Mobile Monday Tech Lab 01

SOFTWARE CRAFTSMANSHIP GUILD

Android Development - Basics

Overview

In this lab we are going to create, code, and run a simple Android app. Along the way we will take a tour of the major components of an Android app and start to become familiar with the new Android Studio from Google. Our objectives are:

1. Become familiar with Android Studio
2. Understand how to create a new project in Android Studio
3. Understand the basic project structure of an Android app
4. Create a simple app and understand the following:
    a. How to create a layout with text input and a button
    b. How to read input from the user
    c. How to react to button clicks
    d. How to display feedback to the user with the 'toast' control
5. Understand how to execute an app on the emulator
6. Understand how to enable an Android phone for development
7. Understand how to load and run an app on an Android phone

Preliminaries

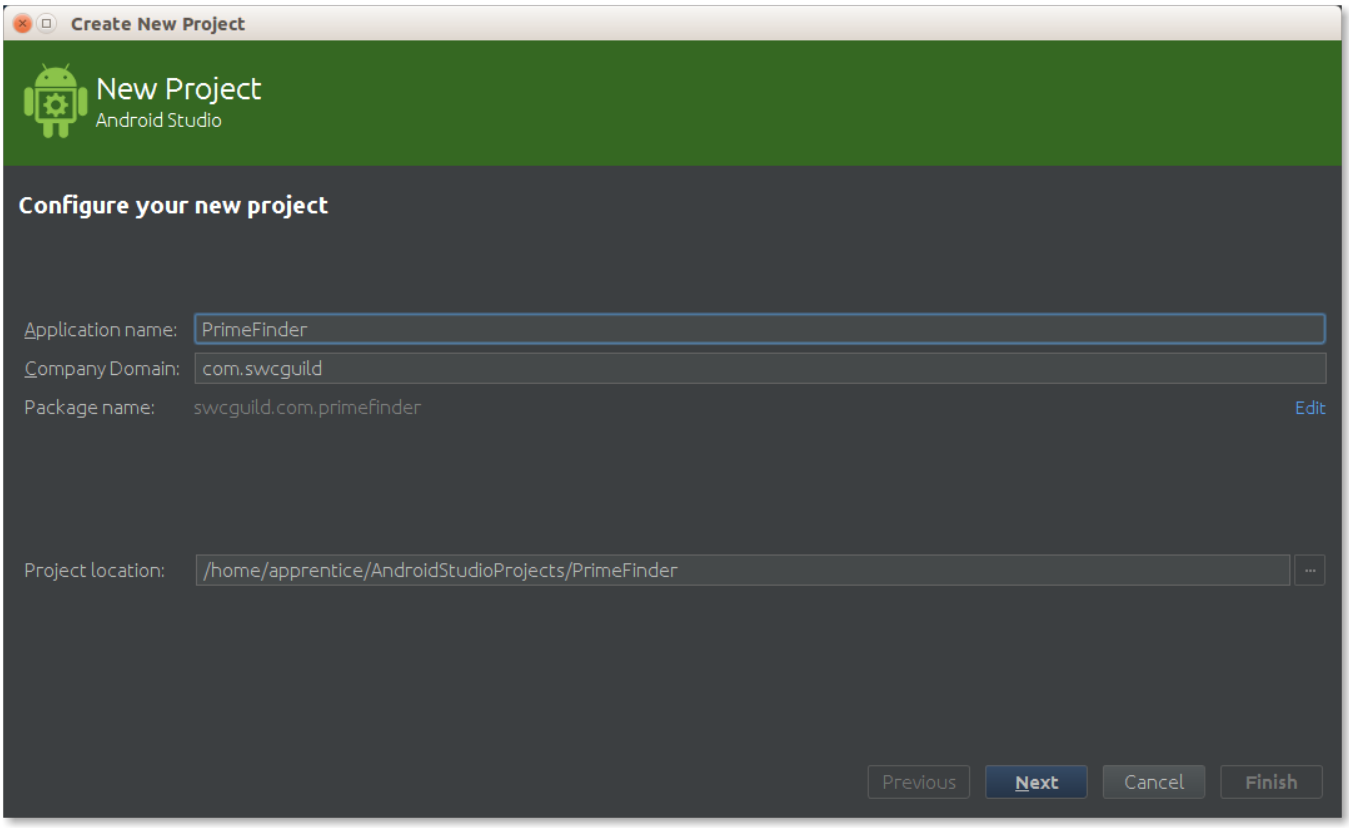
Before you can start developing an Android app, you must install the Java SDK (I recommend Java 8) and an Android Integrated Development Environment (IDE). You can develop Android apps using any of the three major Java IDEs - Eclipse, NetBeans, or IntelliJ Idea. In this series, we will be using Google's new Android Studio, which is based on IntelliJ Idea. At this point, it looks like Google is putting their efforts behind Android Studio which means it will most likely be the best supported option moving forward.

You can find directions for installing the Java SDK for your platform @ http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html.

The directions for installation of Android Studio can be found @ https://developer.android.com/sdk/installing/studio.html, Keep in mind that the Android development environment consists of Android Studio and the Android SDK. The link above contains instructions for configuring all parts of the environment.

First Steps - Project Creation

In this section we will create the initial shell of the project that we'll be developing in this lab. After we create the shell, we'll take a quick tour of the project.
Open Android Studio and click on the **File → New Project…** menu. You should see the following dialog:

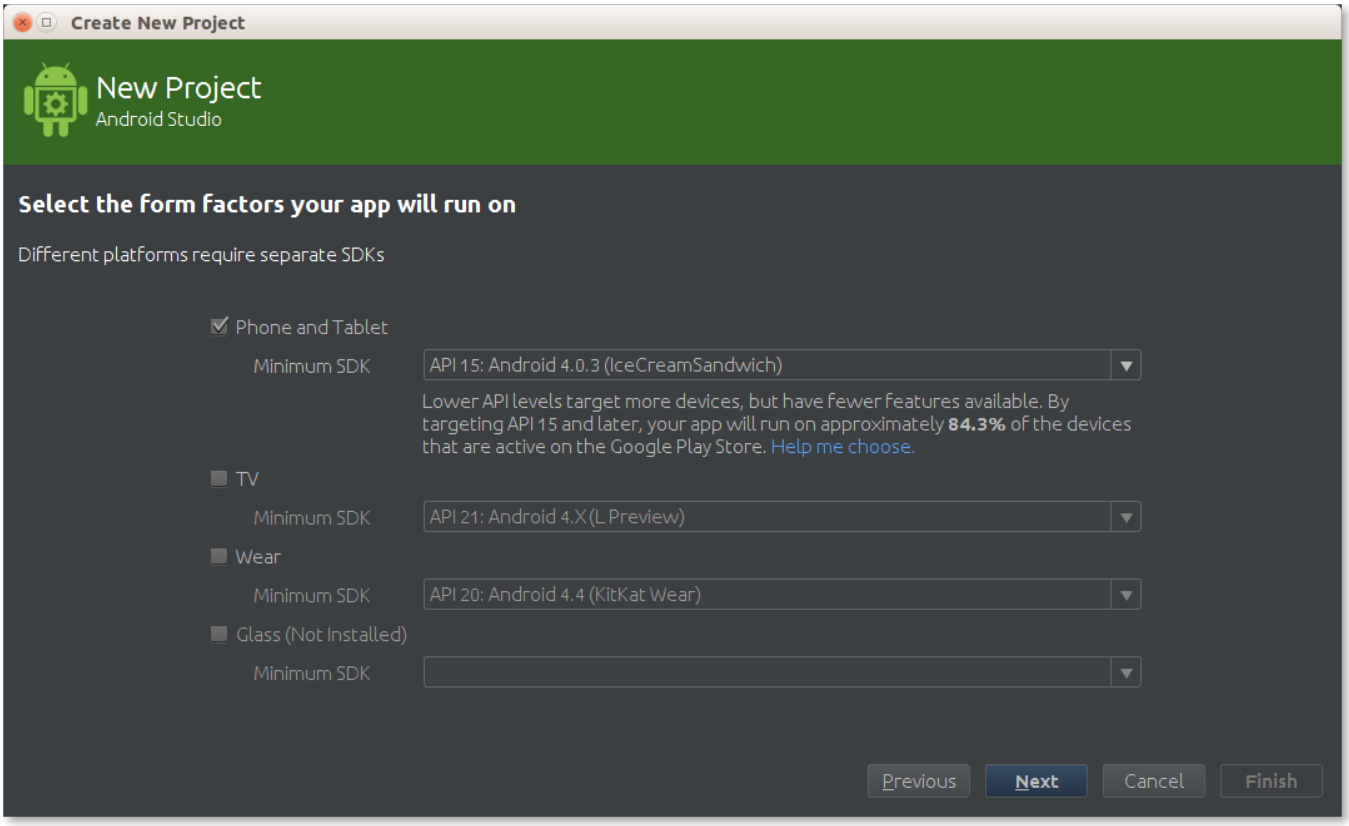


This dialog box allows you to name your application, set up your Java package names, and choose a location for the source code. Please fill in the values as shown above:

- Application name: PrimeFinder
- Company Domain: com.swcguild
- Project location: This will be specific to your environment

Click "Next"

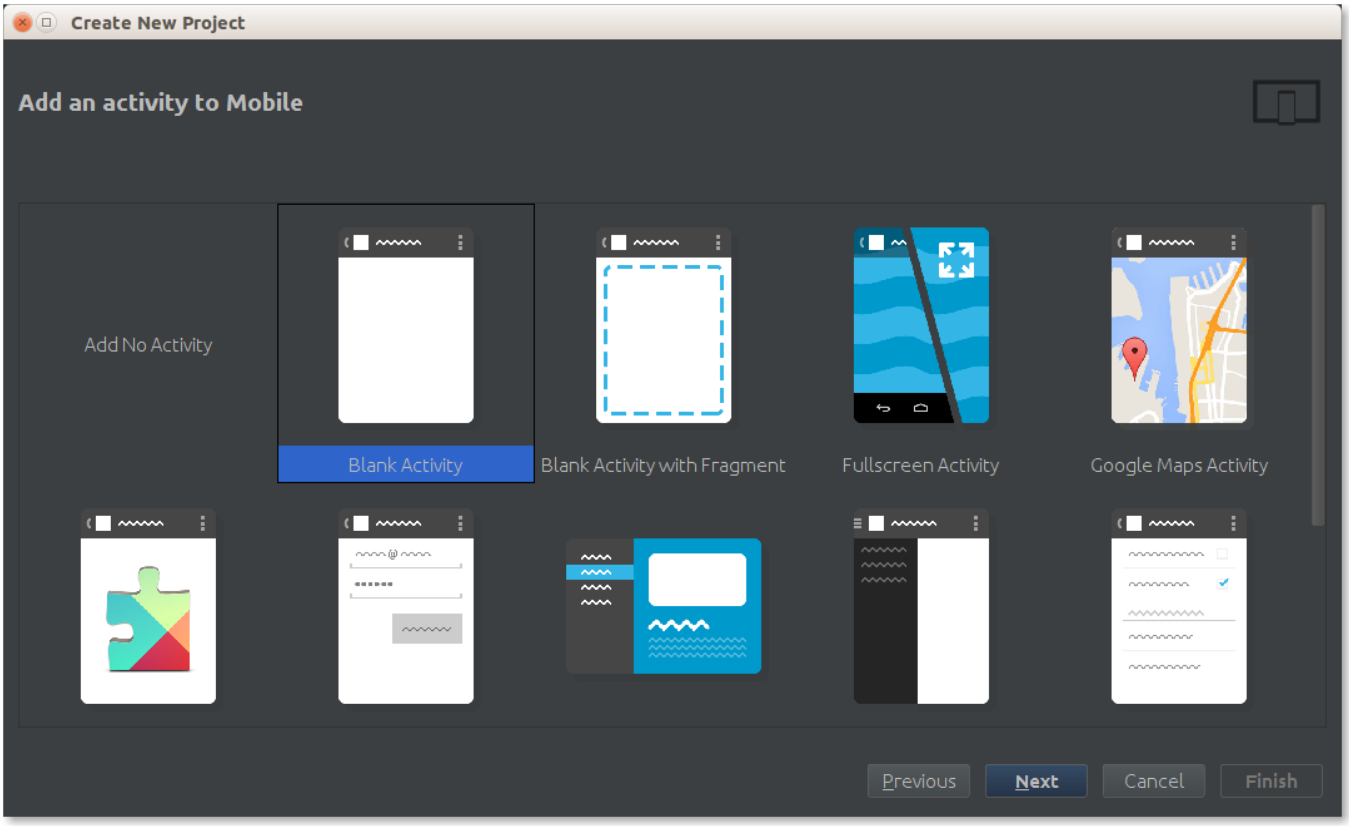
You should now see the following dialog:



This dialog allows you to choose the platforms on which your application will run. For these sessions we will target only Phone and Tablet. After choosing the form factor (Phone and Tablet) you must also choose the Minimum SDK version that you wish to support. The lower the version, the greater the number of devices you will be able to support. The tradeoff is that you will not be able to take advantage of the advanced features of more recent Android versions. The applications we will build for this series do not take advantage of any advanced features which is why we have chosen API 15 (Ice Cream Sandwich). You can choose whatever version you wish.

