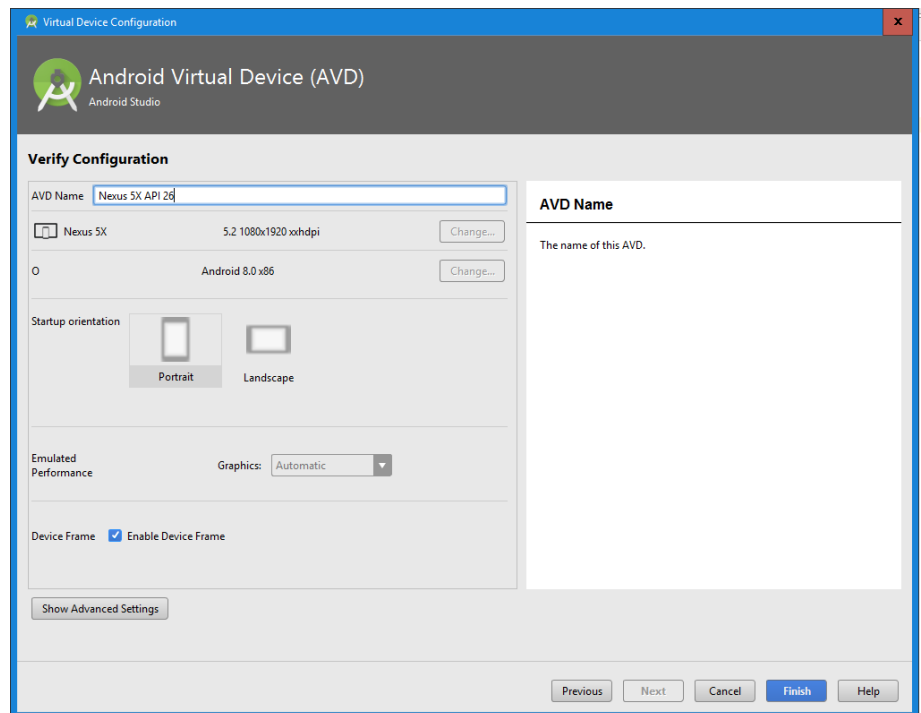


*Note that when we created our Android project in section 2, we chose an API that would work on 100% of Android devices. Despite our emulator having a very high API level, our application will work fine on it!*

Accept the license agreement in the following panel that pops up and press “Next” to continue with the installation.

Once the installation is complete, press “Finish” and “Next” to continue with the installed system image selected.

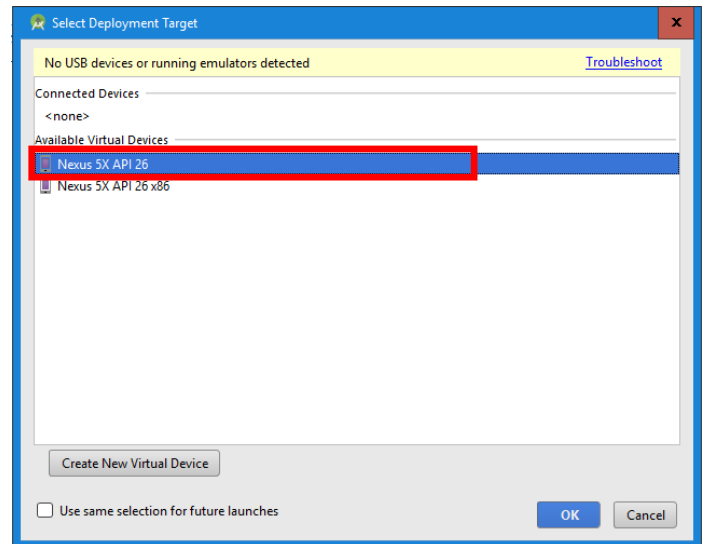
At this point all the default settings are fine for our emulator. Press “Finish” to complete the installation.



Choose the newly installed “Nexus 5X API 26” and press “OK” to boot up the emulator.

Wait for the emulator to load and the complete Android OS to load as well.

*If the OS isn't loading and reports an error like: “System UI isn't responding” simply choose the option to “Close app” to any/all of them and it should reboot correctly and quickly.*



Once the emulator has loaded, wait a few seconds and your Android application should launch automatically. (Unless a crash like the one described above occurred, then simply press the “Run” button again in Android Studio).

If everything works correctly, your emulator should launch the blank application and look something similar to this:

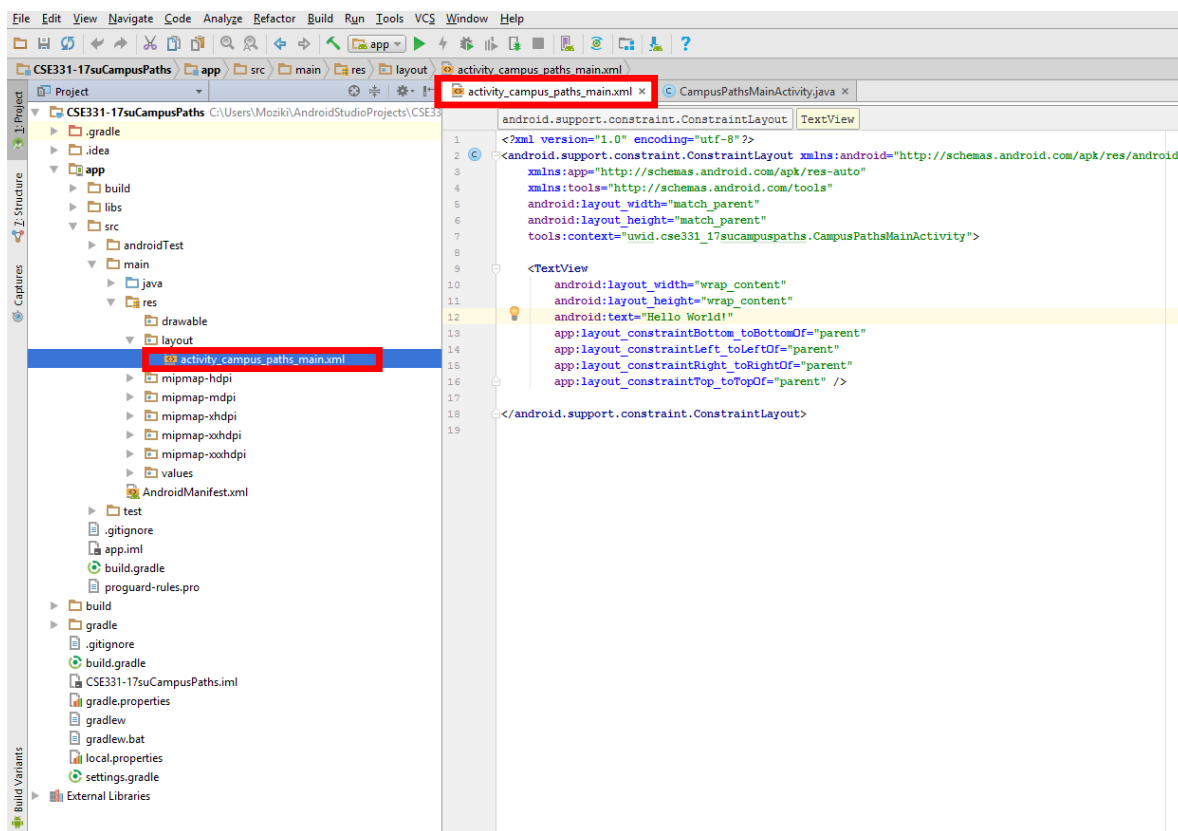


## 5. Adding a Button to your Application

To add widgets (like Buttons) and other components to your Android application, Android Studio uses a graphical interface to visually move and apply components to the view.

Navigate to your “activity\_campus\_paths\_main.xml” (or other Main Activity xml file). This tab may already be open next to your MainActivity.java, however, if you close this pane here is how you navigate to the Main Activity xml file:

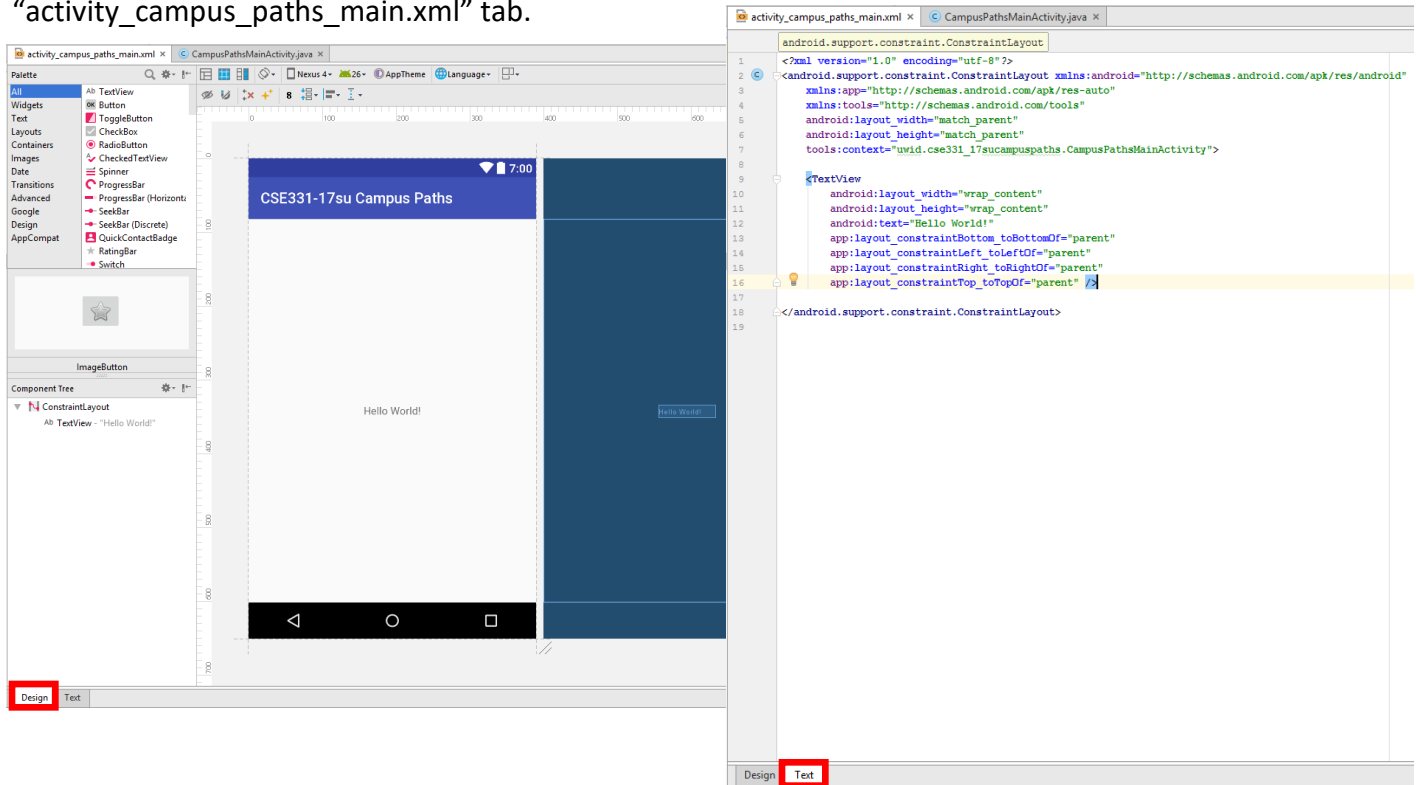
Under your project folder, find app -> src -> main -> res -> layout



The “activity\_campus\_paths\_main.xml” may appear in either a “Text” or “Design” layout.



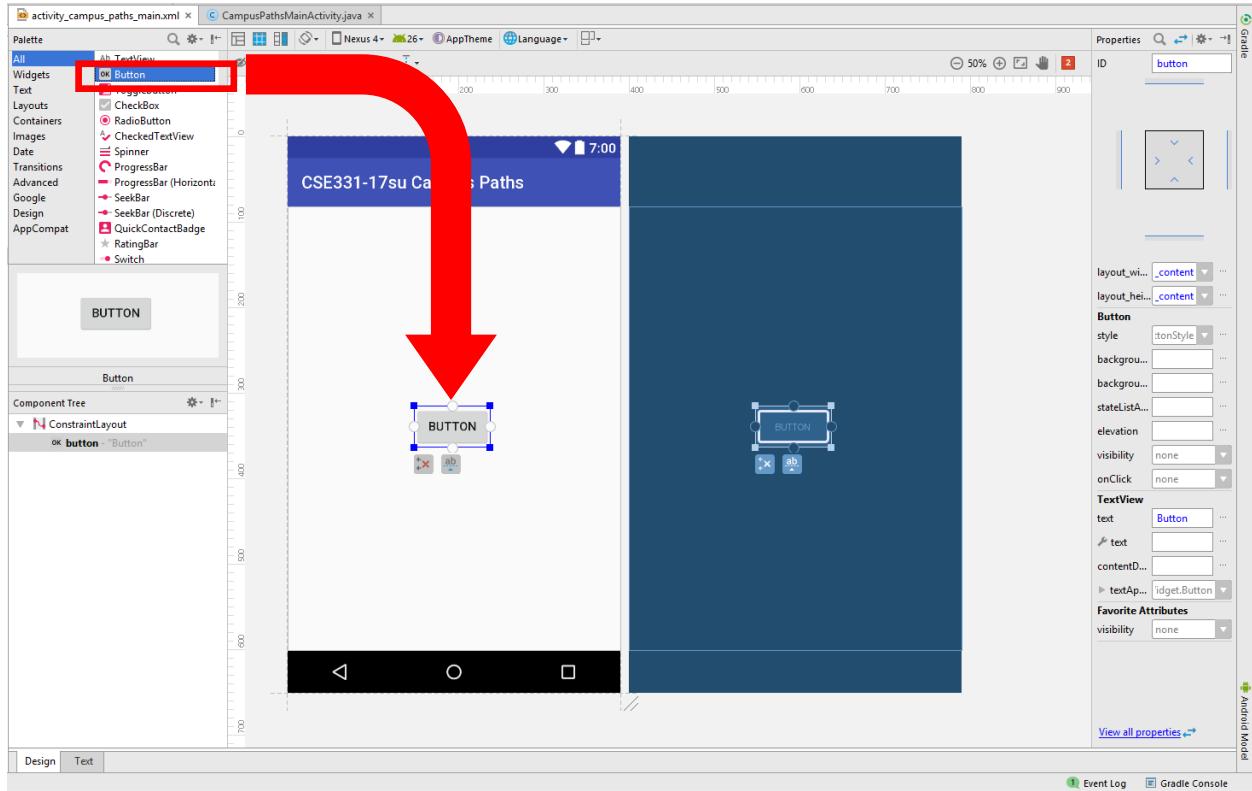
Both have their own useful ways of modifying the application layout – to change between the two easily simply click the respective buttons at the bottom of Android Studio when in the “activity\_campus\_paths\_main.xml” tab.



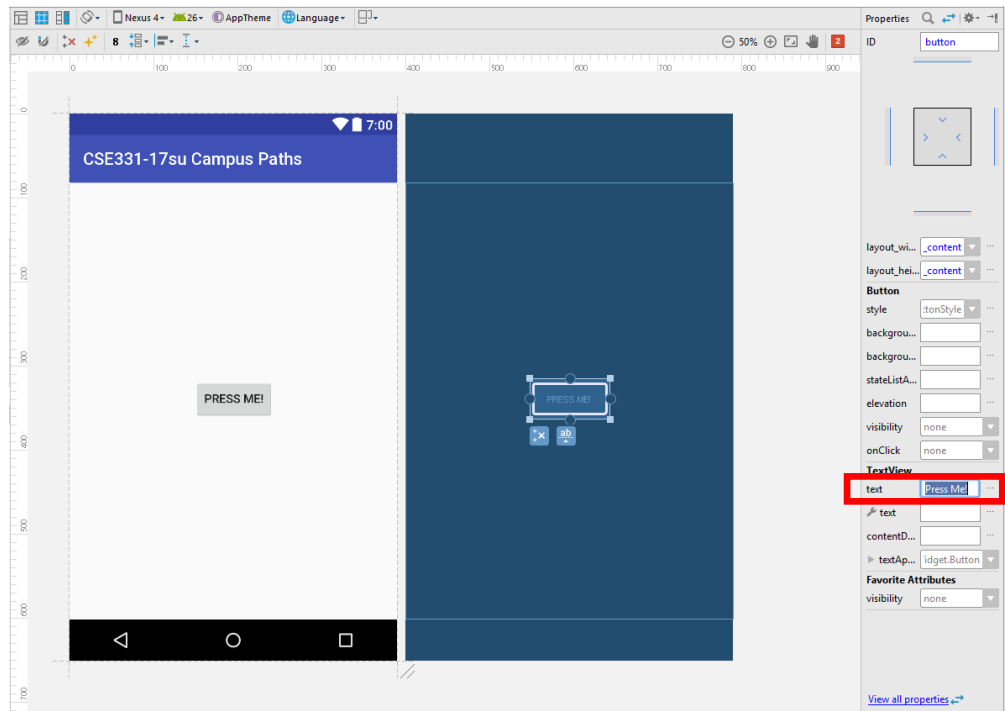
To initially add a Button, you’ll want to be in the “Design” layout.

*Feel free to remove the “Hello World!” text by simply clicking on the element either in the graphical layout or in the “Component Tree” off to the side and press Delete.*

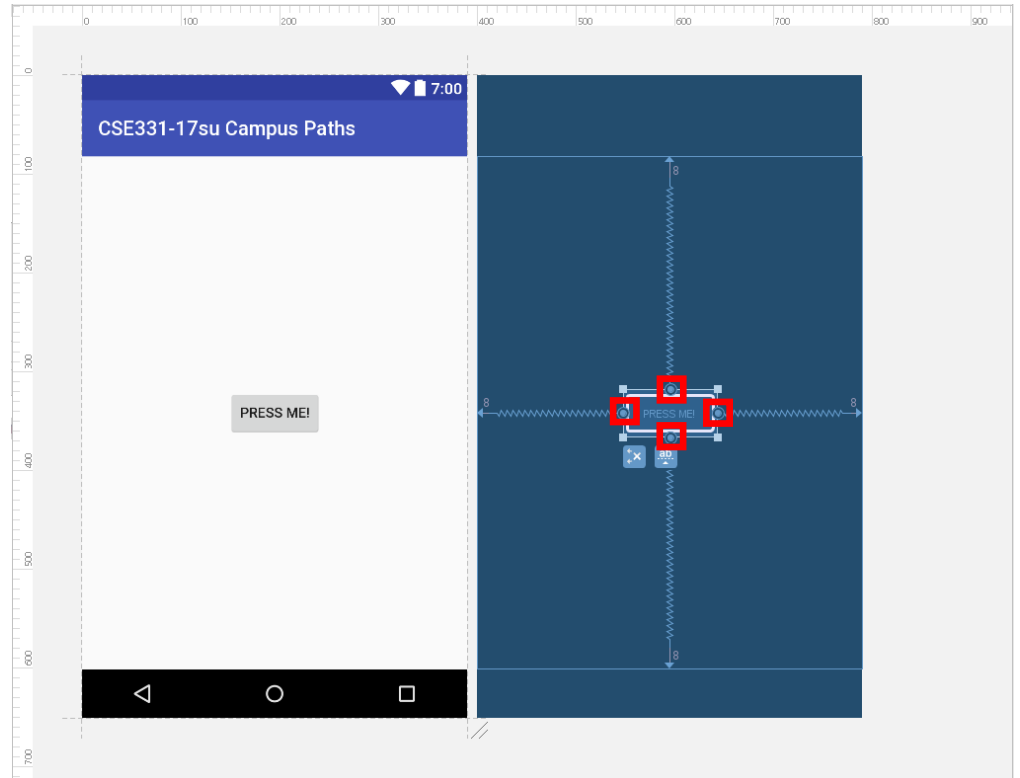
Select "Button" in the upper-left palette and drag it into the graphical view of your Android application layout.



We'll rename this Button by editing the "text" data in the "Properties" tab on the right.

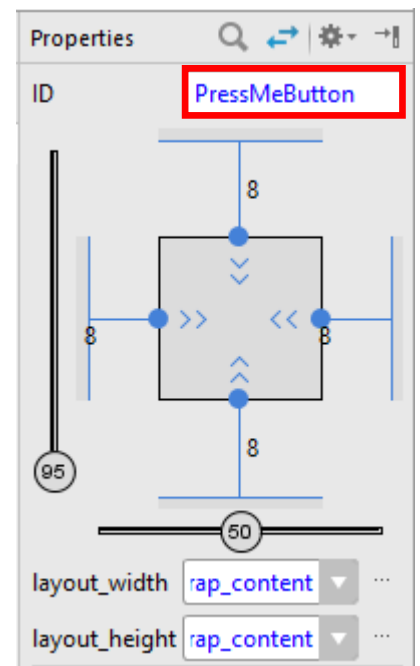


Next, anchor the button to its position in the center of the screen by dragging from the solid dark blue circles in the center of each side of the rectangular button. For now, simply dragging from the solid circles to each border of the application screen will be fine. The circles should fill in as a lighter blue when they are properly anchored.



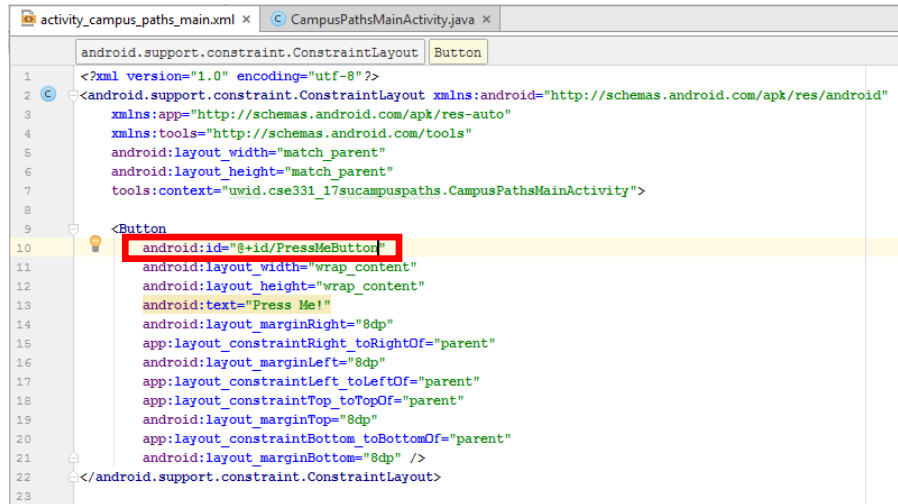
With the Button added to your application layout, you'll want to rename it to something identifiable for when you're programming callback functionality in Java.

To change the ID of your Button which you'll use to find the correct widget and create a reference to in code later, with the Button selected under the "Properties" tab in the "Design" layout, in the upper right-hand corner is a text box to edit the ID.



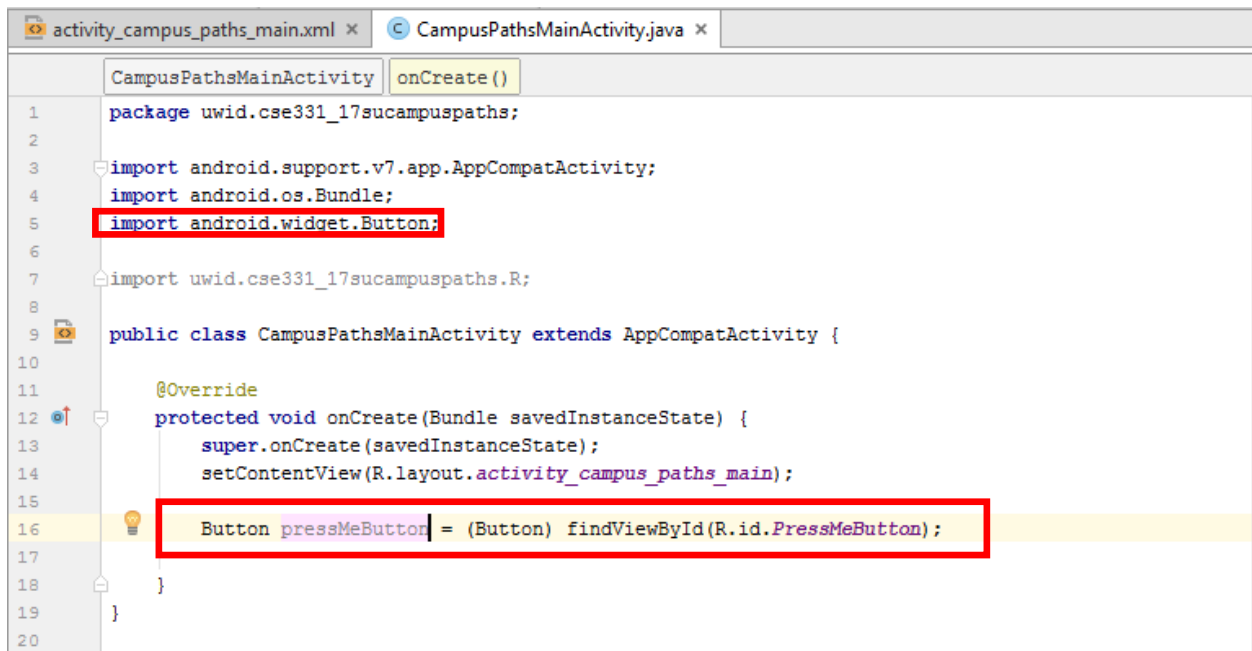
Alternatively, switch to the “Text” view in your Main Activity xml file. Notice under “<Button” a line saying “android:id=@id/button” – change the ending portion of this ID (the text after the ‘/’) to name the Button something more recognizable.

*Note that the technique described previously updates this text ID automatically.*



```
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     tools:context="uwid.cse331_17sucampuspaths.CampusPathsMainActivity">
8
9     <Button
10         android:id="@+id/PressMeButton"
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:text="Press Me!"
14         android:layout_marginRight="8dp"
15         app:layout_constraintRight_toRightOf="parent"
16         android:layout_marginLeft="8dp"
17         app:layout_constraintLeft_toLeftOf="parent"
18         app:layout_constraintTop_toTopOf="parent"
19         android:layout_marginTop="8dp"
20         app:layout_constraintBottom_toBottomOf="parent"
21         android:layout_marginBottom="8dp" />
22 </android.support.constraint.ConstraintLayout>
23
```

Inside your CampusPathsMainActivity.java file now, you’ll have to create a reference to this specific Button. The easiest way to do this is by using the “findViewById” method which takes in an ID from your Main Activity xml file by using “R.id.PressMeButton” or other Android ID found in the xml file. Be sure to cast the result to the proper type of component! Also, don’t forget to import the proper class!



```
1 package uwid.cse331_17sucampuspaths;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.widget.Button;
6
7 import uwid.cse331_17sucampuspaths.R;
8
9 public class CampusPathsMainActivity extends AppCompatActivity {
10
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_campus_paths_main);
15
16         Button pressMeButton = (Button) findViewById(R.id.PressMeButton);
17
18     }
19 }
20
```

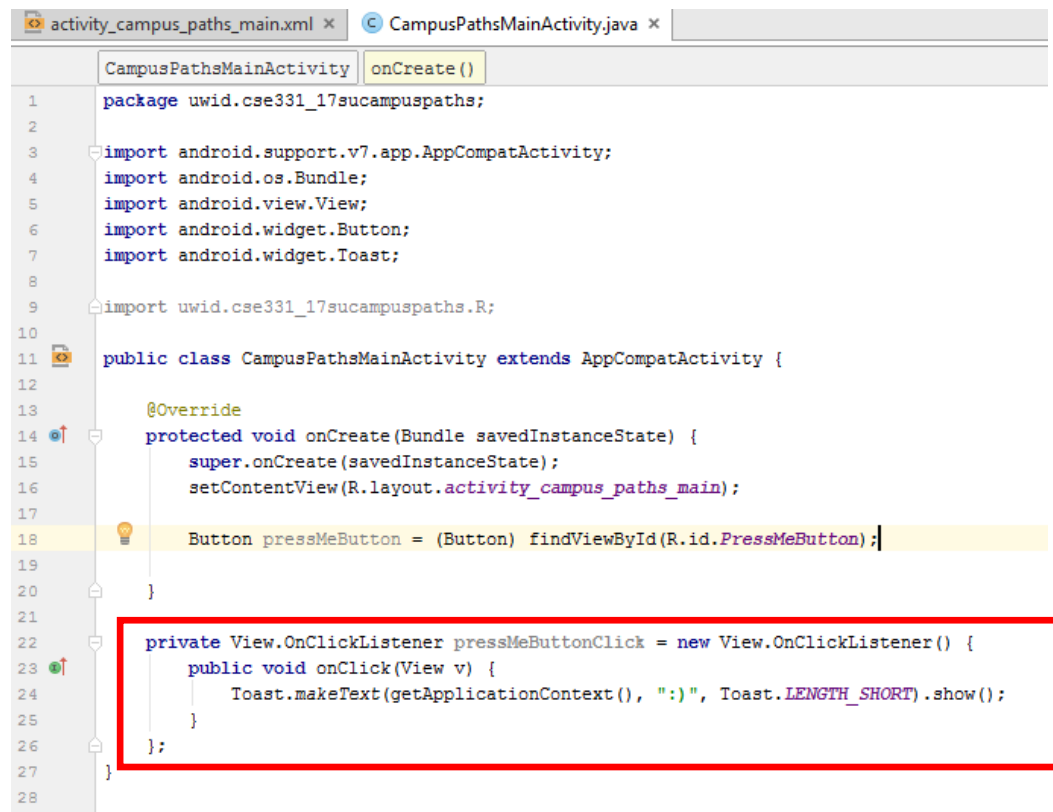
With the new button variable created, you'll want to add callback functionality so your application responds when the user presses your button.

To do so, you'll have to add an "OnClickListener" to your button. This can be done in a number of different ways by creating an anonymous inner class or even an entirely different class. For this guide, I'll show you how to use an anonymous inner class (as it is the most practical and least tedious version to implement for listeners who have unique functionality not shared by other buttons).

Outside of your "onCreate" method and CampusPathsMainActivity class, add a new private class like the one shown below:

Notice the new import statements in addition to the syntax of the anonymous inner class declaration and method declaration.