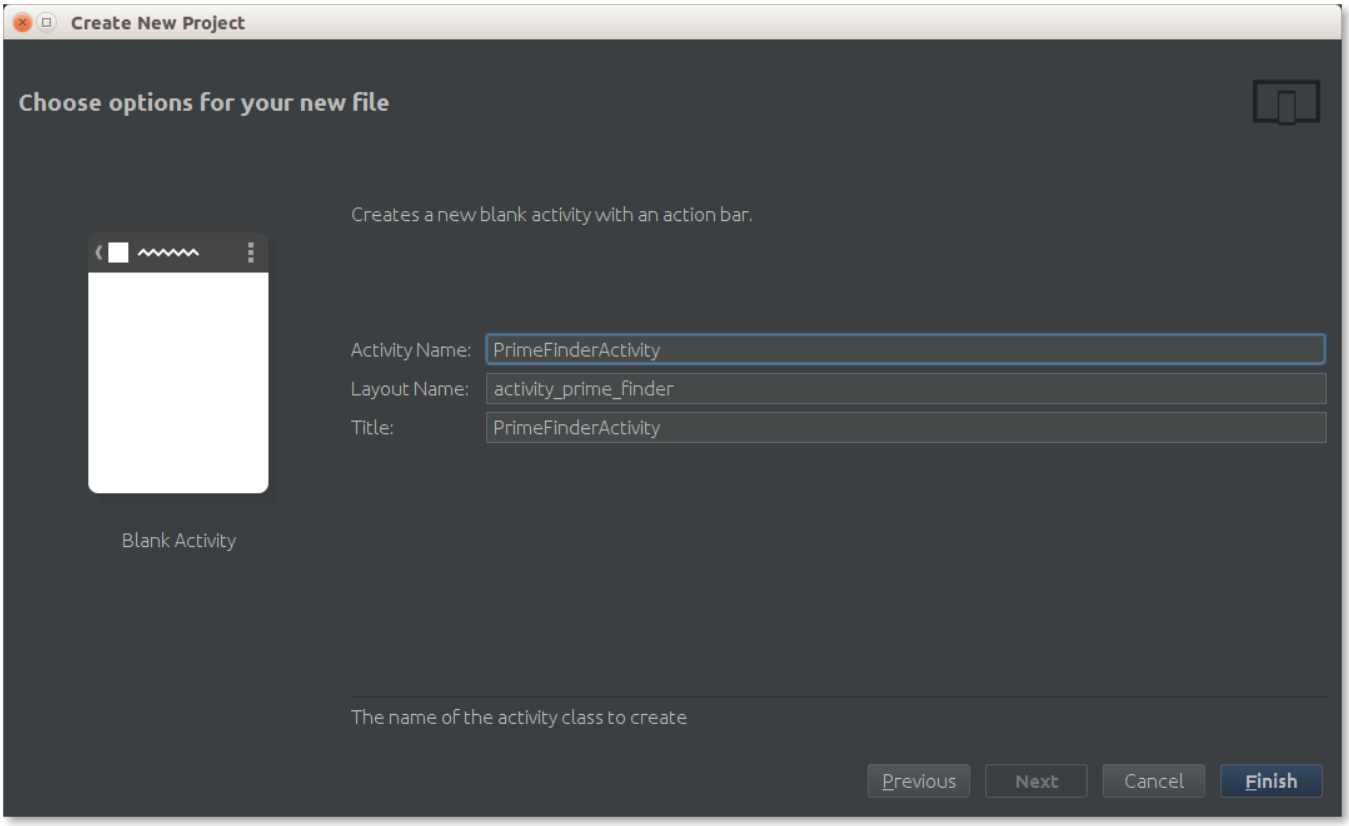
After choosing your form factor and API version, please click “Next”.

You should now see the following dialog:



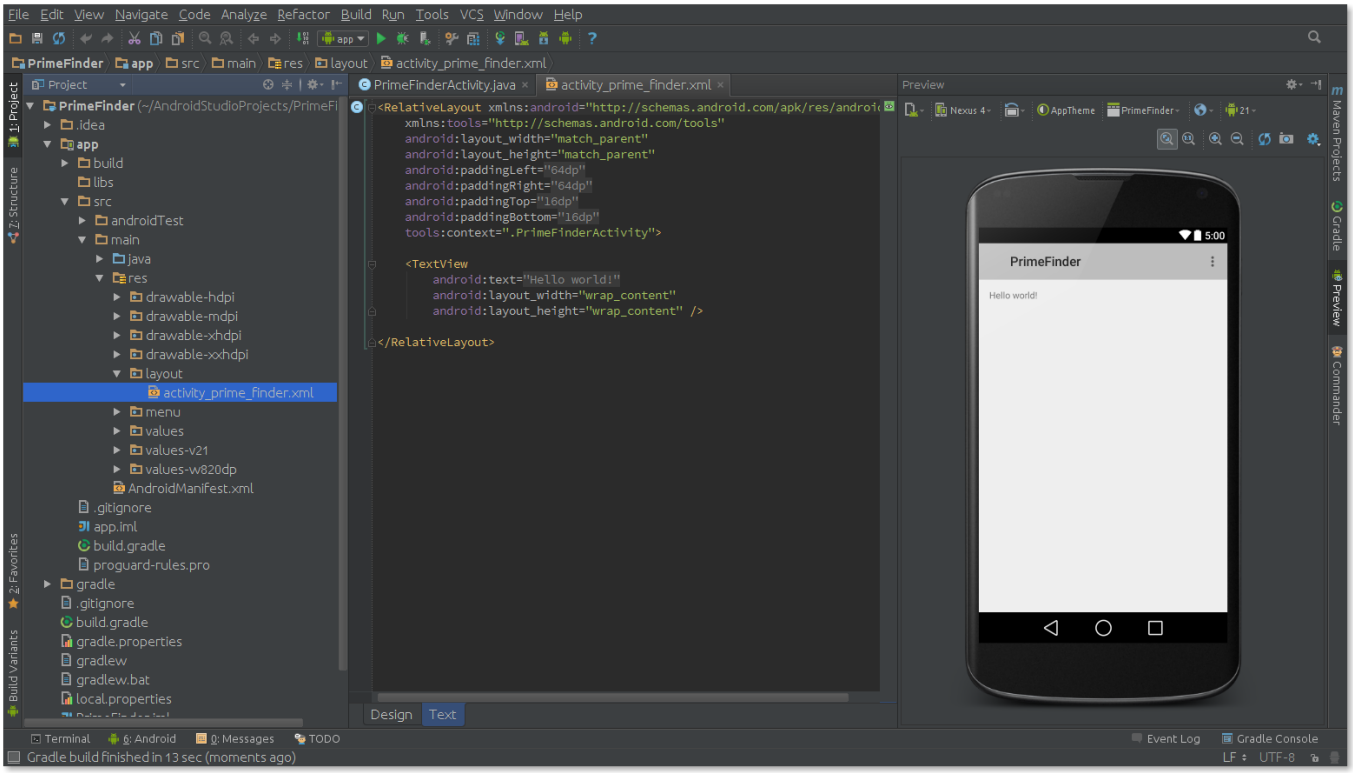
This dialog allows you to add an activity to your project. Select “Blank Activity” and click “Next”.

You should now see the following dialog:



This dialog allows you to configure the Activity that you just created. Please change the Activity Name to **PrimeFinderActivity** so that it matches to form above. Click “Finish” Android Studio will now create your project.

When the project has been created you should see the following:

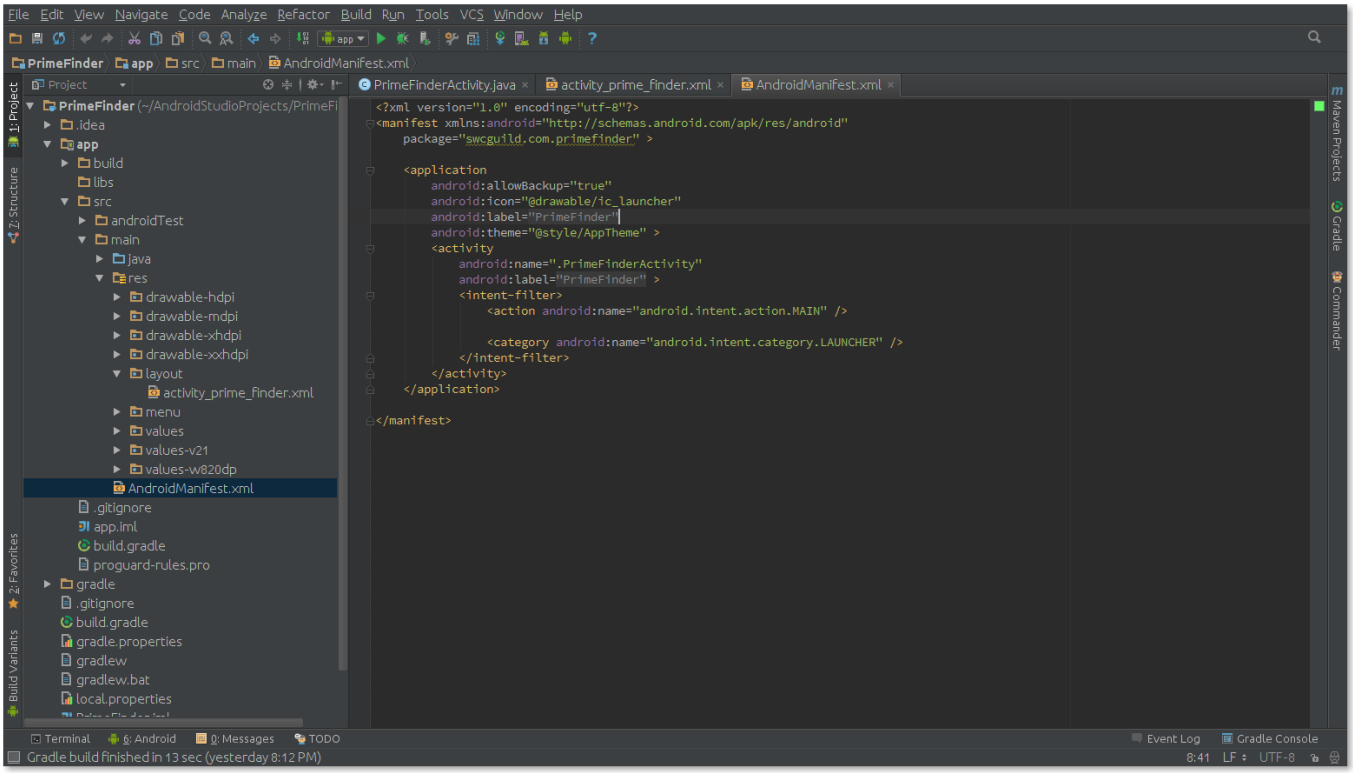


Congratulations! You've just created your first Android project.

Android Project Tour

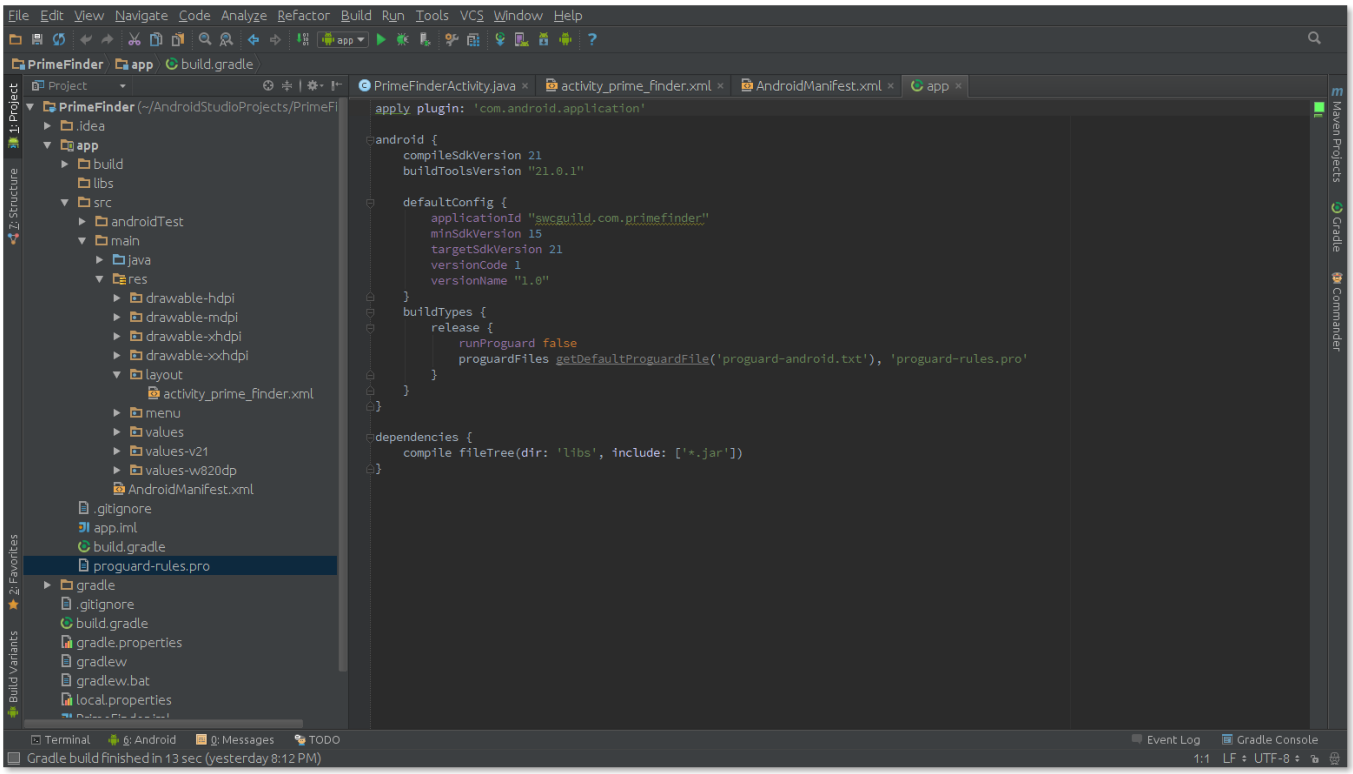
In this section we will take a quick look at the major components of an Android project. This is by no means a complete tour but it will allow us to become familiar with the pieces that we need to understand for our first app.