*Two versions of Android OnClickListener exist – one in "Dialog Interface" and one in "View." Make sure to select the View version of the OnClickListener.*



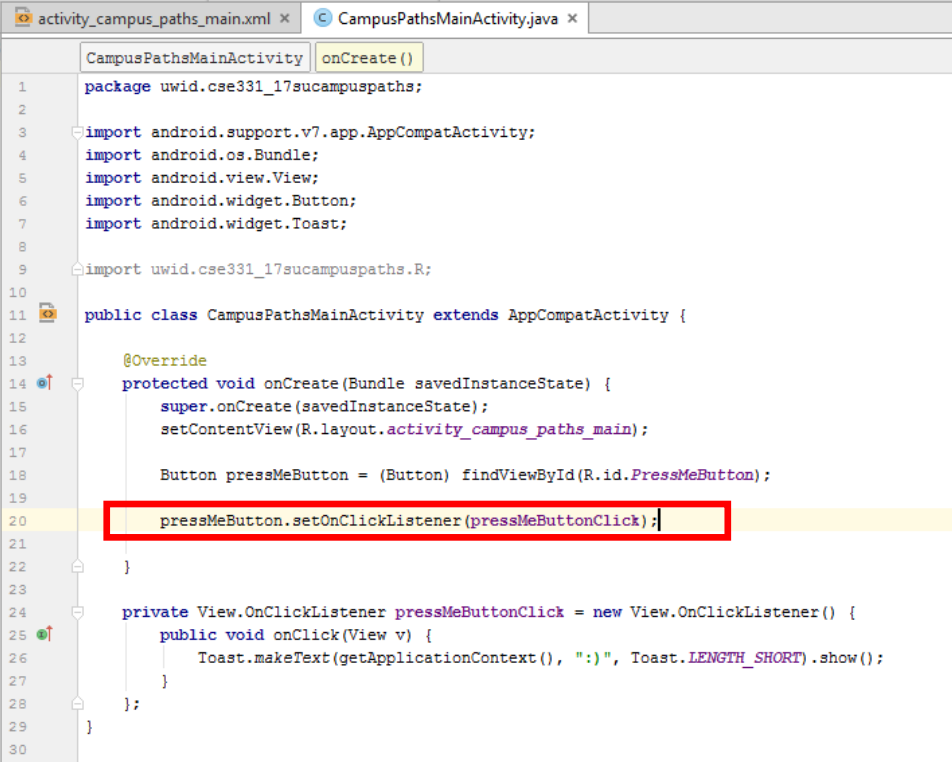
```
activity_campus_paths_main.xml x CampusPathsMainActivity.java x
CampusPathsMainActivity onCreate ()
1 package uwid.cse331_17sucampuspaths;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.view.View;
6 import android.widget.Button;
7 import android.widget.Toast;
8
9 import uwid.cse331_17sucampuspaths.R;
10
11 public class CampusPathsMainActivity extends AppCompatActivity {
12
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_campus_paths_main);
17
18         Button pressMeButton = (Button) findViewById(R.id.PressMeButton);
19
20     }
21
22     private View.OnClickListener pressMeButtonClick = new View.OnClickListener() {
23         public void onClick(View v) {
24             Toast.makeText(getApplicationContext(), ":", Toast.LENGTH_SHORT).show();
25         }
26     };
27 }
28
```

With the class declaration shown, a new object called "pressMeButtonClick" is created alongside the definition of the anonymous inner class.

You must override the “onClick” method in your new anonymous inner class. This is where the functionality you want paired when the button is clicked will go. Alternatively, after adding the OnClickListener to your button, the “onClick” method overridden in the listener will be the callback occurring when the button is clicked.

*Notice the line in the body of the “onClick” method – “Toast.makeText” is similar to printing to the console in Eclipse except in your Android application, a bubble of text will display in the bottom center of the screen. This is useful for testing initial callback functionality in your application to ensure everything is working as expected when interacting with the UI.*

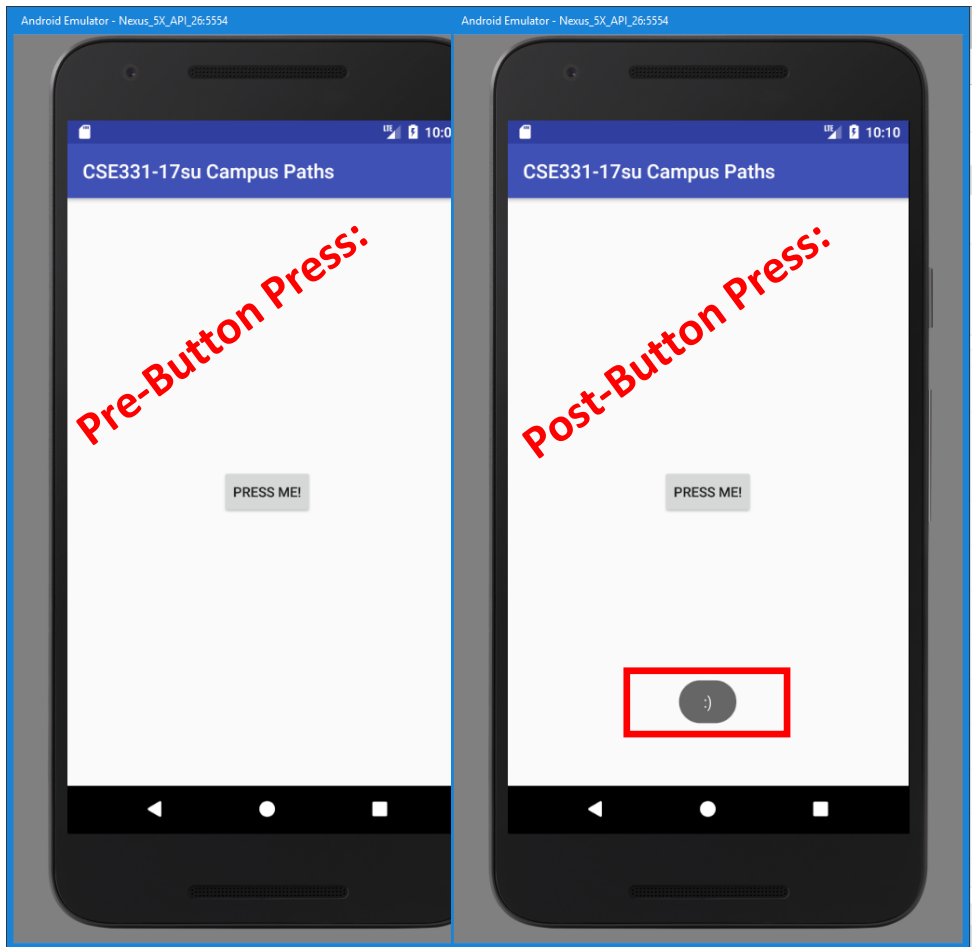
Finally, to set the listener to your found button, simply invoke the method “setOnClickListener” on your Button reference passing in the created OnClickListener object from the definition of the anonymous inner class.



```
activity_campus_paths_main.xml × CampusPathsMainActivity.java ×
CampusPathsMainActivity onCreate ()
1 package uid.cse331_17sucampuspaths;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.view.View;
6 import android.widget.Button;
7 import android.widget.Toast;
8
9 import uid.cse331_17sucampuspaths.R;
10
11 public class CampusPathsMainActivity extends AppCompatActivity {
12
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_campus_paths_main);
17
18         Button pressMeButton = (Button) findViewById(R.id.PressMeButton);
19
20         pressMeButton.setOnClickListener(pressMeButtonClick);
21
22     }
23
24     private View.OnClickListener pressMeButtonClick = new View.OnClickListener() {
25         public void onClick(View v) {
26             Toast.makeText(getApplicationContext(), ":", Toast.LENGTH_SHORT).show();
27         }
28     };
29 }
30
```

Now when the user clicks your Button in your application, whatever behavior was defined in the overridden “onClick” method of that OnClickListener will be the executed on the callback.

Clicking the “Run App” green ‘play’ button at the top of Android Studio and firing up an Android Emulator, you’ll see the listener in action!



## 6. Adding an ImageView and making it 'Drawable' in your Application

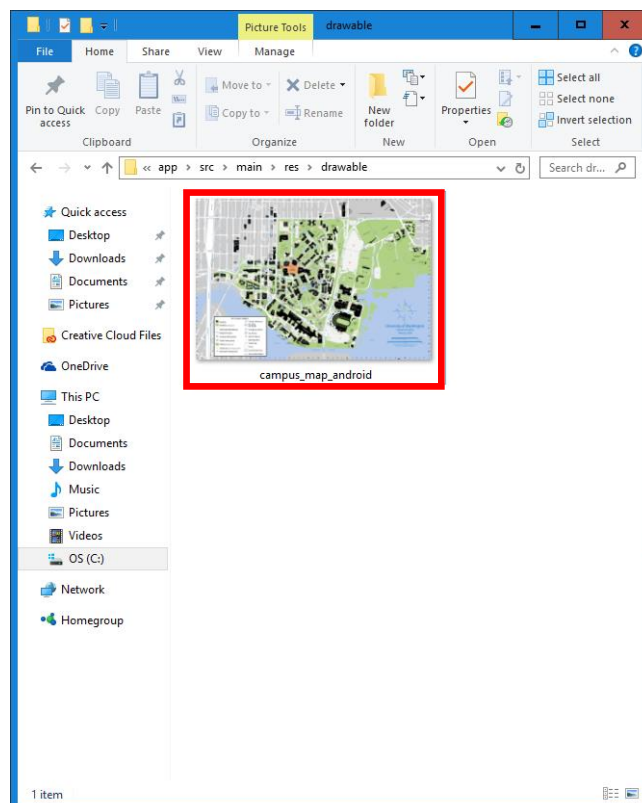
In order to have Android Studio be able to recognize and load our image of the campus map, you'll want to move the file to the "drawable" directory.

*\*\* Note that your Android application WILL crash attempting to use the original campus map image designed for creating the GUI in Java Swing on launch. As a result, to avoid this crash, you can either use the scaled down version of the campus map I created ("campus\_map\_android" which has its resolution scaled down to 50% of the original) or to scale down the image manually (reducing its overall resolution) in an image editor using trial-and-error with resolutions until the application no longer crashes on launch. When creating this application, the campus map image scaled to 50% of its original resolution no longer caused any crashes when loading on my end.*

Navigate to the "drawable" directory in your Android Studio Project.

Specifically, in your Android Studio project, navigate to: app -> src -> main -> res -> drawable

Copy the modified campus map image, "campus\_map\_android" (with correct resolution to prevent crashes), into the "drawable" folder.

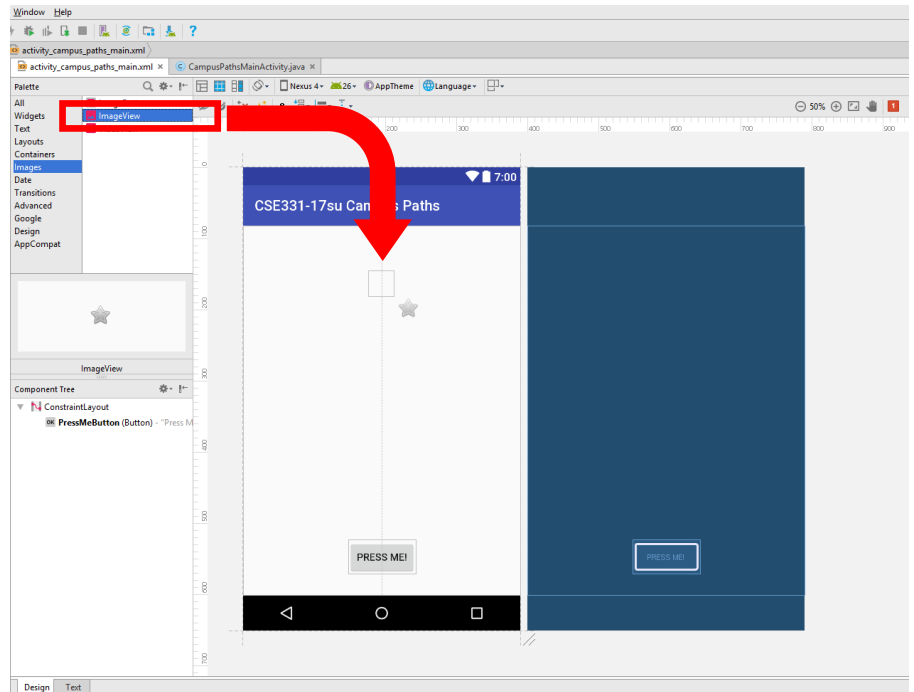




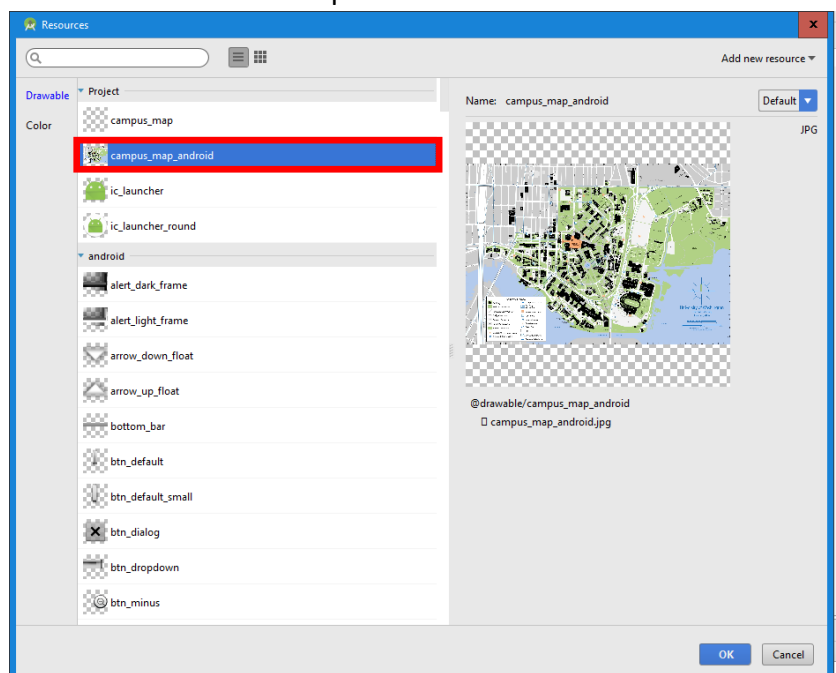
In order to create an image of the UW campus map in your application, you'll want to add an ImageView component to your Main Activity xml file (in the Design view).

*I moved my "PRESS ME!" Button created in the previous section downward in order to create space for the campus map image – feel free to do the same yourself by simply clicking and dragging the Button downward.*

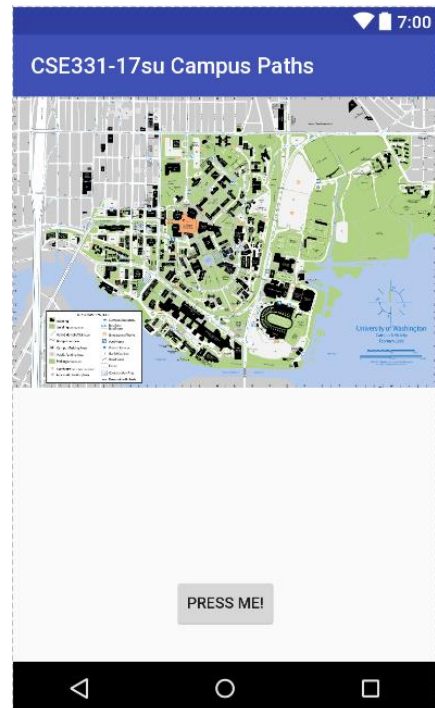
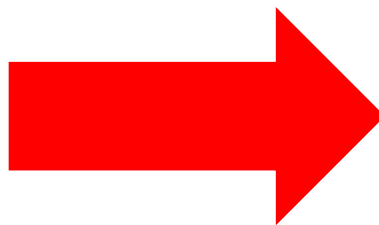
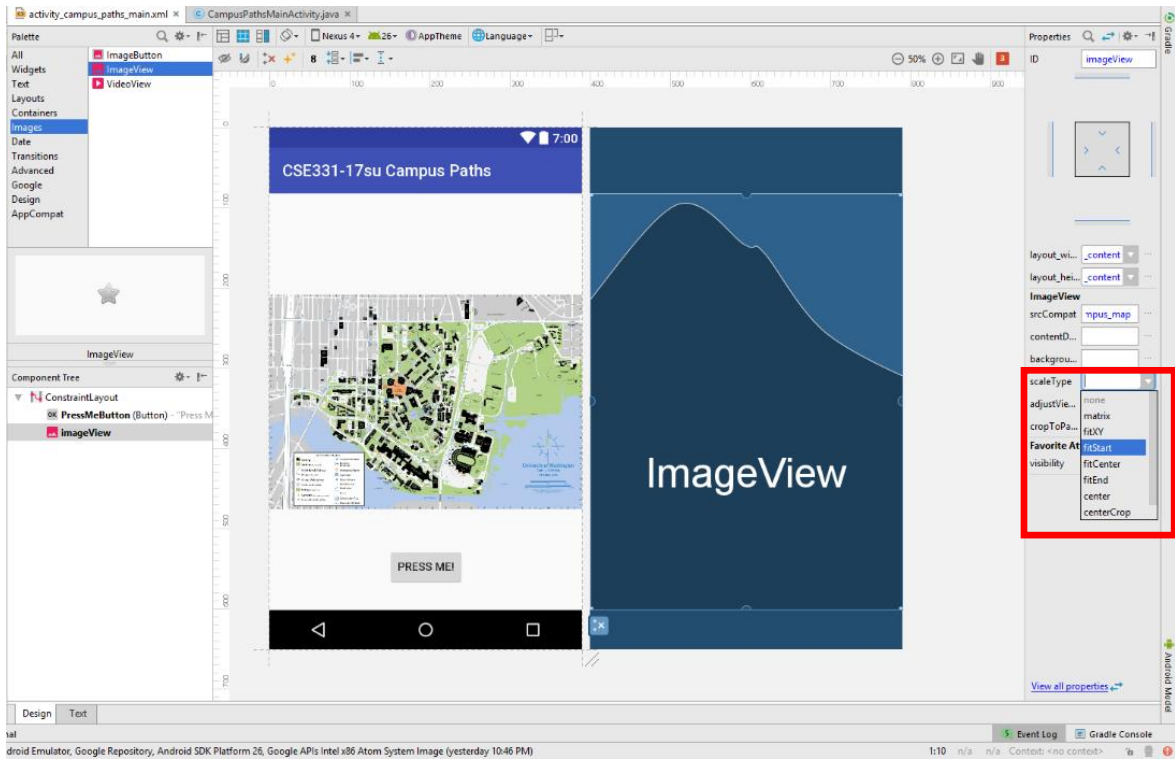
On the design "Palette," navigate to the "Images" tab and drag and drop an ImageView into your design layout. This will prompt a new pane to open.



With the campus map image now in the "drawable" folder, it should appear at the top of this pane, if not, click on "Drawable" off to the left side to make the contents of the "drawable" directory appear. Click on "campus\_map\_android" to select it then press "OK" to continue.



After loading the campus map image, drag the ImageView to center it to the design layout screen. With the ImageView selected, off to the right of Android Studio and under the “Properties” tab, open the “scaleType” dropdown menu and change its orientation to “fitStart.”

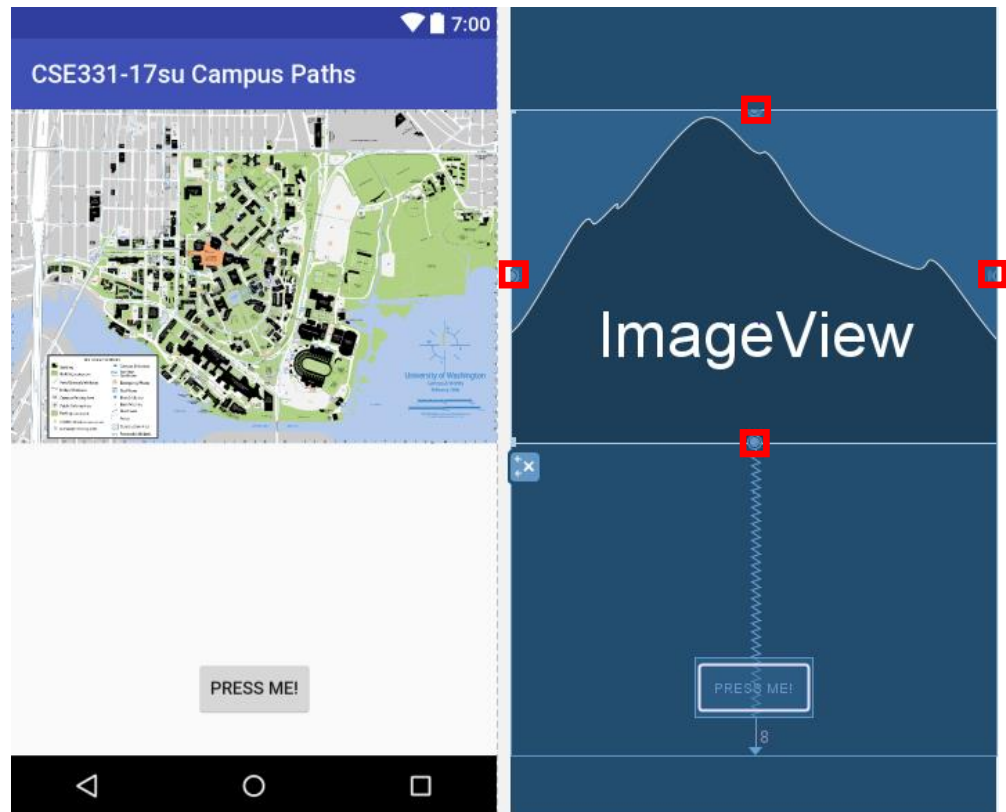


*The reason for orienting the campus map image to the start of the ImageView pane is due to the fact that the campus coordinates provided are positioned relative to the coordinate (0, 0) being the upper left-hand corner of the image. With Android Studio's ImageView, the coordinate (0, 0) corresponds to the upper left-hand corner of the outlining border in which the image is contained. In the instance of loading your campus map image shown previously, by setting the "scaleType" to "fitStart," the upper-left hand corner of the image will be aligned to the upper-left hand corner of the ImageView thus aligning the origins of the coordinate systems.*

Anchor your image view to each side of the application in the design layout by grabbing the circular pins in the center of each edge of the ImageView rectangle and dragging them to each outer edge of the application. The anchors may be hard to see if the ImageView fills the size of the application – if you're confused on how to anchor the view, look back at Section 5 about adding a button where it is anchored to each side of the screen.

You may want to scale the bottom of the ImageView up slightly from the bottom (since the campus map won't fill the entire view) which will allow for other components to anchor more easily to the ImageView. Feel free to anchor the top of the Button to the bottom of the ImageView as well.

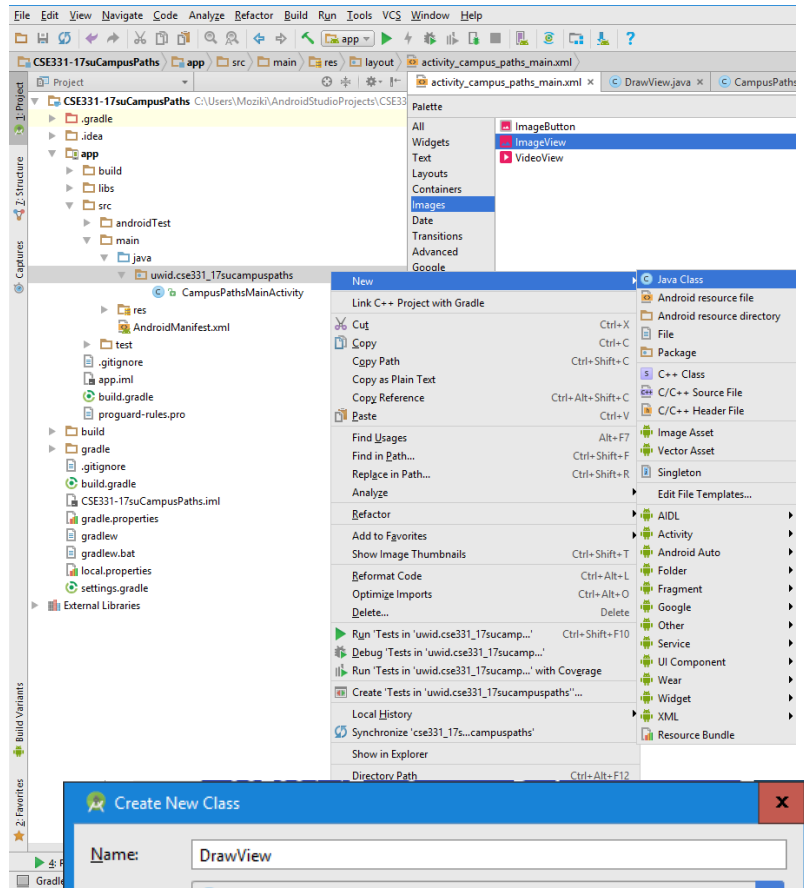
Once the image is anchored, the circular pins should be filled in a light blue color showing they were successfully anchored.



Run your application in an emulator to ensure the image loads correctly (and without any crashes!).

In order to make it possible to draw or paint over the ImageView, you'll have to extend the AppCompatActivity class and override some functionality.

Create a new Java class in your Android Studio project by right clicking on your directory which contains your Main Activity Java class inside the "java" folder (located under app -> src -> main -> java) and choosing "New" then "Java Class." For me this is "uwid.cse331\_17sucampuspaths"



In the pane that opens, name your class something appropriate and similar to ImageView so it implies the functionality off which it is based.

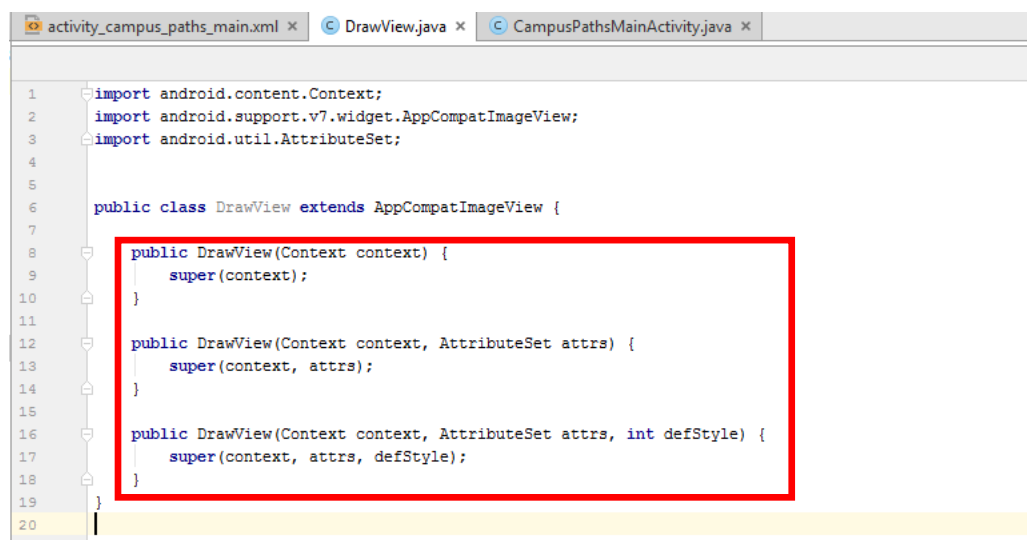
In the box that prompts for a "Superclass," start typing "AppCompatActivity" and choose the class from "android.support.v7.widget."

Make sure you choose the "AppCompatActivity" and not the regular "ImageView!" If you choose the latter, Android Studio will give you a warning explaining how you should extend from the AppCompatActivity version, not the ImageView version.

Conversely, simply type: “android.support.v7.widget.ImageView” into this box. Press “Ok” to continue.

To properly implement your child class of ImageView so we are able to draw on the image, you’ll have to override three constructors and 1 method.

For the constructors, you’ll simply want to invoke the Superclass constructor. Their implementations are shown below. Import Android classes accordingly for parameters passed to constructors which are not recognized.



```
1  import android.content.Context;
2  import android.support.v7.widget.AppCompatImageView;
3  import android.util.AttributeSet;
4
5
6  public class DrawView extends AppCompatImageView {
7
8      public DrawView(Context context) {
9          super(context);
10     }
11
12     public DrawView(Context context, AttributeSet attrs) {
13         super(context, attrs);
14     }
15
16     public DrawView(Context context, AttributeSet attrs, int defStyle) {
17         super(context, attrs, defStyle);
18     }
19 }
20
```

*Later when adding path-drawing functionality to your application, you may want to configure your Paint component in the constructors as well. Don’t worry about this at this point as we’ll be addressing Paint components later in the guide.*

Next, you’ll want to override the “onDraw” method. This is a protected method that is invoked whenever the state of your ImageView has been “invalidated.” We’ll cover more on how to invalidate your image to cause the view to be updated with the newly drawn assets later. For now, know that you should never call the “onDraw” method yourself!

To override the “onDraw” method, look at the starter implementation below (importing Android classes when appropriate, like the Canvas class):

```
activity_campus_paths_main.xml x DrawView.java x CampusPathsMainActivity.java x
DrawView onDraw()
1 import android.content.Context;
2 import android.graphics.Canvas;
3 import android.support.v7.widget.AppCompatImageView;
4 import android.util.AttributeSet;
5
6
7 public class DrawView extends AppCompatImageView {
8
9     public DrawView(Context context) {
10         super(context);
11     }
12
13     public DrawView(Context context, AttributeSet attrs) {
14         super(context, attrs);
15     }
16
17     public DrawView(Context context, AttributeSet attrs, int defStyle) {
18         super(context, attrs, defStyle);
19     }
20
21     @Override
22     protected void onDraw(Canvas canvas) {
23         super.onDraw(canvas);
24
25     }
26 }
27
```

Notice that the super method “onDraw” is invoked right away. By not invoking the superclass’ method, the image you loaded and in this case, the campus map, will no longer appear once this method is called.

After invoking the superclass’ “onDraw” method, you can add any drawing functionality you desire. This could consist of drawing a circle, drawing a nicely labeled dog named “Muffin,” or even drawing the shortest path between two buildings on campus!

We’ll start with the first one because that seems to be the simplest for now and leaves the challenge of drawing the shortest path between two buildings, or even a nicely labeled dog (named “Muffin” accordingly) for you to solve later. For reference though, here is a skillfully drawn image of “Muffin.”