Manifest File



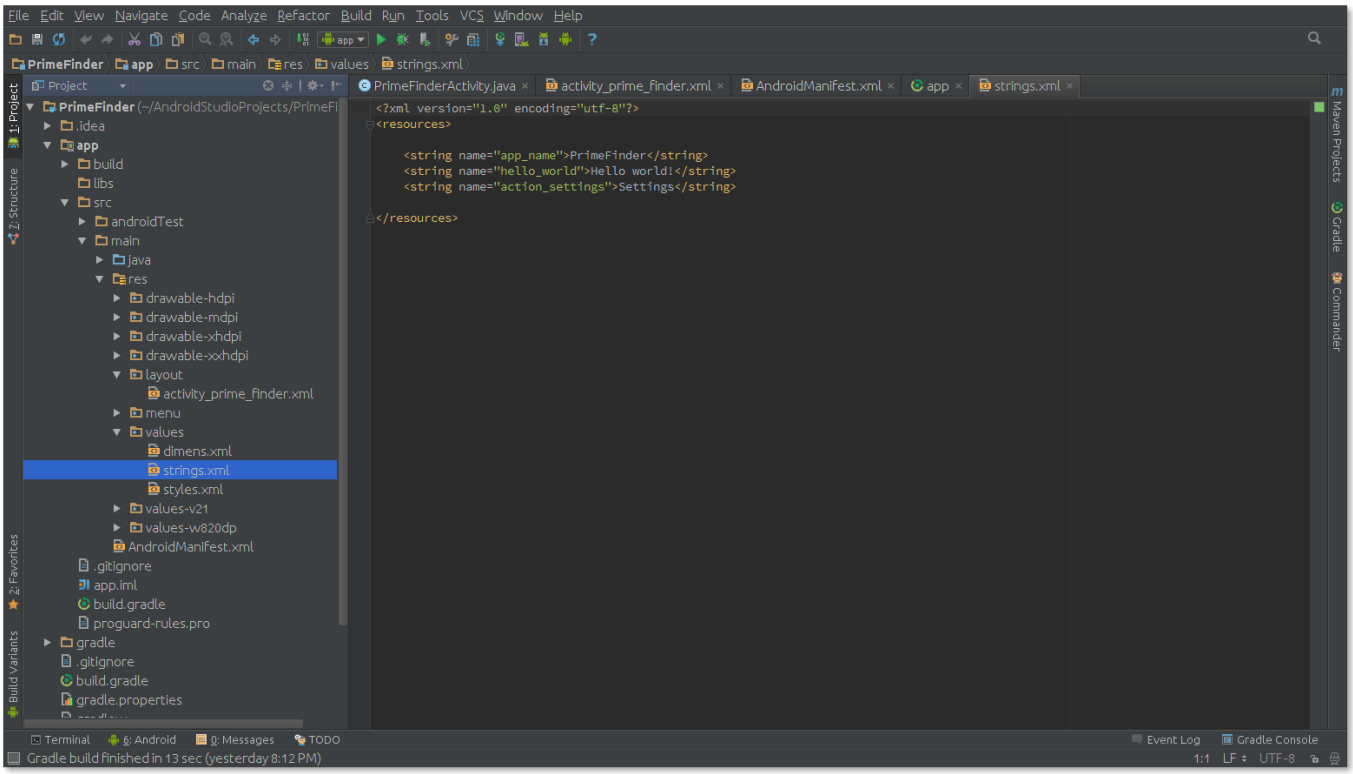
The **AndroidManifest.xml** file describes the basics of your project including the Java package you are using, the name and style of the app, and any activities you have created. Android Studio automatically adds entries for additional Activities as they are created.

Gradle Build Automation



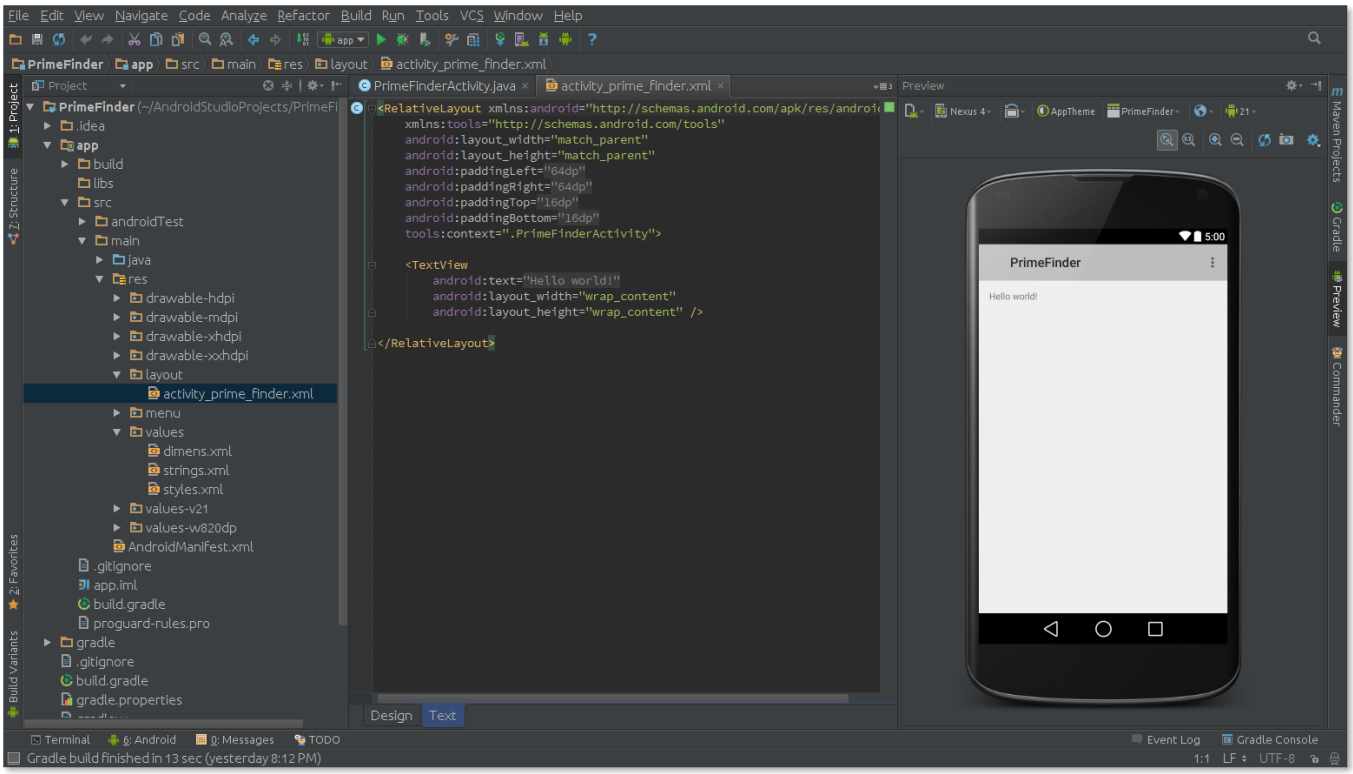
Gradle (www.gradle.org) is the build automation tool that Android Studio uses. The **build.gradle** file describes the setting used to build your app such as the version of Android tools being used, the min and targeted Android API version, the version of your application, and library dependency information.

String Resources

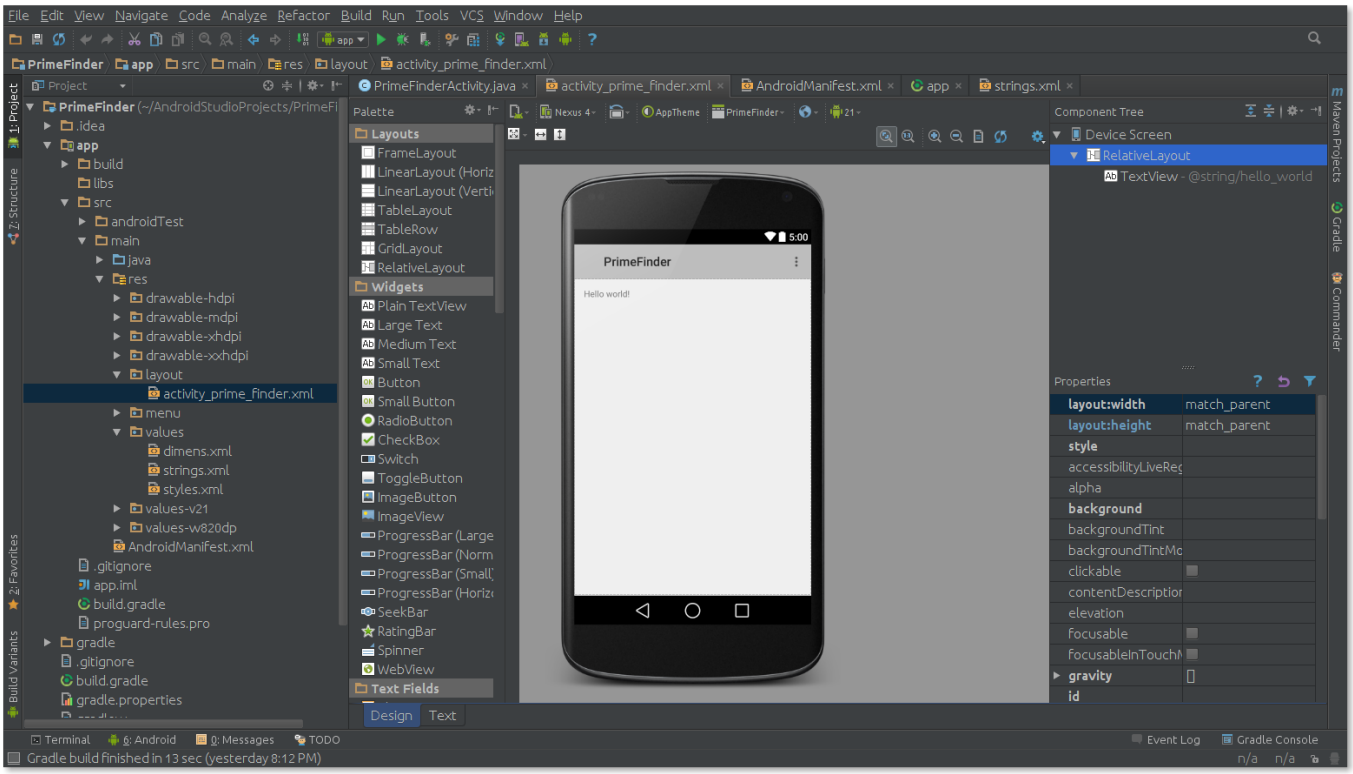


The **strings.xml** file contains the text that appears on various labels, buttons, and other widgets in your application. It is considered a best practice to externalize these values to this file rather than to hardcode the values in your Java code. We will see how to add values to this file and access them later in this lab.

XML Layout Files



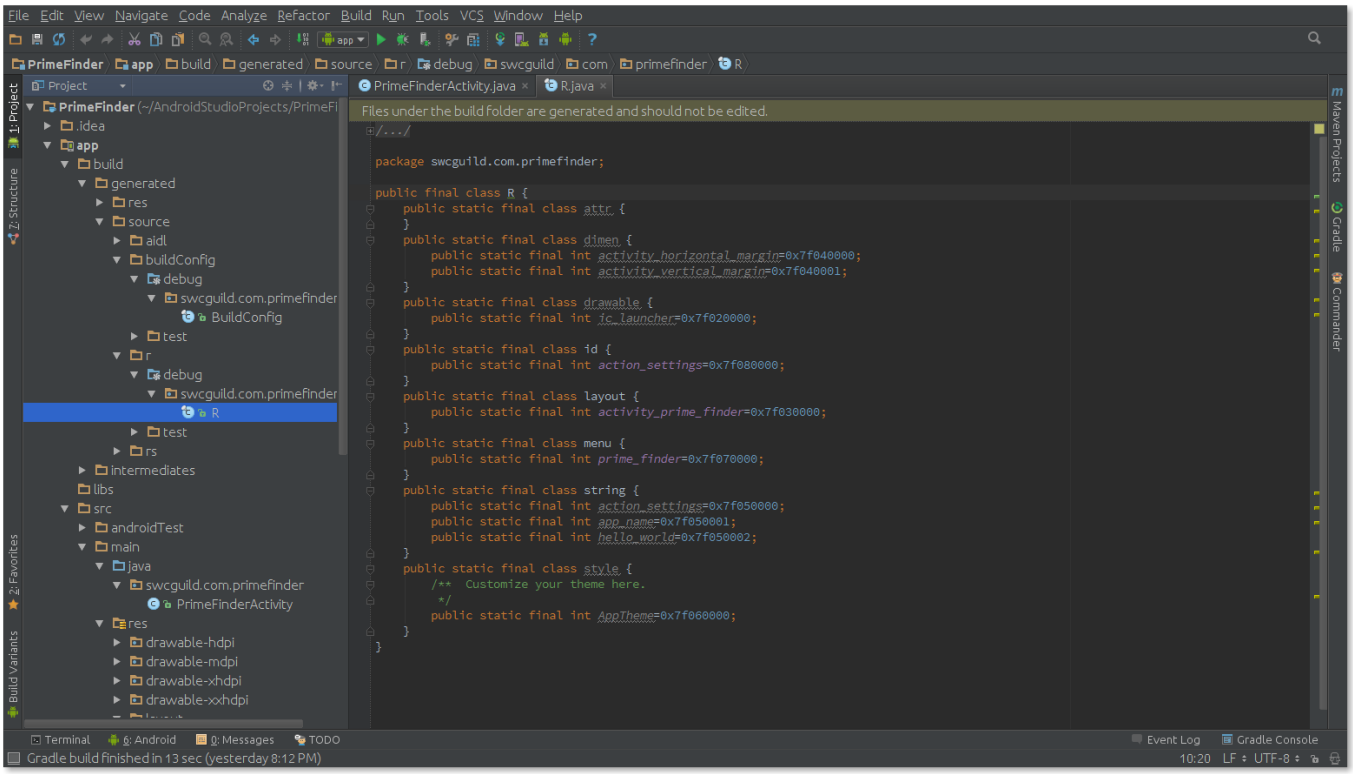
Activity Layout File - XML View



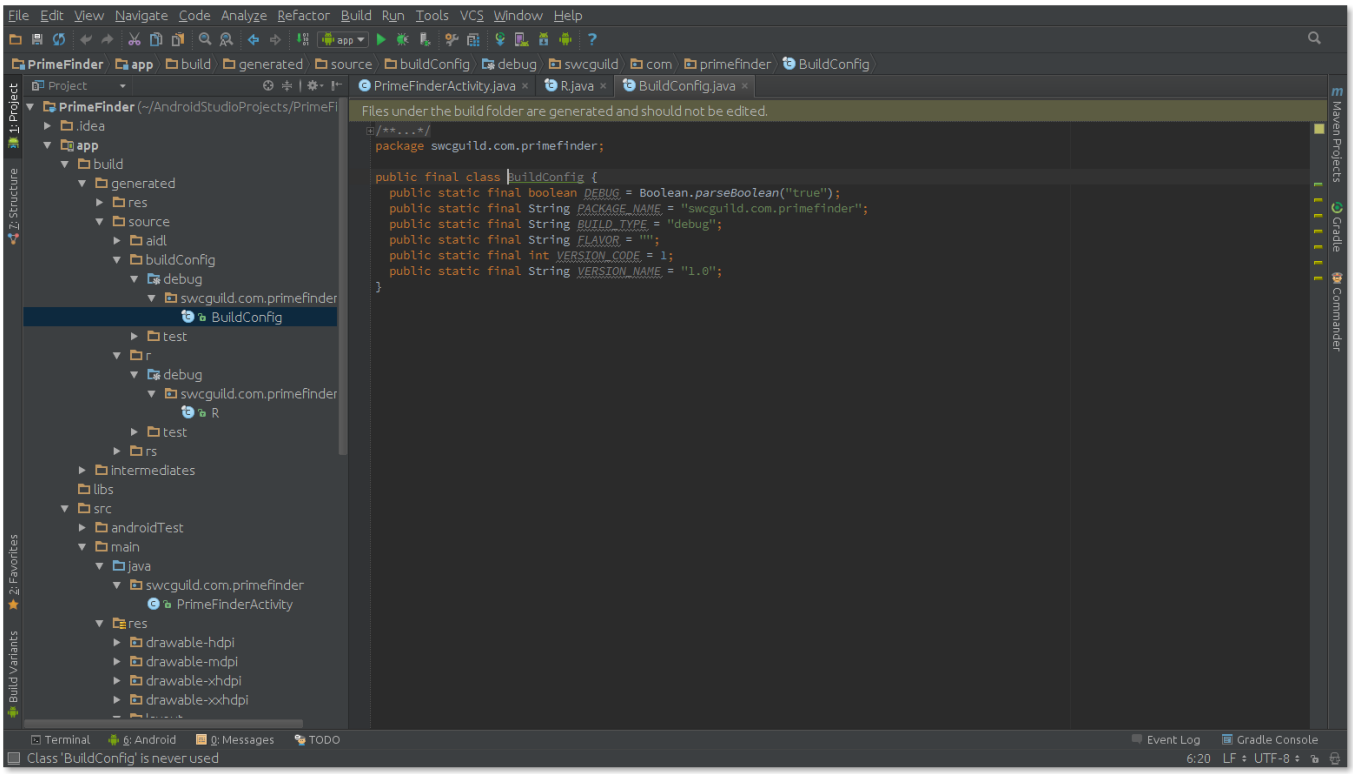
Activity Layout File - Design View

Screen layout information is externalized from your Java code into layout XML files. Each Activity has its own layout file. In our case, we see **activity_prime_finder.xml**. The design view provides a drag-and-drop interface for creating these screens. You can also code them directly in XML if you wish. You will probably use both views as you develop an app.

Generated Source Code

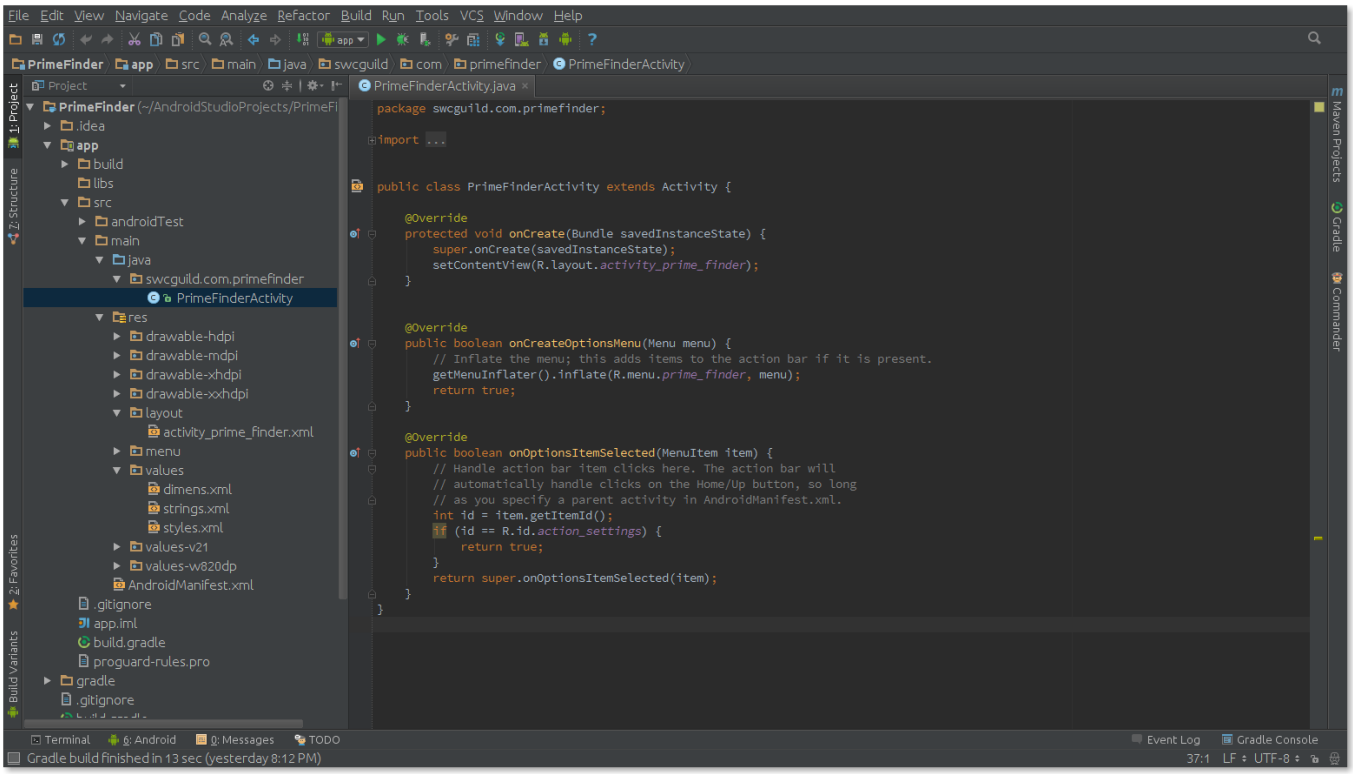


R.java is a generated Java file that bridges the gap between the resource files and the Java world. It is always auto generated, which means **you should never edit this file!**



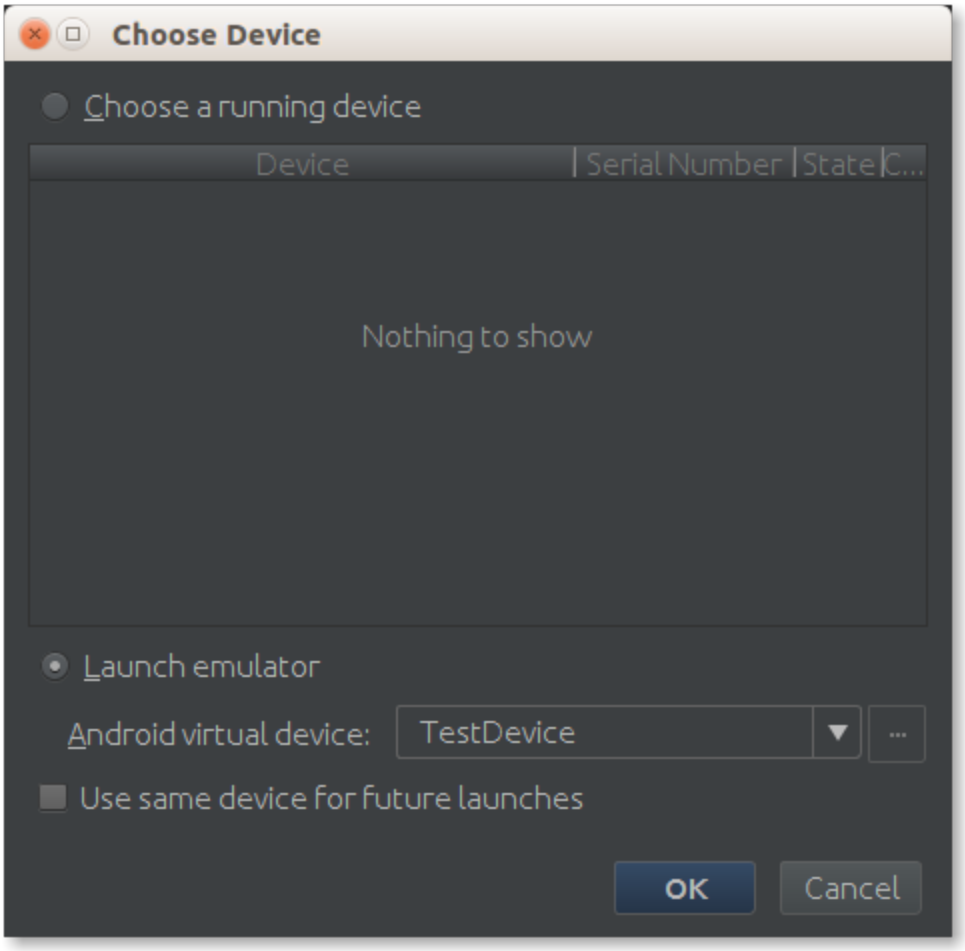
BuildConfig.java contains build setting for your project. Again, because this file is generated, **you should never edit this file!**