In your “onDraw” method, create a new “Paint” object by calling the Paint constructor as shown below (be sure to import the “Paint” class!):

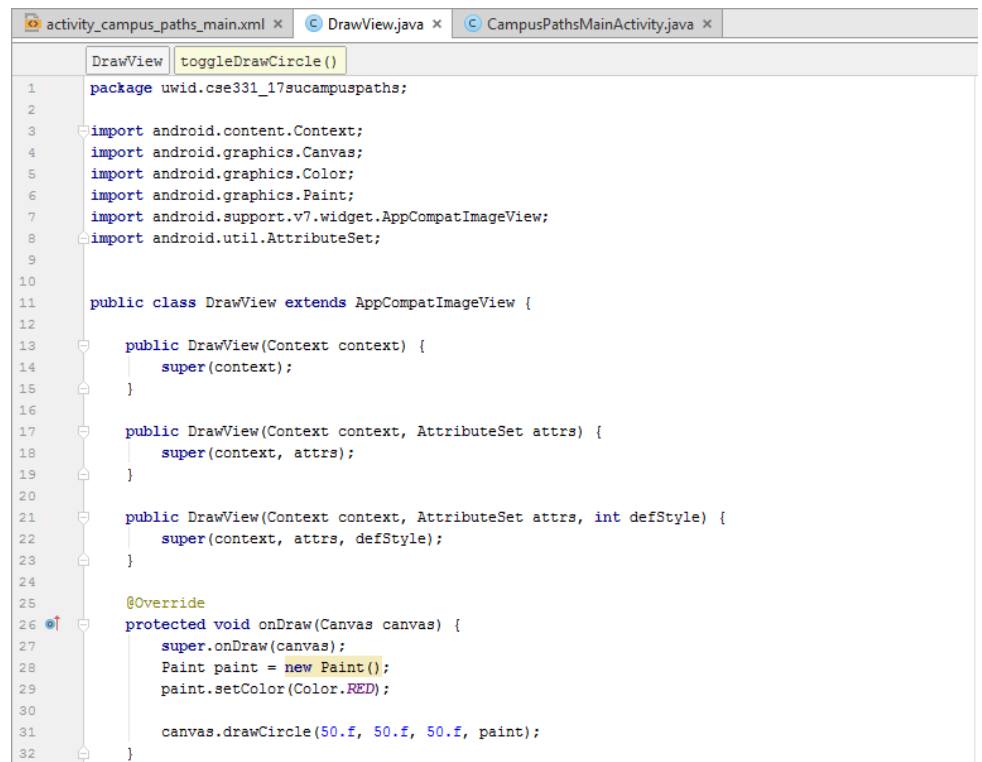
```
activity_campus_paths_main.xml x DrawView.java x CampusPathsMainActivity.java x
DrawView onDraw()
1 import android.content.Context;
2 import android.graphics.Canvas;
3 import android.graphics.Paint;
4 import android.support.v7.widget.AppCompatImageView;
5 import android.util.AttributeSet;
6
7
8 public class DrawView extends AppCompatImageView {
9
10     public DrawView(Context context) {
11         super(context);
12     }
13
14     public DrawView(Context context, AttributeSet attrs) {
15         super(context, attrs);
16     }
17
18     public DrawView(Context context, AttributeSet attrs, int defStyle) {
19         super(context, attrs, defStyle);
20     }
21
22     @Override
23     protected void onDraw(Canvas canvas) {
24         super.onDraw(canvas);
25         Paint paint = new Paint();
26     }
27
28 }
```

You can set the color and stroke width of your “Paint” by invoking the methods “setStrokeWidth” and “setColor” on your new Paint object. These are particularly helpful for painting paths between two landmarks on campus (and somewhat less useful for painting Muffin) because you’ll want to draw lines later on the canvas and have them appear a reasonable size and a distinguishable color. For now though, to draw a circle all you’ll need to set is the paint color. Choose a bright color like red by accessing the Color constant “Color.RED”

```
activity_campus_paths_main.xml x DrawView.java x CampusPathsMainActivity.java x
DrawView onDraw()
1 import android.content.Context;
2 import android.graphics.Canvas;
3 import android.graphics.Color;
4 import android.graphics.Paint;
5 import android.support.v7.widget.AppCompatImageView;
6 import android.util.AttributeSet;
7
8
9 public class DrawView extends AppCompatImageView {
10
11     public DrawView(Context context) {
12         super(context);
13     }
14
15     public DrawView(Context context, AttributeSet attrs) {
16         super(context, attrs);
17     }
18
19     public DrawView(Context context, AttributeSet attrs, int defStyle) {
20         super(context, attrs, defStyle);
21     }
22
23     @Override
24     protected void onDraw(Canvas canvas) {
25         super.onDraw(canvas);
26         Paint paint = new Paint();
27         paint.setColor(Color.RED);
28     }
29
30 }
```

Now, using the “Canvas” parameter passed in as an argument to the “onDraw” method, invoke the method “drawCircle” and provide some arbitrary positive coordinates to offset it from the top-left corner of the ImageView. An example of this is shown below:

*Note that Android’s drawing methods via the “Canvas” object often take in float values as arguments as opposed to doubles. If you used Double to represent path weights/costs in your Campus Paths project, simply call the method “floatValue()” on any Double (wrapper class) to get its value as a float.*



```
1 package uwid.cse331_17sucampuspaths;
2
3 import android.content.Context;
4 import android.graphics.Canvas;
5 import android.graphics.Color;
6 import android.graphics.Paint;
7 import android.support.v7.widget.AppCompatActivity;
8 import android.util.AttributeSet;
9
10
11 public class DrawView extends AppCompatActivity {
12
13     public DrawView(Context context) {
14         super(context);
15     }
16
17     public DrawView(Context context, AttributeSet attrs) {
18         super(context, attrs);
19     }
20
21     public DrawView(Context context, AttributeSet attrs, int defStyleAttr) {
22         super(context, attrs, defStyleAttr);
23     }
24
25     @Override
26     protected void onDraw(Canvas canvas) {
27         super.onDraw(canvas);
28         Paint paint = new Paint();
29         paint.setColor(Color.RED);
30
31         canvas.drawCircle(50.f, 50.f, 50.f, paint);
32     }
33 }
```

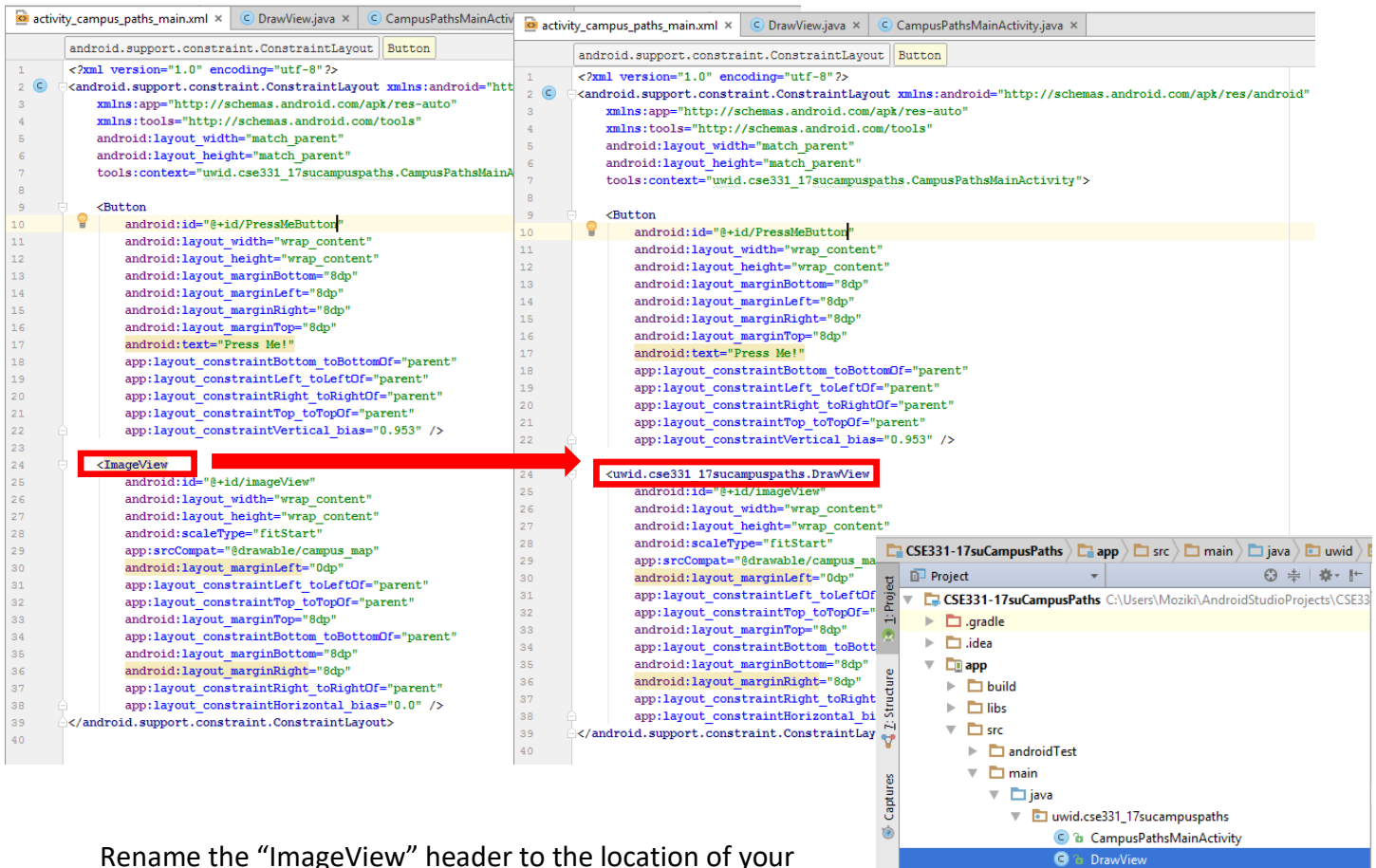
The first argument passed to “drawCircle” indicates the ‘x’ coordinate in the Cartesian plane of the center of the circle. The second argument passed indicates the ‘y’ coordinate of this Cartesian pair corresponding to the circle’s center. The third argument represents the radius of the circle.

By supplying the paint object we created with our individual settings, we’re able to indicate the color of the circle we want to appear.

Since “onDraw” is invoked when the application starts, this circle should appear right away when launching the emulator. However, as of right now, your ImageView is not being seen as the child class you implemented.



To change what type of ImageView the campus map image is, navigate to your Main Activity xml file and open the “Text” editor layout.



Rename the “ImageView” header to the location of your new class – this includes the name of the directory which contains the Java class you created.

*If when tabbing back to the “Design” layout for your application, the campus map image no longer appears, simply rebuild your project (and tab between “Text” and “Design” view) to make the image appear again.*

With that completed, your ImageView should now function as the newly implemented child class of ImageView!

Run the emulator to ensure the circle is drawn over the map inside the application!

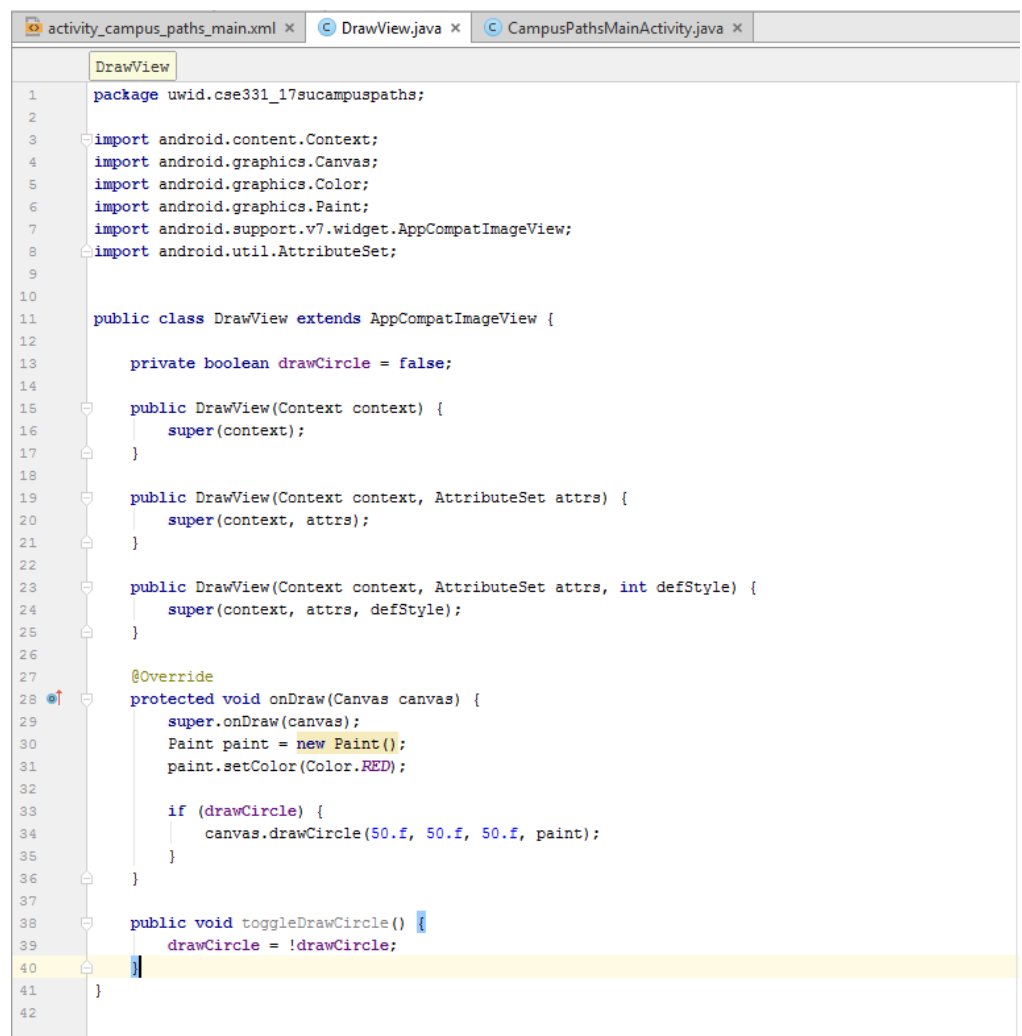
In order to make your application more interesting and interactive at this point, you'll want to pair the functionality of the Button you created previously with a change in what is drawn on campus map!

To do this, you'll want to update some sort of value in the DrawView code which triggers a change in the red circle appearing. In the example, I'll use a simple Boolean value, but this could be anything like an object becoming instantiated (no longer null), a random value updating the color every press, and much more.

Add a Boolean field to your DrawView called "DrawCircle" – for the sake of this example, make the field private so you'll have to change the Boolean value by invoking a method from the DrawView.

After creating the field, add a new public method called "toggleDrawCircle" similar to the example shown below.

Then, inside the "onDraw" method, change the implementation so we only draw a circle now when "DrawCircle" is true.



```
1 package uwid.cse331_17sucampuspaths;
2
3 import android.content.Context;
4 import android.graphics.Canvas;
5 import android.graphics.Color;
6 import android.graphics.Paint;
7 import android.support.v7.widget.AppCompatActivity;
8 import android.util.AttributeSet;
9
10
11 public class DrawView extends AppCompatActivity {
12
13     private boolean drawCircle = false;
14
15     public DrawView(Context context) {
16         super(context);
17     }
18
19     public DrawView(Context context, AttributeSet attrs) {
20         super(context, attrs);
21     }
22
23     public DrawView(Context context, AttributeSet attrs, int defStyle) {
24         super(context, attrs, defStyle);
25     }
26
27     @Override
28     protected void onDraw(Canvas canvas) {
29         super.onDraw(canvas);
30         Paint paint = new Paint();
31         paint.setColor(Color.RED);
32
33         if (drawCircle) {
34             canvas.drawCircle(50.f, 50.f, 50.f, paint);
35         }
36     }
37
38     public void toggleDrawCircle() {
39         drawCircle = !drawCircle;
40     }
41 }
42
```

Now, back in our Main Activity Java file (CampusPathsMainActivity.java), you'll want to change your Button listener to instead of printing "Toast" text to the screen, calling the "toggleDrawCircle" method instead.