Java Code - Activity

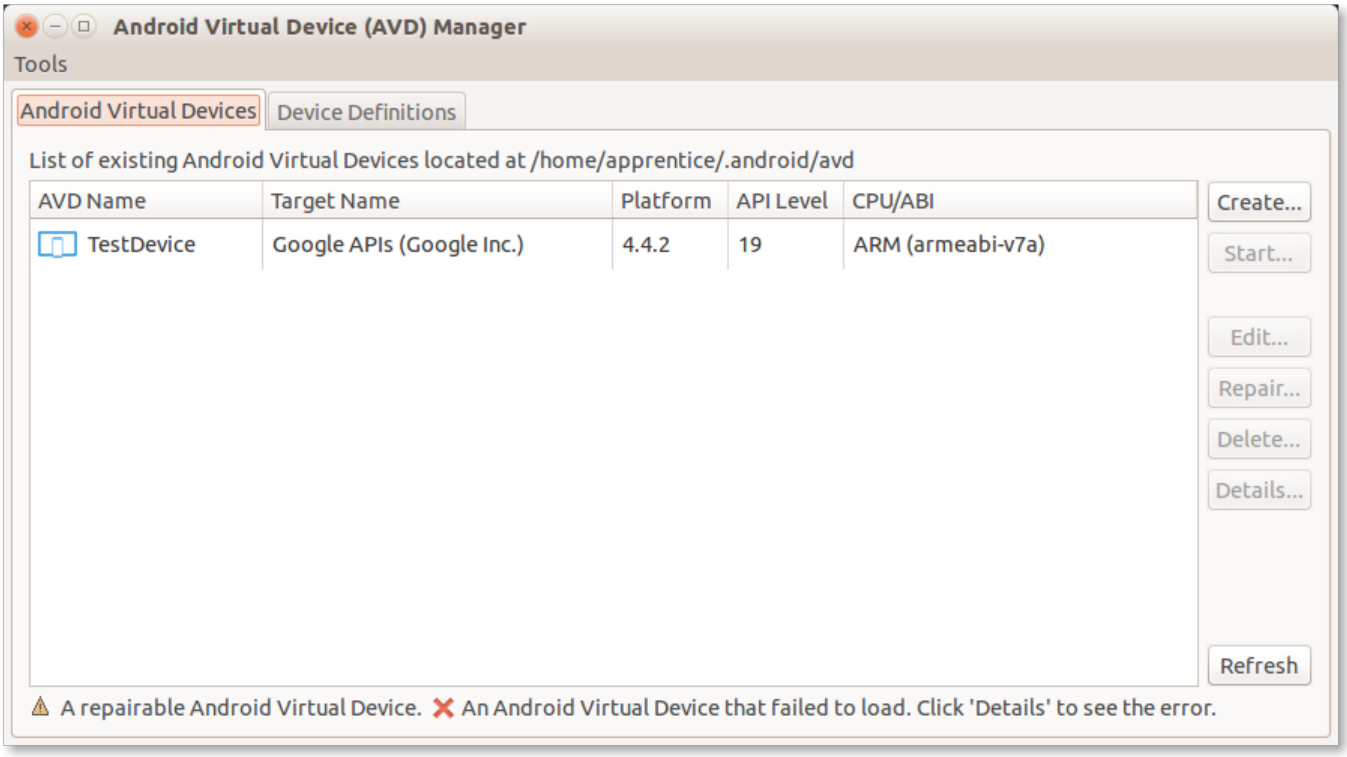


Finally, we get to some Java code! **PrimeFinderActivity.java** contains the application logic for your Android app. Right now it doesn't do anything except display "Hello, World!" - we'll fix that in the next section.

Running the App - Emulator/Phone/Tablet



When you click the **Run App** button (green arrow right underneath the **Run** menu), Android studio will prompt you to choose a device on which to run the application. If you have a physical Android device plugged into your machine (via USB), it will be shown as an option in the top window (we'll see how this is done later). You can also choose to launch an emulator. All of the emulators that you have configured will show in the **Android virtual device** drop down. Choose a device from the dropdown and click 'OK' to run your application. If you don't have a virtual device configured, click the ellipses (...) to launch the Android Virtual Device (AVD) Manager. The AVD Manager looks like this:



From this window, you can create an Android virtual device on which to run your app.

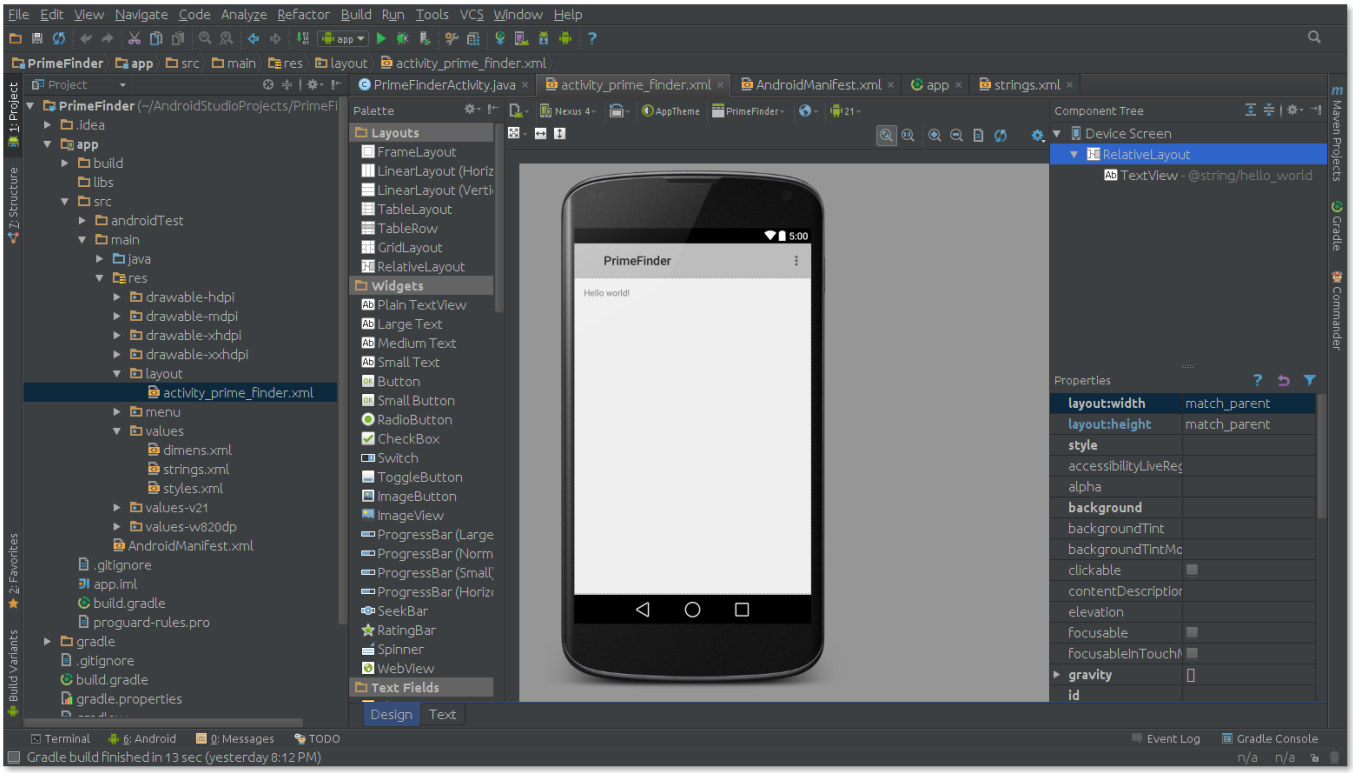
Coding the App - PrimeFinder

Now we will create a simple app that will tell us if a number is prime or not. The major features of this app will be:

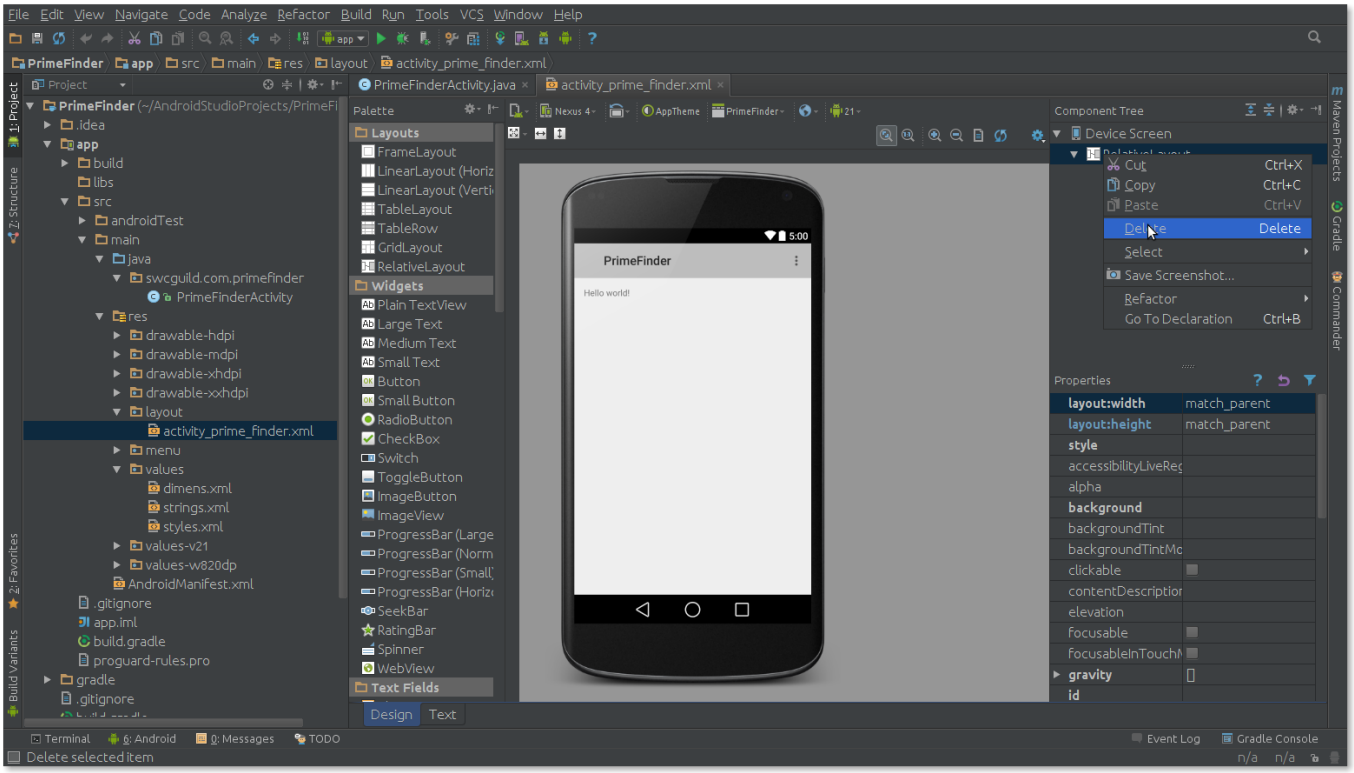
1. Text input box
2. Button
3. 'Toast' notification
4. Business logic to calculate the answer

Deleting Existing Layout

The first step in this app is to delete the existing Activity layout that displays “Hello, World!” Double click on the **activity_prime_finder.xml** file and then click the **Design** view:

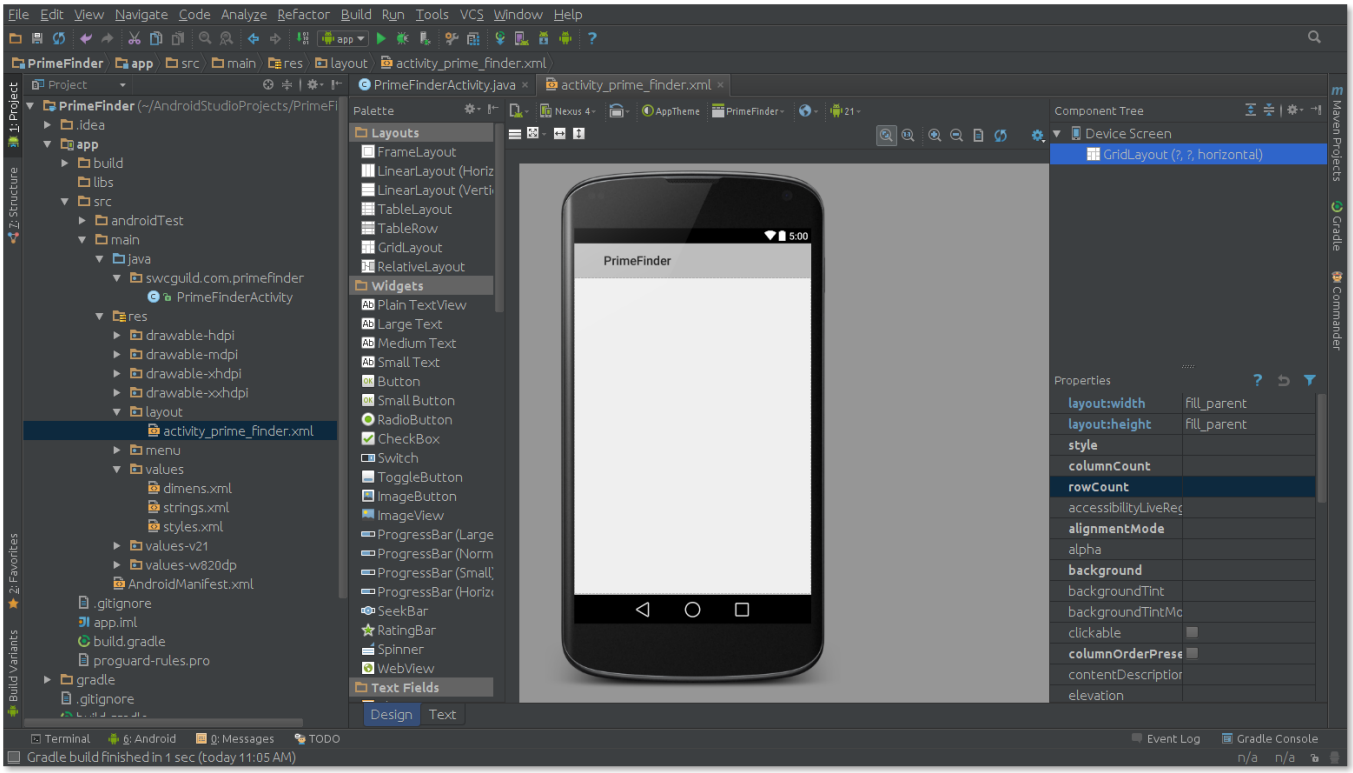


Now, right click on the **Relative Layout** item in the Component Tree window (upper right hand corner of screen - highlighted in blue above) and select **Delete**.

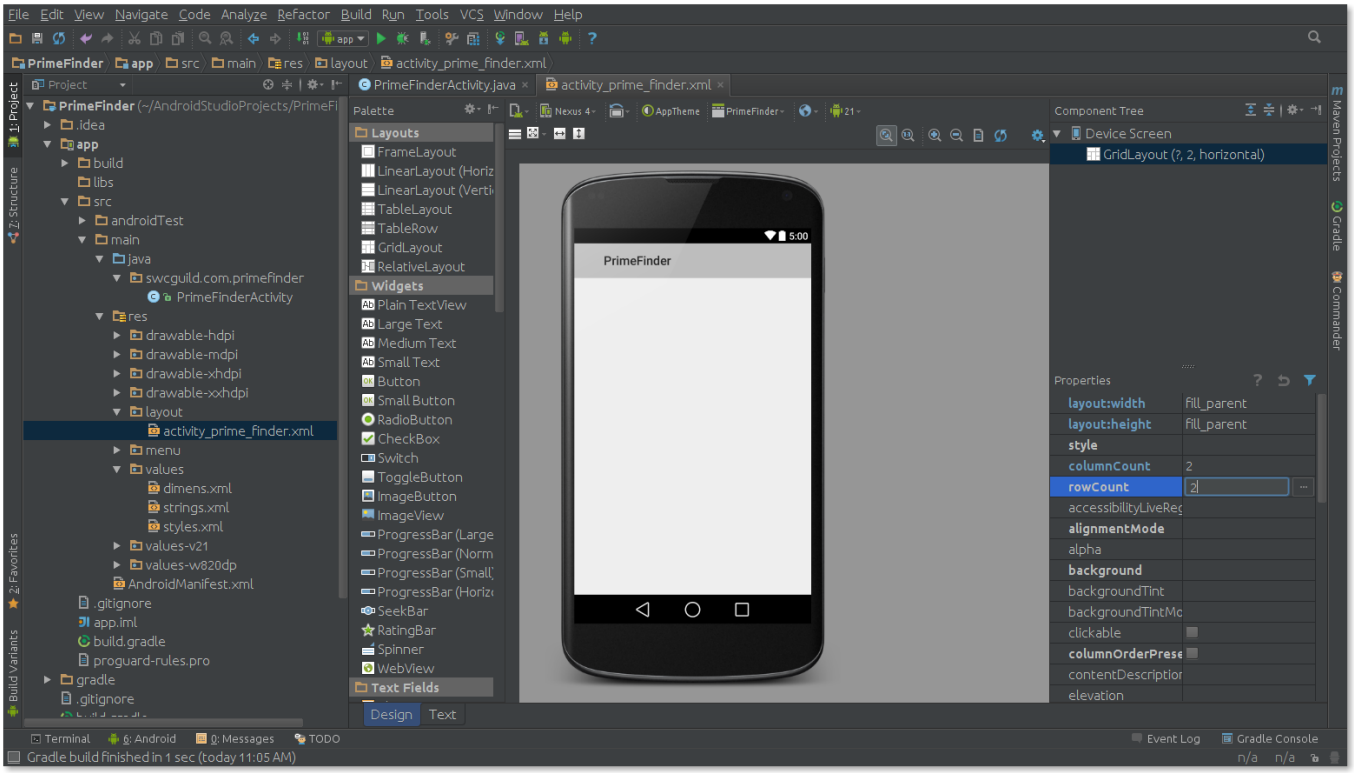


Creating the New Layout

Our next step is to create a new view for our application. We will use the **Grid Layout**. Select the Grid Layout items from the Layouts folder in the Palettes pane and drag it over to the Device Screen in the Component Tree. When you are done, it should look like the following:

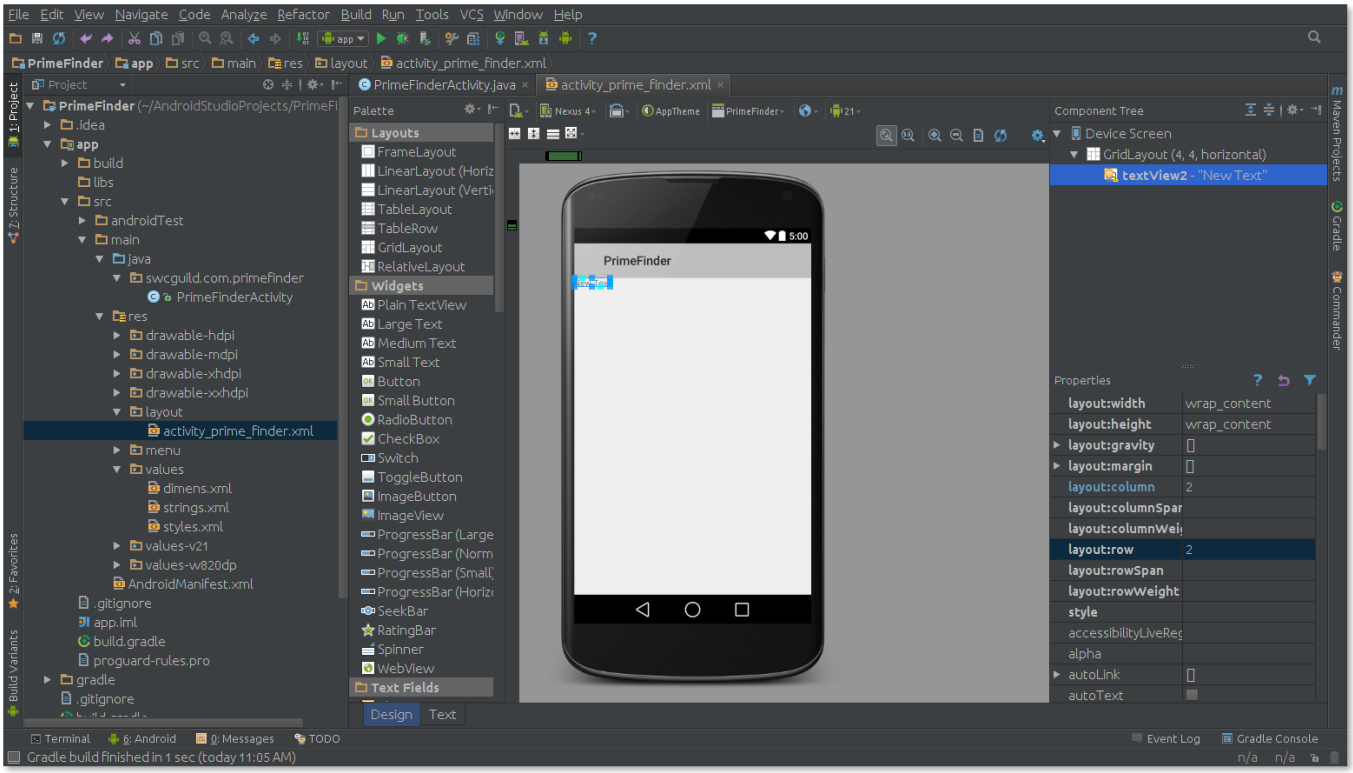


Now we must specify the number of rows and columns in our grid. These values are specified in the **columnCount** and **rowCount** items of the **Properties** pane. Please set both these values to **2**:



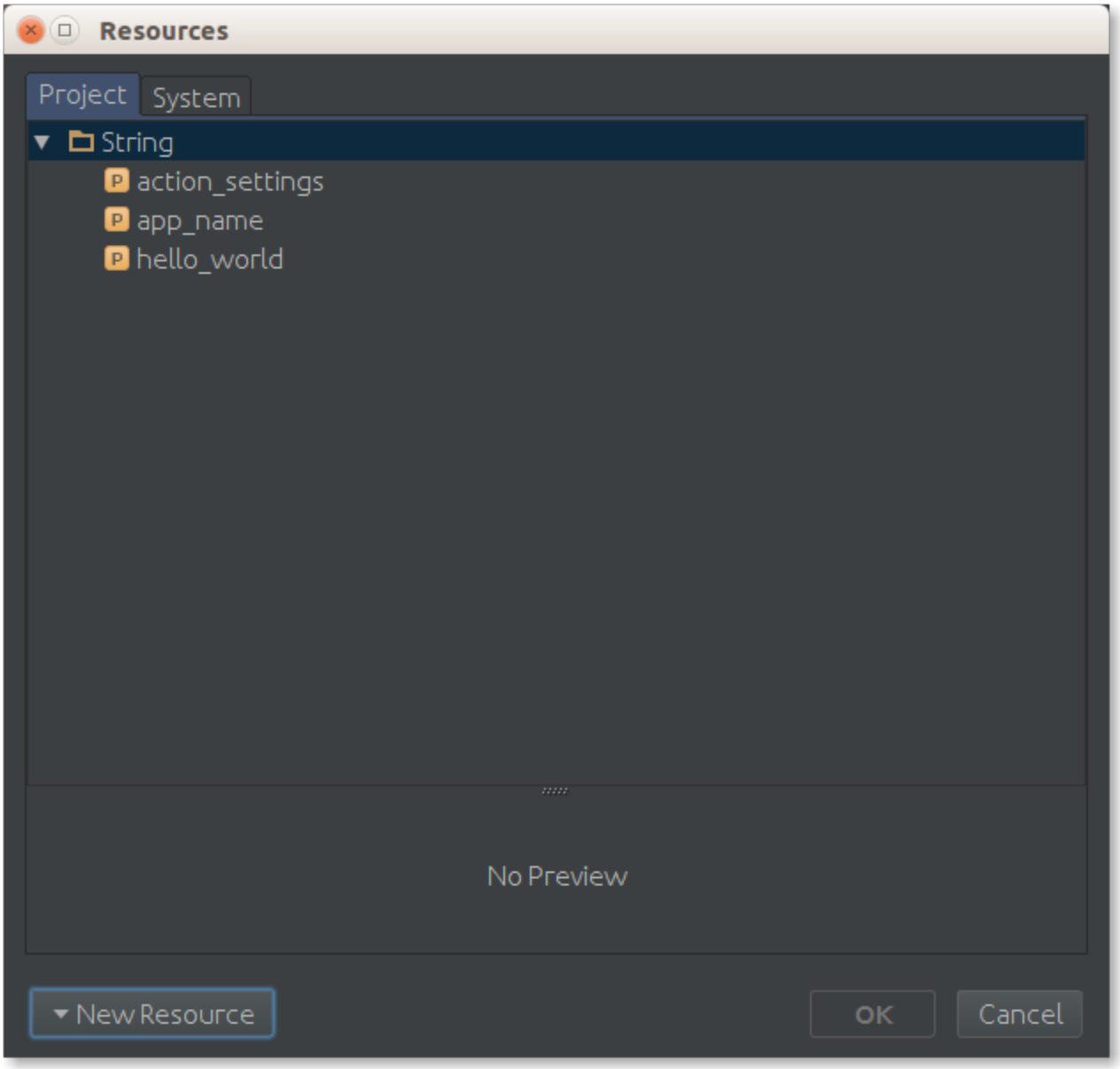
Add Label

Next we will create the label for our text input box. Drag the **Plain TextView** widget and drag it onto the screen (anywhere will do, we'll fix the location next).