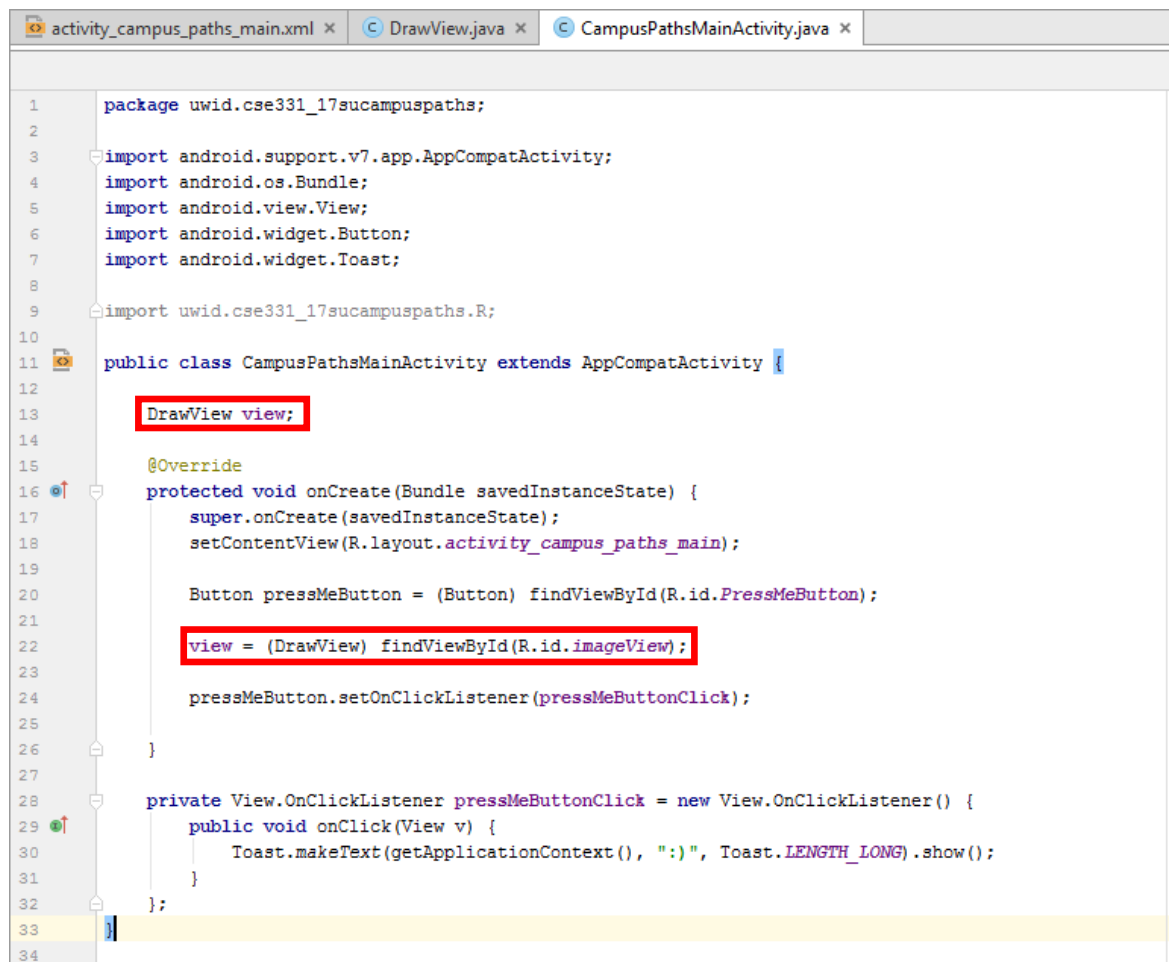
You'll notice though, you currently have no way to reach the DrawView you want in your Main Activity Java file. In order to create a reference to your view, you'll have to do an ID lookup similar to how you found the Button by its ID previously.

Use the method "findViewById" and supply "R.id.imageView" (or however you named your ImageView in the Main Activity xml file if you changed it) as an argument.

*The default ID for an ImageView is "imageView" however, feel free to change its ID to anything else.*

Cast the result to a "DrawView" and store it in a field similar to the way shown below:

*Feel free to remove the line that creates the "Toast" text as well!*



```
1 package uwid.cse331_17sucampuspaths;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.view.View;
6 import android.widget.Button;
7 import android.widget.Toast;
8
9 import uwid.cse331_17sucampuspaths.R;
10
11 public class CampusPathsMainActivity extends AppCompatActivity {
12     DrawView view;
13
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_campus_paths_main);
19
20         Button pressMeButton = (Button) findViewById(R.id.PressMeButton);
21
22         view = (DrawView) findViewById(R.id.imageView);
23
24         pressMeButton.setOnClickListener(pressMeButtonClick);
25
26     }
27
28     private View.OnClickListener pressMeButtonClick = new View.OnClickListener() {
29         public void onClick(View v) {
30             Toast.makeText(getApplicationContext(), ":", Toast.LENGTH_LONG).show();
31         }
32     };
33
34 }
```

Now with a reference to the DrawView, you're able to invoke the method which toggles the drawn circle on and off through the Button listener. However, in order to update the image correctly, you'll have to call the "invalidate" method on the DrawView as well. The "invalidate" method signals to the view that it needs to be redrawn, which in the specification of "invalidate" states that it will eventually call "onDraw" after "invalidate" has been invoked.

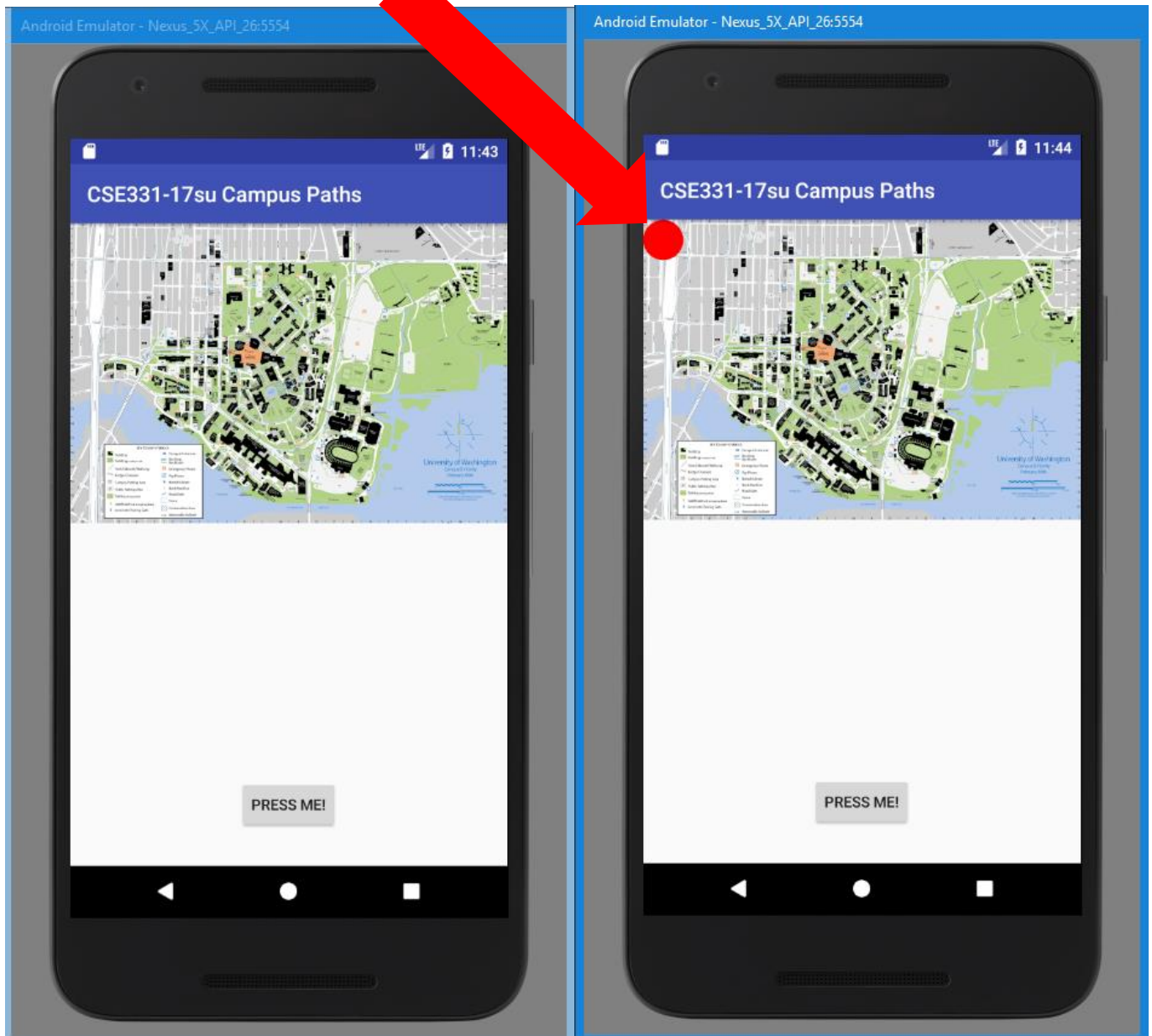
You can either invalidate the DrawView from the Main Activity or inside the DrawView itself (i.e. in the "toggleDrawCircle" method). Personally, I prefer invalidating the DrawView inside the DrawView as it separates functionality as the Main Activity is not responsible for updating what is drawn in the view.

Inside the Button listener, invoke the "toggleDrawCircle" method of the DrawView. Additionally, inside the "DrawView" class, add the line "this.invalidate()" to the end of the "toggleDrawCircle" method shown below.

*Alternatively, add "view.invalidate()" to the end of the Button listener "onClick" method.*

```
activity_campus_paths_main.xml x DrawView.java x CampusPathsMainActivity.java x
CampusPathsMainActivity pressMeButtonClick new OnClickListener onClick()
1 package uwid.cse331_17sucampuspaths;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.view.View;
6 import android.widget.Button;
7 import android.widget.Toast;
8
9 import uwid.cse331_17sucampuspaths.R;
10
11 public class CampusPathsMainActivity extends AppCompatActivity {
12
13     DrawView view;
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_campus_paths_main);
19
20         Button pressMeButton = (Button) findViewById(R.id.PressMeButton);
21
22         view = (DrawView) findViewById(R.id.imageView);
23
24         pressMeButton.setOnClickListener(pressMeButtonClick);
25
26     }
27
28     private View.OnClickListener pressMeButtonClick = new View.OnClickListener() {
29         public void onClick(View v) {
30             view.toggleDrawCircle();
31         }
32     };
33
34 }
30 Paint paint = new Paint();
31 paint.setColor(Color.RED);
32
33 if (drawCircle) {
34     canvas.drawCircle(50.f, 50.f, 50.f, paint);
35 }
36
37
38 public void toggleDrawCircle() {
39     drawCircle = !drawCircle;
40     this.invalidate();
41 }
42
43 }
```

Fire up the emulator and test the application! The Button should now toggle the red circle drawn on top of the campus map!



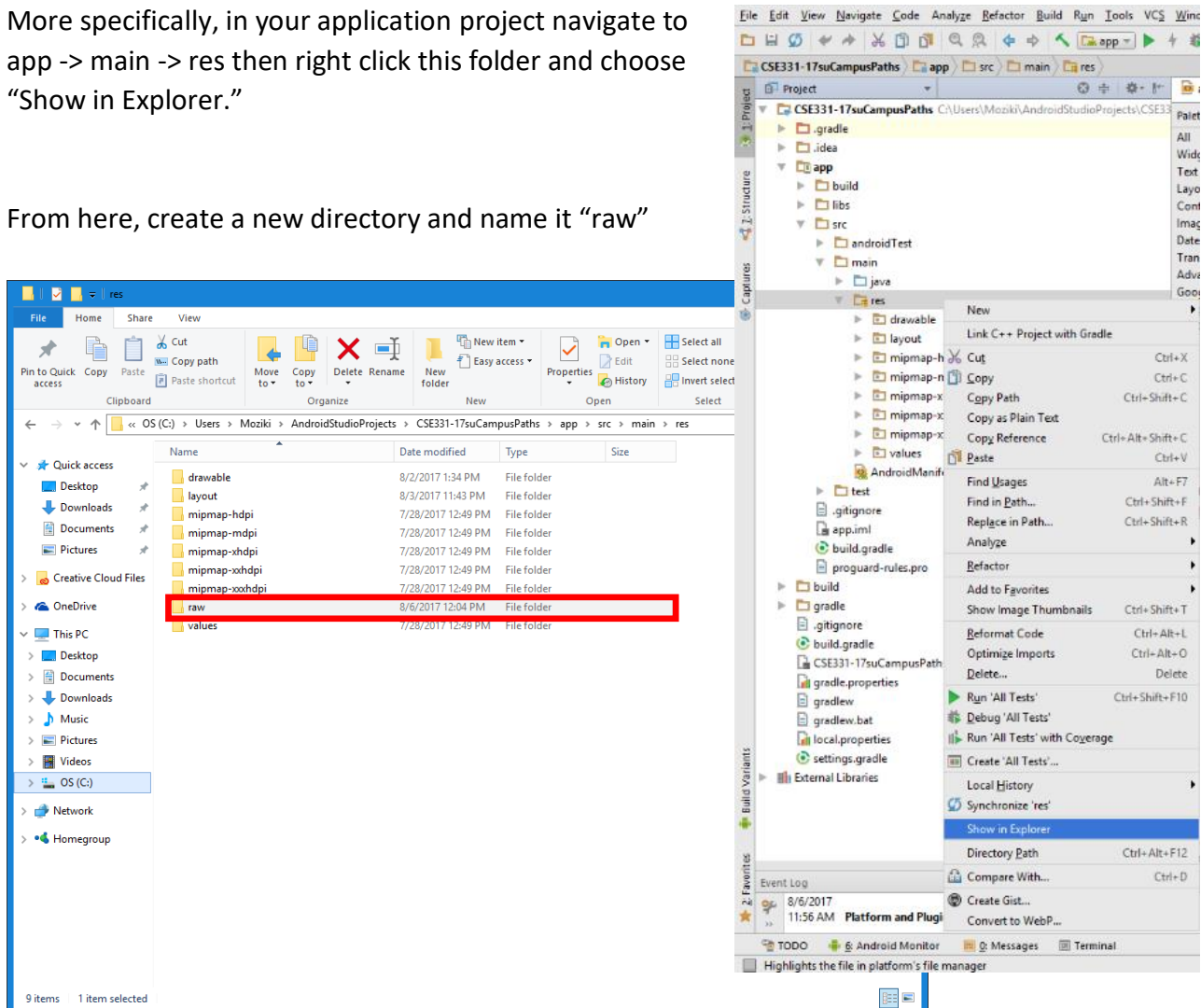
## 7. Loading and Storing Data in a ListView

Loading the provided data files in Android Studio is slightly different from opening a simple text file in Eclipse. Android Studio uses a method called “openRawResource” which provides an input stream for the stored file.

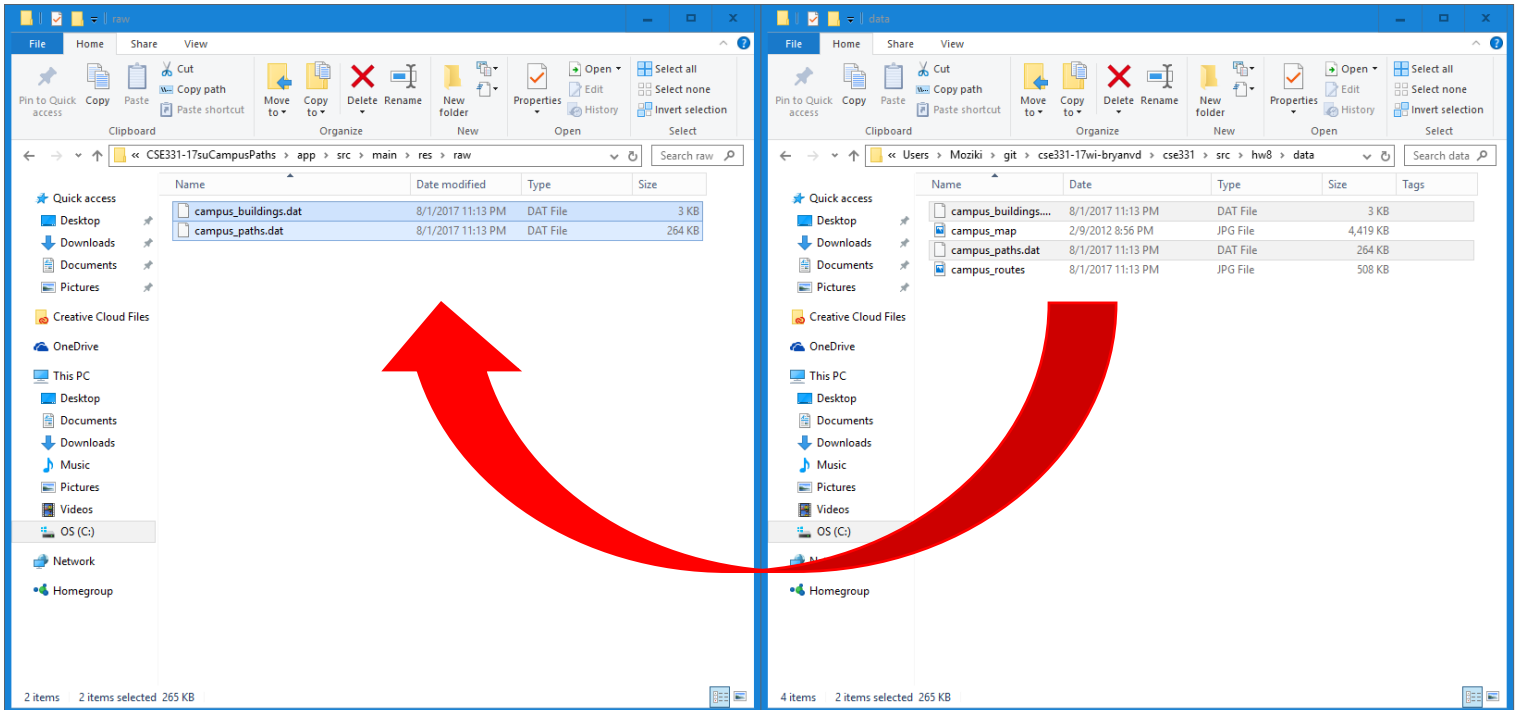
In order to use this method though, you’ll first have to create the “raw” folder to house the campus paths and building data files in the “res” folder of your project.

More specifically, in your application project navigate to app -> main -> res then right click this folder and choose “Show in Explorer.”

From here, create a new directory and name it “raw”



Navigate inside the newly created “raw” folder and copy “campus\_buildings.dat” and “campus\_paths.dat” from your Eclipse “hw8/data” folder to the “raw” directory in your Android Studio project.



With the data files now in your Android Studio project “raw” folder, you’ll now be able to use the “openRawResource” method directed to “R.raw.campus\_buildings.dat” and “R.raw.campus\_paths.dat”

Reference the adjacent code to create two new InputStreams to be able to use the campus building and path information (be sure to import the necessary “java.io.InputStream” as well).

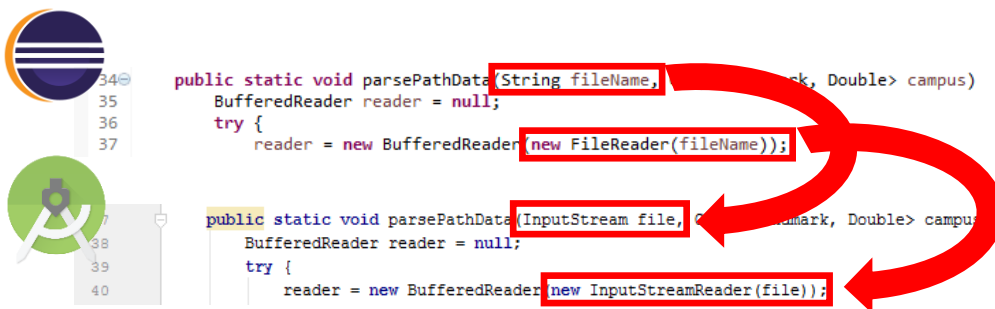
```

activity_campus_paths_main.xml x  C DrawView.java x  C CampusPathsMainActivity.java x
CampusPathsMainActivity  onCreate ()
1  package uid.cse331_17sucampuspaths;
2
3  import android.support.v7.app.AppCompatActivity;
4  import android.os.Bundle;
5  import android.view.View;
6  import android.widget.Button;
7  import android.widget.Toast;
8
9  import java.io.InputStream;
10
11  import uid.cse331_17sucampuspaths.R;
12
13  public class CampusPathsMainActivity extends AppCompatActivity {
14
15      DrawView view;
16
17      @Override
18      protected void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20          setContentView(R.layout.activity_campus_paths_main);
21
22          InputStream pathsInputStream = this.getResources().openRawResource(R.raw.campus_paths);
23          InputStream buildingsInputStream = this.getResources().openRawResource(R.raw.campus_buildings);
24
25
26          Button pressMeButton = (Button) findViewById(R.id.PressMeButton);
27
28          view = (DrawView) findViewById(R.id.imageView);
29
30          pressMeButton.setOnClickListener(pressMeButtonClick);
31
32      }
33
34      private View.OnClickListener pressMeButtonClick = new View.OnClickListener() {
35          public void onClick(View v) {
36              view.toggleDrawCircle();
37          }
38      };
39
40  }

```

Using an `InputStream` may differ from how you implemented your `Campus Parser` in homework 8. In order to get the data to load in `Android Studio`, you may have to adapt this implementation slightly. If you followed a similar parsing structure based off of the provided `MarvelParser` in previous homework assignments, you should still be able to use a `BufferedReader` to parse the data, however, in the constructor instead of using a `FileReader` and a `String` for the file name, simply supply a new `InputStreamReader` taking in the created `InputStream` as a parameter to the `BufferedReader`. Note that since you created a `Jar` to hold your previous project implementations, you may have to recreate the `Jar` file with any updates you make. Alternatively, copying the previous `campus parsing code` to a new `Java class` built for `Android Studio` (with some tweaks to accommodate the new environment) is acceptable as well.

This difference is shown below (if you did not use a `BufferedReader` in your parser, you may have to adapt your `Campus Parser` through other means to accommodate an `InputStream` now).

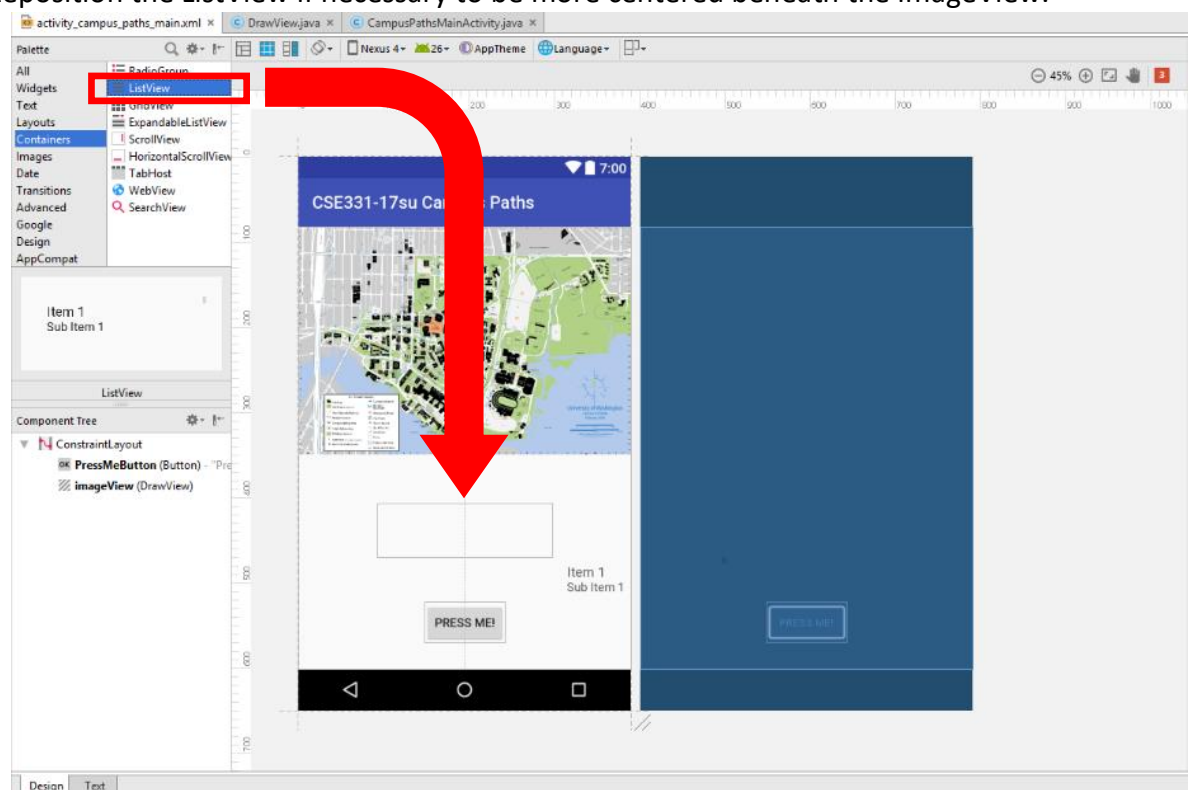


```
34 public static void parsePathData(String fileName, (String fileName, (String fileName, Double> campus)
35     BufferedReader reader = null;
36     try {
37         reader = new BufferedReader(new FileReader(fileName));

public static void parsePathData(InputStream file, (InputStream file, (InputStream file, Double> campus)
38     BufferedReader reader = null;
39     try {
40         reader = new BufferedReader(new InputStreamReader(file));
```

The image shows two code snippets. The first snippet uses `new FileReader(fileName)` to create a `BufferedReader`. The second snippet uses `new InputStreamReader(file)` to create a `BufferedReader`. Red boxes highlight the constructor arguments, and red arrows point from the `FileReader` constructor to the `InputStreamReader` constructor, indicating the adaptation needed for `InputStream`.

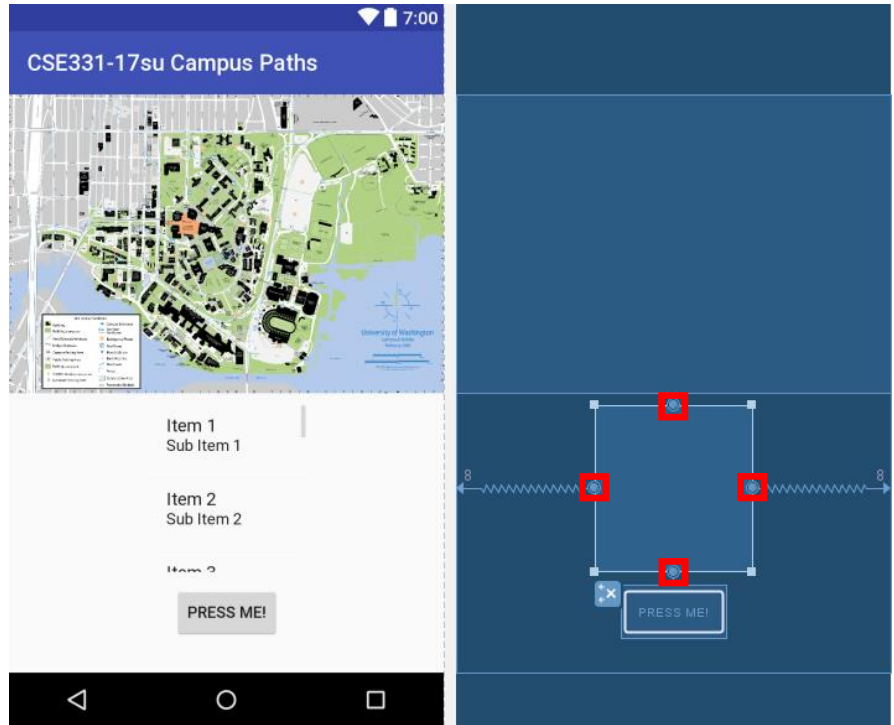
With the data files input and ready to be parsed, you'll need to add a `ListView` to your application design layout. In your `Main Activity xml` file, drag and drop a `ListView` into the application. Reposition the `ListView` if necessary to be more centered beneath the `ImageView`.





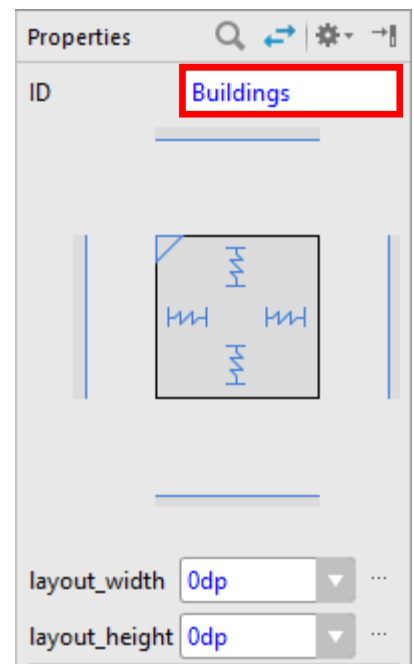
Anchor the ListView to the underside of the ImageView, the top of the Button, and the sides of the application similar to how is shown below.

*For future reference, it may be useful to insert a "Space" element into your application, anchor the "Space" to the bottom of the ImageView, then, anchor subsequent components to this "Space" – I found this helps with aligning components once the application launches in the emulator (i.e. if you had two parallel ListViews for separate data).*



Rename the ListView to something fitting like "Buildings" by editing the ID in the "Properties" tab with the ListView selected or in the Main Activity xml file text layout.

In the following steps, you'll have to configure your reference to your ListView using an ArrayAdapter to easily add items/elements.



Start by creating a new field variable and finding the reference to it by using the “findViewById” method – make sure to cast the result to a “ListView.”

To create a new ArrayAdapter, you’ll have to provide a number of parameters in the constructor: the application context, a layout style, and an ArrayList to hold the items and their order in how you want them to appear in the list. This turns out to be fairly trivial – the easiest way to accomplish this is shown in the example below (be sure to import any necessary classes).