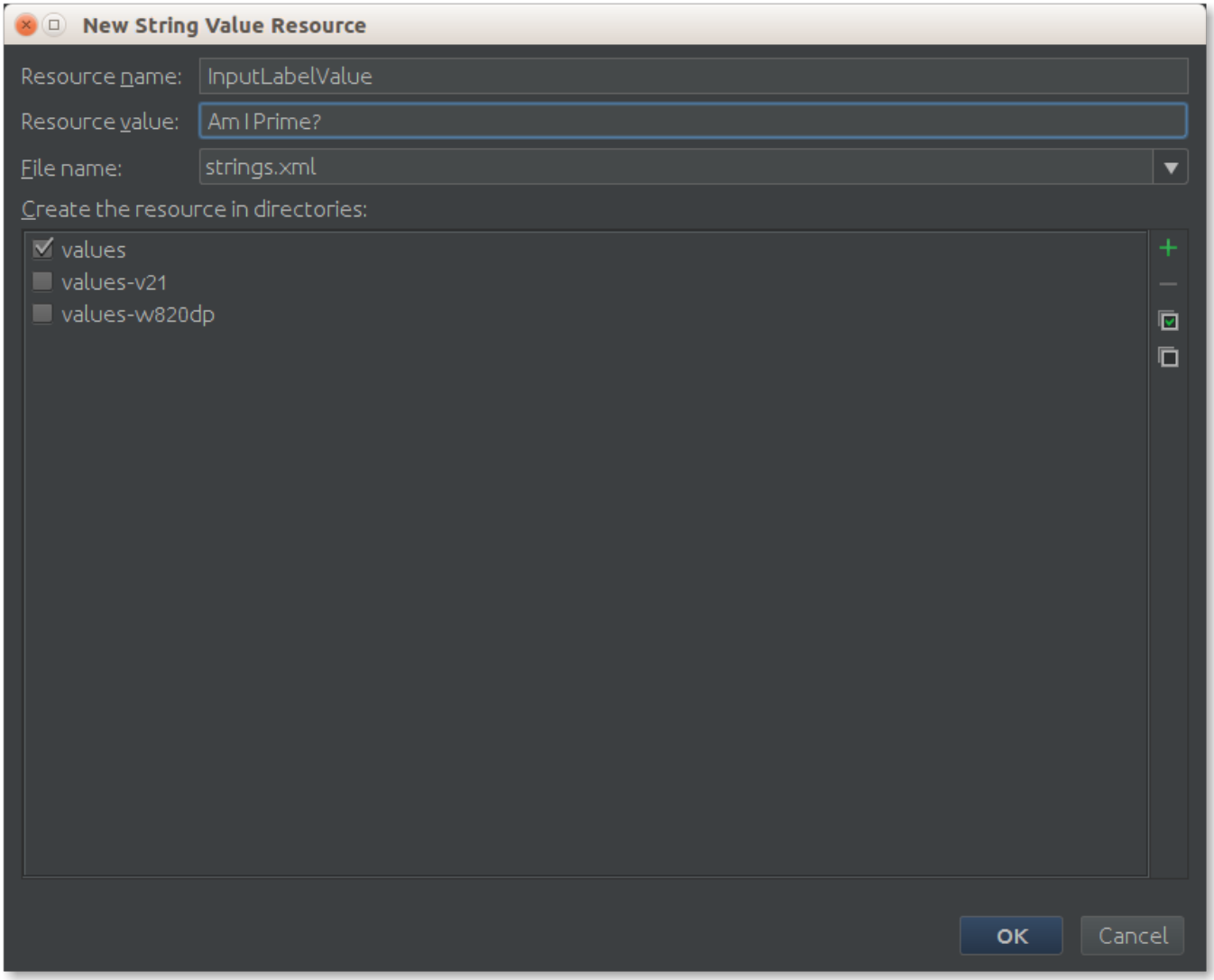


Now we will place the label - set **layoutColumn** and **layoutRow** to **2** (as shown above). Now, in the **Properties** pane, scroll down to **Id** and set it to **InputLabel**.

Now we will create new string resource for our label. The following steps will create a new entry in the **strings.xml** file that we discussed earlier. Scroll down to the **Text** property and click the ellipses to bring up the Resources Dialog:



Click the **New Resource** button, select **New String Resource...** and fill in the form as follows:



Now click **OK**.

Finally, we'll set the text size for our label. Scroll down to **textSize** and set it to **20dp**.

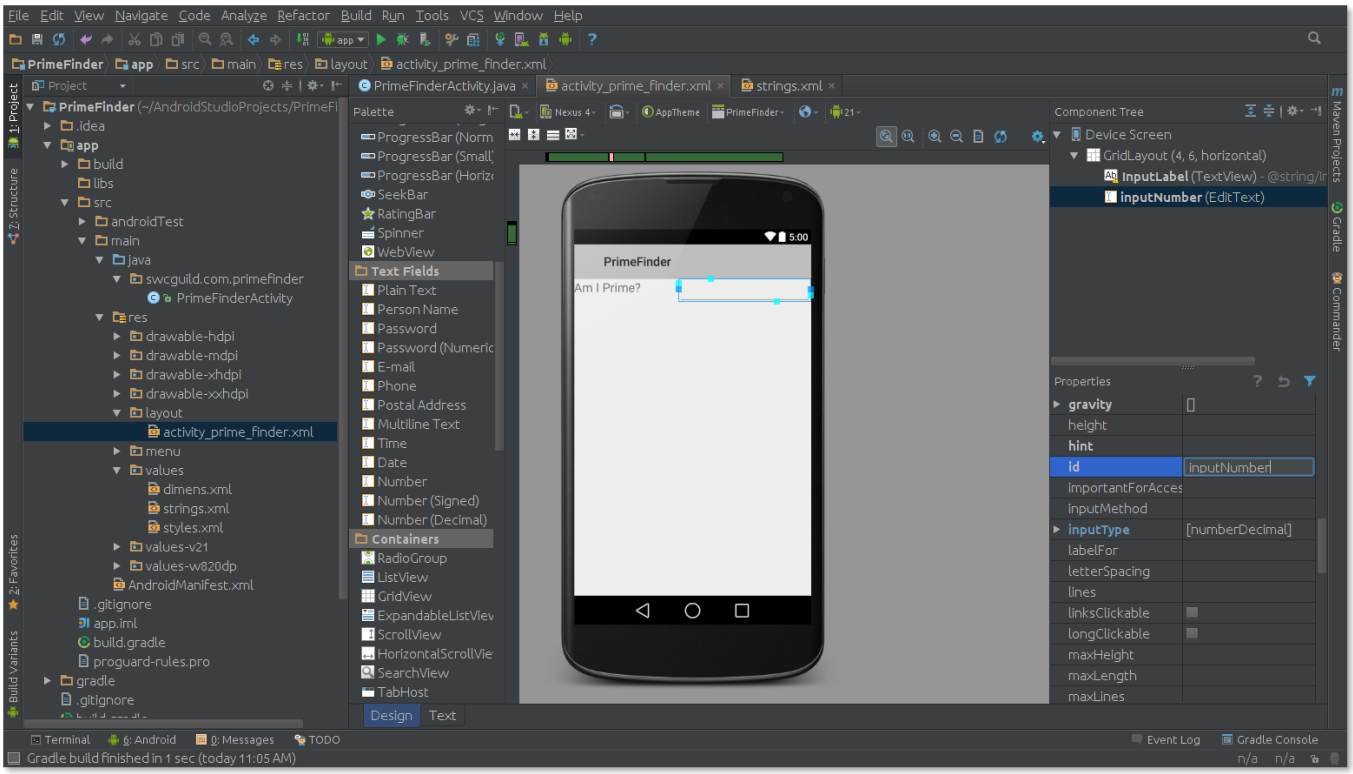
Add Text Box

Now we will add the input field for our app. Grab the **Number (Decimal)** item from the **Text Field** folder in the **Palette** pane and drag it onto the screen next to the label and change the **Id** to **inputNumber**:

Mobile Monday Tech Lab 01

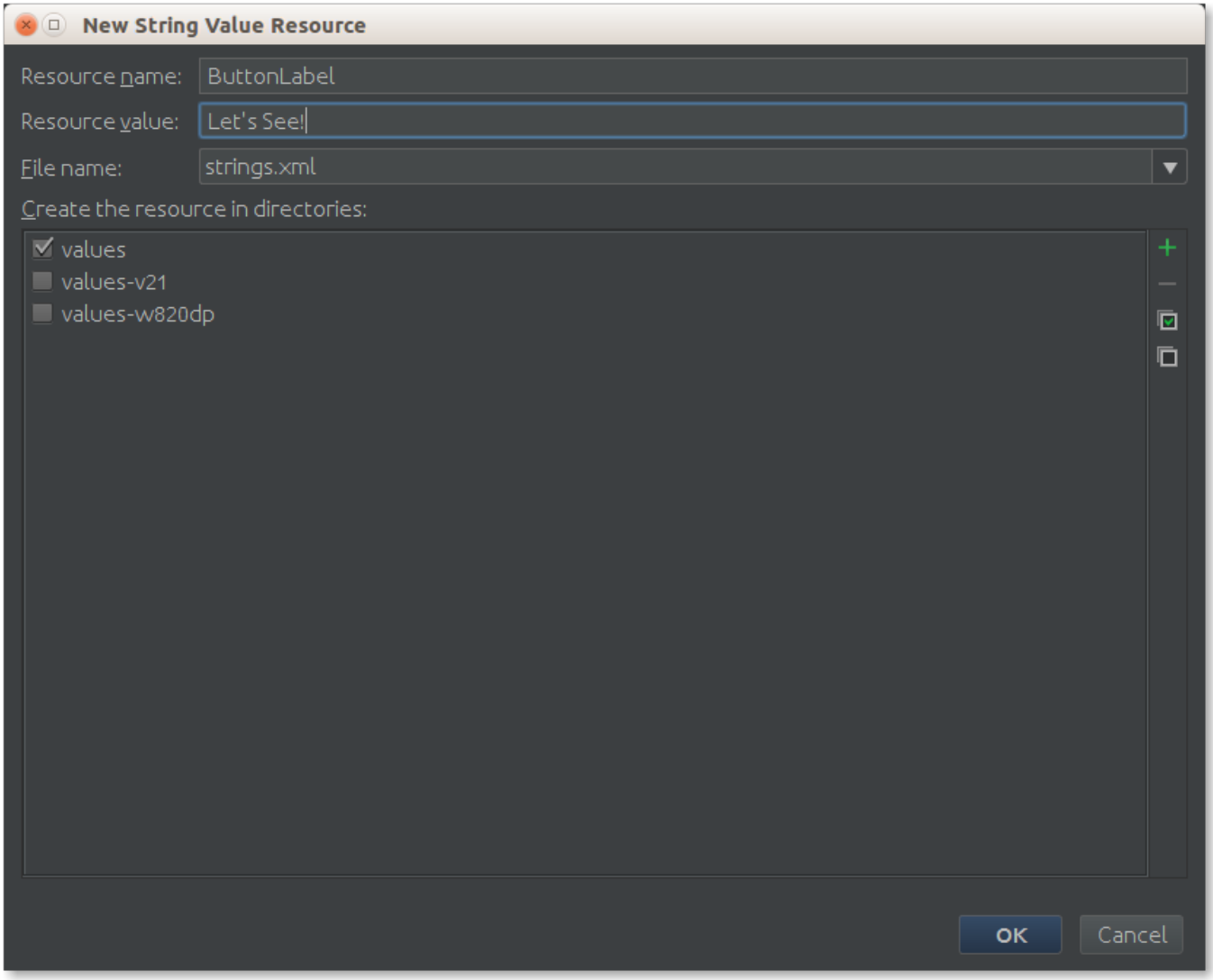
SOFTWARE CRAFTSMANSHIP GUILD

Android Development - Basics

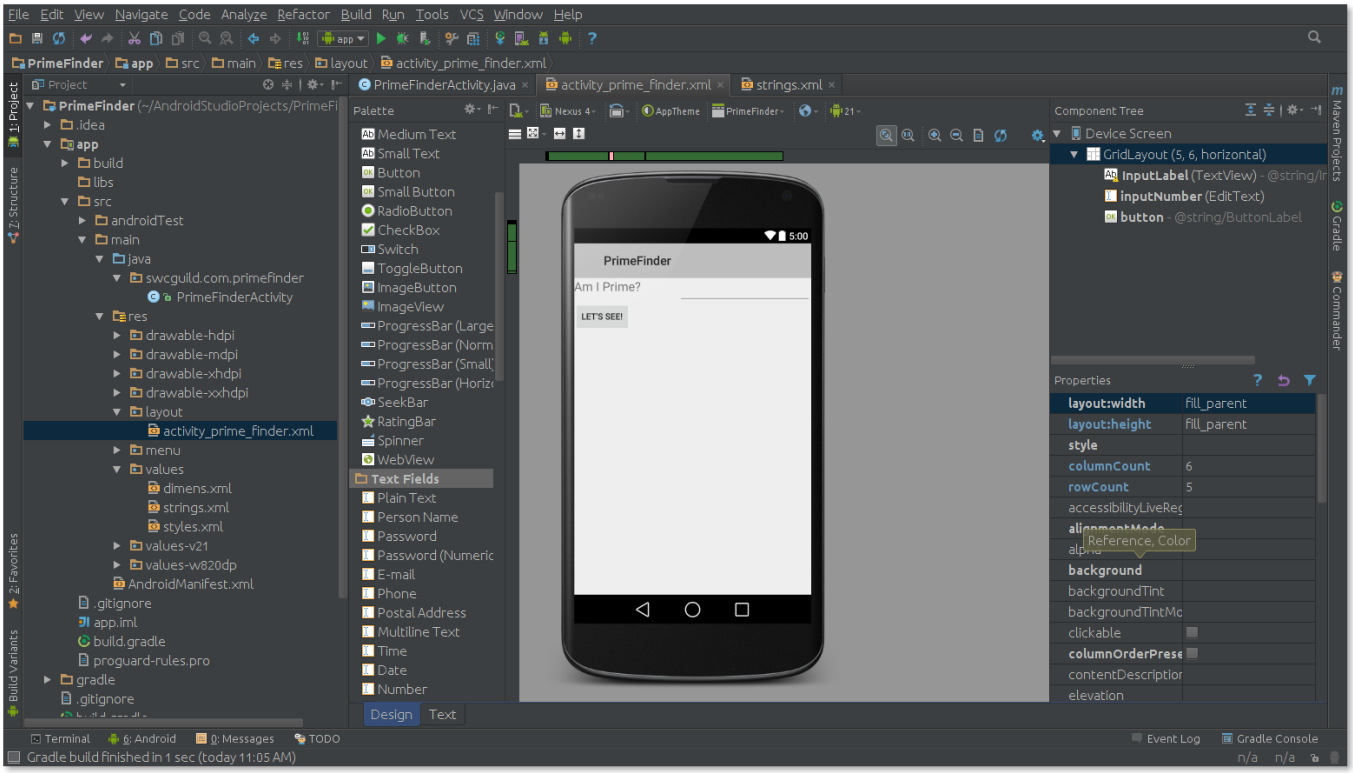


Add Button

We will complete our layout by adding a button. Drag the **Button** widget onto the screen just beneath the Input Label. Change the **inputRow** value for this widget to **3** and change the **Id** to **button**. Now scroll down to the **Text** property and create a new String Resource for the button (like you did for the label above). Set the value to “Let’s See!”:



Your Android Studio screen should now look like this:



Application Code

Now that the layout has been created, we can write our application logic. All of our coding will be done in the **PrimeFinderActivity.java** file.

Step one is to create a class level variable to hold our text input box control. This variable will be of type **TextView** and we'll initialize it in the **onCreate** method. Our application logic will access this variable to get the number that the user has entered.

Add the following as a class level variable of your PrimeFinderActivity class:

```
// this is the input box control where the number will be entered
// by the user.  we'll initialize it in the onCreate method and
// access it in our application logic
private TextView numberInputView;
```

Now, modify the **onCreate** method so that it looks like this:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_prime_finder);
    // initialize our TextView control variable
    numberInputView = (TextView) findViewById(R.id.inputNumber);
}
```

These modifications may cause compilation errors because you haven't included the required **import** statements for the new code. Click on the **Code → Optimize Imports…** menu item (or press Ctl-Alt-O) to add the necessary import statements.

Our next step is to create an onClick listener for our button. This code will run when our button is clicked. It will read the value in the input text box, check to see if it is prime, and will give the answer to user in the form of a 'toast' popup. Add the following code to the PrimeFinderActivity class (after the onOptionsItemSelected method:

```
private View.OnClickListener buttonListener = new View.OnClickListener() {
    public void onClick(View v) {
        String answer = "Yes!";
        // get the entered string value from the numberInputView
        String numString = numberInputView.getText().toString();
        // if the user entered a value, go ahead and process it
        // if not, display an error message
        if (numString != null && !numString.isEmpty()) {
            // now parse the String into an int - we've locked the input
            // so that it only accepts numbers so there is no possibility of
            // a number format exception
            int num = Integer.parseInt(numString);
            if (num%2 == 0) {
                // it's even so it can't be prime...
                answer = "Nope!  It's even...";
            } else {
                // we can start at 3 and we only have to go up to the square root of
                // the number to check for prime
                for (int i = 3; i * i <= num; i += 2) {
                    if (num%i == 0) {
                        // it has at least one factor so it can't be prime
                        answer = "Nope! It has factors!";
                        // go ahead and quit - we don't need to find all of the
                        // factors - if we find one, we're done...
                        break;
                    }
                }
            }
        } else {
            // user did not enter a value
            answer = "Hey! I said enter a number!";
        }
        // show answer in toast popup
        Toast.makeText(PrimeFinderActivity.this, answer, Toast.LENGTH_LONG).show();
    }
};
```

Finally, we have to wire our onClick handler to our button in the onCreate method. Change your onCreate method to match the following:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_prime_finder);
    // initialize our TextView control variable
    numberInputView= (TextView) findViewById(R.id.inputNumber);
    // find our button and initialize our variable
    Button myButton  = (Button) findViewById(R.id.button);
    // add the on click listener to our button
    myButton.setOnClickListener(buttonListener);
}
```

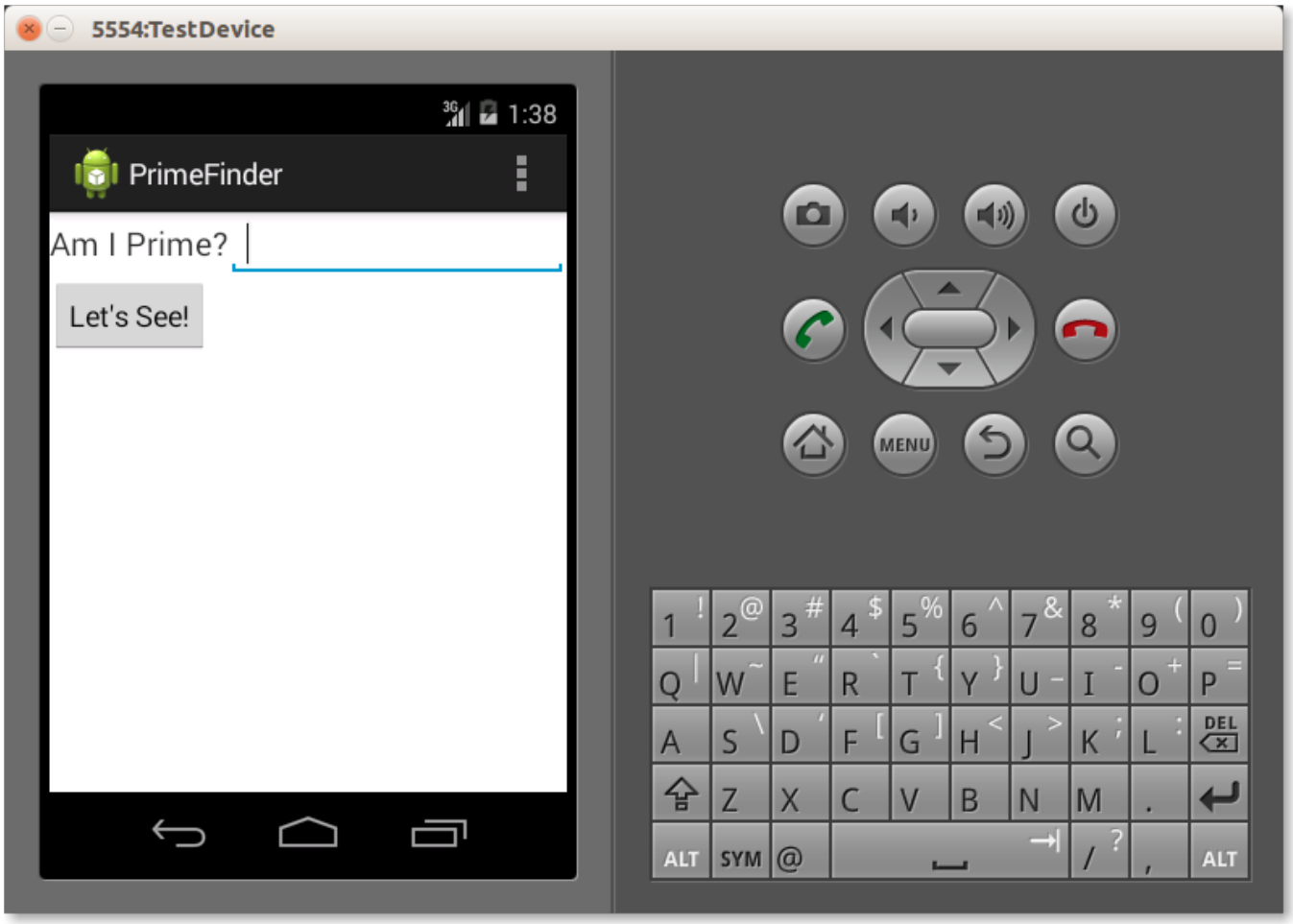
You may need to fix up your import statements again at this point. Click **Ctl-Alt-O** to do this.

All Done!

That completes our simple app. Now it's time to see it in action.

Running the Emulator

Running our app on the emulator is as simple as clicking the **Run Project** button. The **Choose Device** dialog will come up. Simply choose your virtual device (or create a new one) and click **OK**. The emulator can take some time to display and your view will be different for different virtual devices but that app should look something like this:



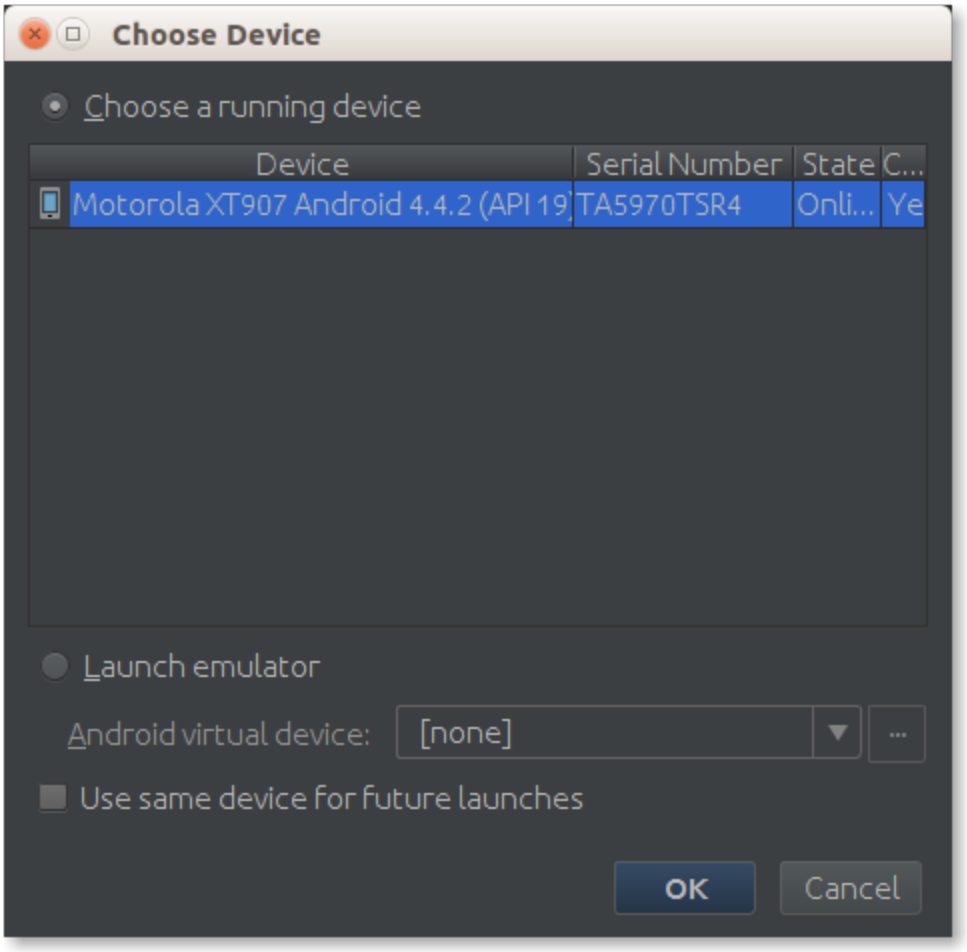
Running on Phone/Tablet

The first step to getting our application to run on your device is to put your device in developer mode. On my phone (RAZR M running Android 4.4.2) you have to go to the **Settings** screen, click on **About phone**, and tap the **Build number** entry seven times, after which you are in developer mode. Other phones, devices, and Android version may be different - you'll have to Google to get instructions for your particular setup.

Once in developer mode, connect your phone to your computer via USB. You should get two messages on your device:

1. Connected as a Media Device
2. USB Debugging Enabled

Make sure that you've shut down the emulator from the previous step. Now go back to Android Studio and click the **Run Project** button - your phone/tablet should now show up as an option:



When you click **OK**, the app will begin to run on your device. The app will remain on the device even after you remove the USB cable. In the next Tech Lab, we'll see how to package our application as an APK and install the app to a physical device.

Wrap Up

That does it for this lab. In future labs we'll explore more advanced topics such as on device storage, graphics, and integration with web APIs.

Thanks!