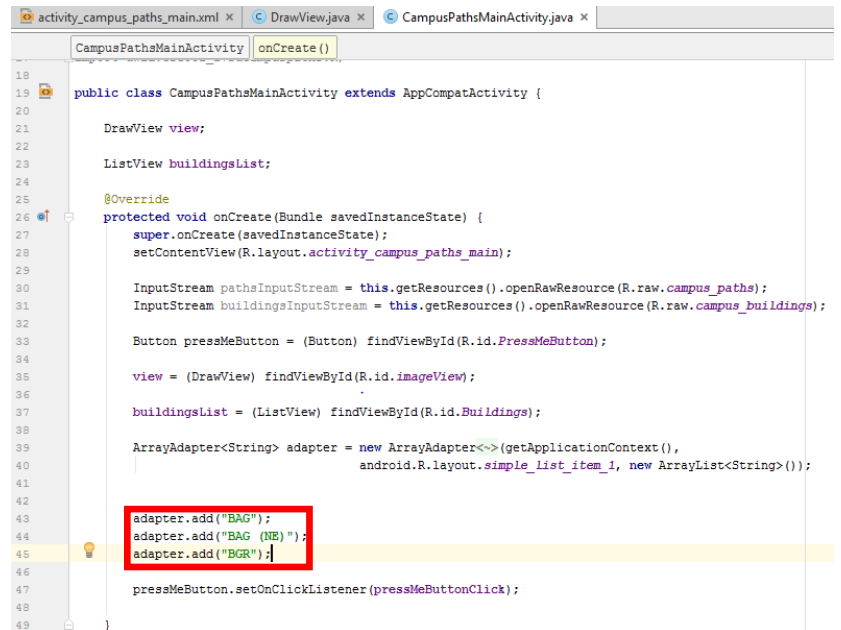
```
activity_campus_paths_main.xml x DrawView.java x CampusPathsMainActivity.java x
CampusPathsMainActivity onCreate()
1 package uwid.cse331_17sucampuspaths;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.view.View;
6 import android.widget.Button;
7 import android.widget.ListView;
8 import java.io.InputStream;
9 import uwid.cse331_17sucampuspaths.R;
10
11 public class CampusPathsMainActivity extends AppCompatActivity {
12
13     DrawView view;
14
15     ListView buildingsList;
16
17     @Override
18     protected void onCreate(Bundle savedInstanceState) {
19         super.onCreate(savedInstanceState);
20         setContentView(R.layout.activity_campus_paths_main);
21
22         InputStream pathsInputStream = this.getResources().openRawResource(R.raw.campus_paths);
23         InputStream buildingsInputStream = this.getResources().openRawResource(R.raw.campus_buildings);
24
25         Button pressMeButton = (Button) findViewById(R.id.PressMeButton);
26
27         view = (DrawView) findViewById(R.id.imageView);
28
29         buildingsList = (ListView) findViewById(R.id.Buildings);
30
31         pressMeButton.setOnClickListener(pressMeButtonClick);
32
33     }
34
35     private View.OnClickListener pressMeButtonClick = new View.OnClickListener() {
36         public void onClick(View v) {
37             view.toggleDrawCircle();
38         }
39     };
40 }
```

```
activity_campus_paths_main.xml x DrawView.java x CampusPathsMainActivity.java x
CampusPathsMainActivity onCreate()
4 import android.os.Bundle;
5 import android.view.View;
6 import android.widget.ArrayAdapter;
7 import android.widget.Button;
8 import android.widget.ListView;
9 import java.io.InputStream;
10 import java.util.ArrayList;
11
12 import uwid.cse331_17sucampuspaths.R;
13
14 public class CampusPathsMainActivity extends AppCompatActivity {
15
16     DrawView view;
17
18     ListView buildingsList;
19
20     @Override
21     protected void onCreate(Bundle savedInstanceState) {
22         super.onCreate(savedInstanceState);
23         setContentView(R.layout.activity_campus_paths_main);
24
25         InputStream pathsInputStream = this.getResources().openRawResource(R.raw.campus_paths);
26         InputStream buildingsInputStream = this.getResources().openRawResource(R.raw.campus_buildings);
27
28         Button pressMeButton = (Button) findViewById(R.id.PressMeButton);
29
30         view = (DrawView) findViewById(R.id.imageView);
31
32         buildingsList = (ListView) findViewById(R.id.Buildings);
33
34         ArrayAdapter<String> adapter = new ArrayAdapter<String>(getApplicationContext(), android.R.layout.simple_list_item_1, new ArrayList<String>());
35
36         pressMeButton.setOnClickListener(pressMeButtonClick);
37
38     }
39
40     private View.OnClickListener pressMeButtonClick = new View.OnClickListener() {
41         public void onClick(View v) {
42             view.toggleDrawCircle();
43         }
44     };
45 }
```

Note that you can experiment with other layout styles by looking at the autocompleted suggestions after “android.R.layout.”

Since the ArrayAdapter instantiation is a fairly long line of code, I spread it out onto multiple lines for the remainder of this section. Note you could arguably (and likely should) delegate the ArrayAdapter configuration to a separate method, but for the purposes of this guide, I will leave it in the “onCreate” method.

How you ultimately handle preparing your building data to a form which can be used for the ListView depends entirely on your implementation – I suggest using the ‘short names’ of the buildings as they will fit far more comfortably in the limited screen space. For now, I’ll be adding a couple static building names – “BAG,” “BAG (NE),” and “BGR” by calling the “add” method on the adapter.



```
18
19
20
21 DrawView view;
22
23 ListView buildingsList;
24
25
26 @Override
27 protected void onCreate(Bundle savedInstanceState) {
28     super.onCreate(savedInstanceState);
29     setContentView(R.layout.activity_campus_paths_main);
30
31     InputStream pathsInputStream = this.getResources().openRawResource(R.raw.campus_paths);
32     InputStream buildingsInputStream = this.getResources().openRawResource(R.raw.campus_buildings);
33
34     Button pressMeButton = (Button) findViewById(R.id.PressMeButton);
35
36     view = (DrawView) findViewById(R.id.imageView);
37
38     buildingsList = (ListView) findViewById(R.id.Buildings);
39
40     ArrayAdapter<String> adapter = new ArrayAdapter<>(getApplicationContext(),
41         android.R.layout.simple_list_item_1, new ArrayList<String>());
42
43     adapter.add("BAG");
44     adapter.add("BAG (NE)");
45     adapter.add("BGR");
46
47     pressMeButton.setOnClickListener(pressMeButtonClick);
48
49 }
```

After adding the elements to the adapter, you’ll want to pair this adapter with the ListView invoking the “setAdapter” method on the reference you created previously.



```
17 import uwid.cse331_17sucampuspaths.R;
18
19 public class CampusPathsMainActivity extends AppCompatActivity {
20
21     DrawView view;
22
23     ListView buildingsList;
24
25
26 @Override
27 protected void onCreate(Bundle savedInstanceState) {
28     super.onCreate(savedInstanceState);
29     setContentView(R.layout.activity_campus_paths_main);
30
31     InputStream pathsInputStream = this.getResources().openRawResource(R.raw.campus_paths);
32     InputStream buildingsInputStream = this.getResources().openRawResource(R.raw.campus_buildings);
33
34     Button pressMeButton = (Button) findViewById(R.id.PressMeButton);
35
36     view = (DrawView) findViewById(R.id.imageView);
37
38     buildingsList = (ListView) findViewById(R.id.Buildings);
39
40     ArrayAdapter<String> adapter = new ArrayAdapter<>(getApplicationContext(),
41         android.R.layout.simple_list_item_1, new ArrayList<String>());
42
43     adapter.add("BAG");
44     adapter.add("BAG (NE)");
45     adapter.add("BGR");
46
47     buildingsList.setAdapter(adapter);
48
49     pressMeButton.setOnClickListener(pressMeButtonClick);
50
51
52 }
```

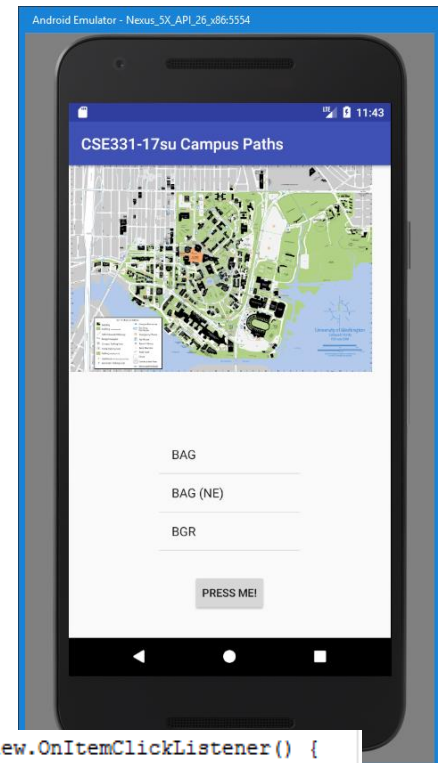
You can try running your emulator and see that the new ListView appears in your application! However, when selecting any of the items in the list, nothing happens. The next step is add a listener and callback functionality to your ListView so that you can use the value selected by the user.

To do so, you'll want to create a similar listener to when you implemented the Button in the previous sections.

However, this time, you'll want to implement a "ListView.OnItemClickListener" and override the "onItemClick" method in an anonymous inner class in your Main Activity Java file.

Create the listener similar to the one shown below.

```
70 private ListView.OnItemClickListener listViewItemClick = new ListView.OnItemClickListener() {
71     public void onItemClick(AdapterView<?> adapter, View v, int position, long id) {
72         String buildingShortName = (String) buildingsList.getItemAtPosition(position);
73         Toast.makeText(getApplicationContext(), buildingShortName, Toast.LENGTH_LONG).show();
74     }
75 };
76
77 private View.OnClickListener pressMeButtonClick = new View.OnClickListener() {
78     public void onClick(View v) {
79         view.toggleDrawCircle();
80     }
81 };
82 }
```



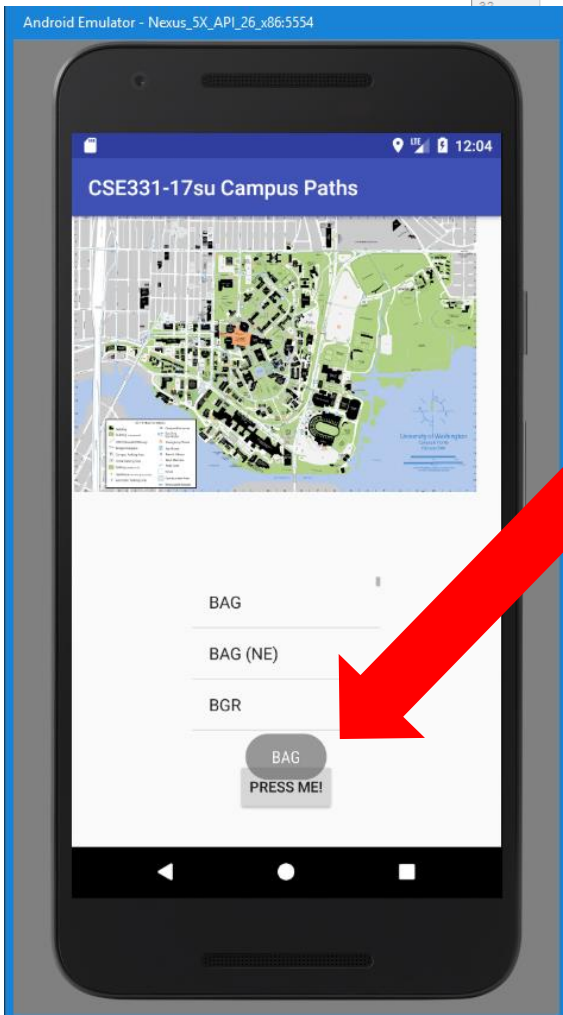
There are a few key differences to note between the Button listener and the ListView listener in this example. Besides the differing method and class declarations, the overridden method in the body of the class, "onItemClick" takes far more parameters. The 'adapter' is the adapter currently associated with the ListView (like the one you just created – notice the use of a wild card here too you should recognize this now! Since we created our adapter to be used with Strings, but more generally, it could be an adapter for a number of different classes, this listener does not necessarily know the type used in the adapter for the current ListView. Therefore, a wild card is used to make the method compatible for all types of adapters! Pretty neat if I do say so myself :)! Furthermore, the 'position' integer corresponds to the position of the item selected in the list that was clicked. This can be easily used in the "getItemAtPosition" method from the ListView to get the information about the element that you want.

I simply output the retrieved information from the ListView to the “Toast” text on screen (remembering to cast the retrieved result to the type which we know that it is). However, you can do far more with this information in your own project. By storing the selected item in a field or other variable, you can use the building name to find the start or destination for a path between the two on campus!

The last thing to do is to set the “onItemClickListener” to the ListView similar to how you set up the listener for the Button in the “onCreate” method of your Main Activity Java file.

Run your application and press on any item in the ListView to see how the callback functionality now works!

```
activity_campus_paths_main.xml | DrawView.java | CampusPathsMainActivity.java |
CampusPathsMainActivity | onCreate()
17 import uwid.cse331_17sucampuspaths.R;
18
19 public class CampusPathsMainActivity extends AppCompatActivity {
20
21     DrawView view;
22
23     ListView buildingsList;
24
25     @Override
26     protected void onCreate(Bundle savedInstanceState) {
27         super.onCreate(savedInstanceState);
28         setContentView(R.layout.activity_campus_paths_main);
29
30         InputStream pathsInputStream = this.getResources().openRawResource(R.raw.campus_paths);
31         InputStream buildingsInputStream = this.getResources().openRawResource(R.raw.campus_buildings);
32
33         Button pressMeButton = (Button) findViewById(R.id.PressMeButton);
34
35         view = (DrawView) findViewById(R.id.imageView);
36
37         buildingsList = (ListView) findViewById(R.id.Buildings);
38
39         ArrayAdapter<String> adapter = new ArrayAdapter<>(getApplicationContext(),
40             android.R.layout.simple_list_item_1, new ArrayList<String>());
41
42         adapter.add("BAG");
43         adapter.add("BAG (NE)");
44         adapter.add("BGR");
45
46         buildingsList.setAdapter(adapter);
47         buildingsList.setOnItemClickListener(listViewItemClick);
48
49         pressMeButton.setOnClickListener(pressMeButtonClick);
50     }
51 }
```



## 8. Conclusion and Helpful Hints

When it comes to creating the actual Campus Paths project in Android, this layout should hopefully provide enough of a foundation to complete the assignment, however, the Android features and API have so much more to offer and explore – we have barely scratched the surface!

I would suggest if you're interested in pursuing other features you could include in your Android application, to simply scroll through the various widgets in the "Design" layout of your Main Activity xml file and search online for how you could go about implementing them for this project!

In terms of more relevant hints for this project, when it comes to drawing the path on the campus map, you will likely (almost definitely) have to scale down the coordinate values provided in the "campus\_paths.dat" file. Since the original campus map image had a much higher resolution, this means the coordinate points in the provided data file correspond to the original map which is not accurate for the one used in your application. When experimenting with drawing the path in the application, I noticed that scaling the coordinates to 0.22 or 0.23 of their original values lined up the paths nicely on the image. If you do not scale down the values, you may be very confused and frustrated as to why nothing is appearing on the map when invalidating the ImageView – you've been warned!

Remember to try to maintain the principles of MVC – Model, View, Controller. You'll want to keep these functionalities largely independent of one another if you're going to have a successful application. The Main Activity you created throughout this guide is very similar to the "Controller" aspect of this principle – hide the details from the other two from this class!

Lastly, I had a lot of fun creating this guide so it truly was an honor making it! I hope you found it helpful and useful. Good luck on the final project!