HI all, this is going to be part 1 of a series of an Android tutorial development! I’ve spent a good year working on Android and have learned a lot.

However, for the first part of the tutorial, we’ll get orientated with what Android provides by creating a simple Button app!

I will assume that you will have limited knowledge of Android so I will try to explain everything.

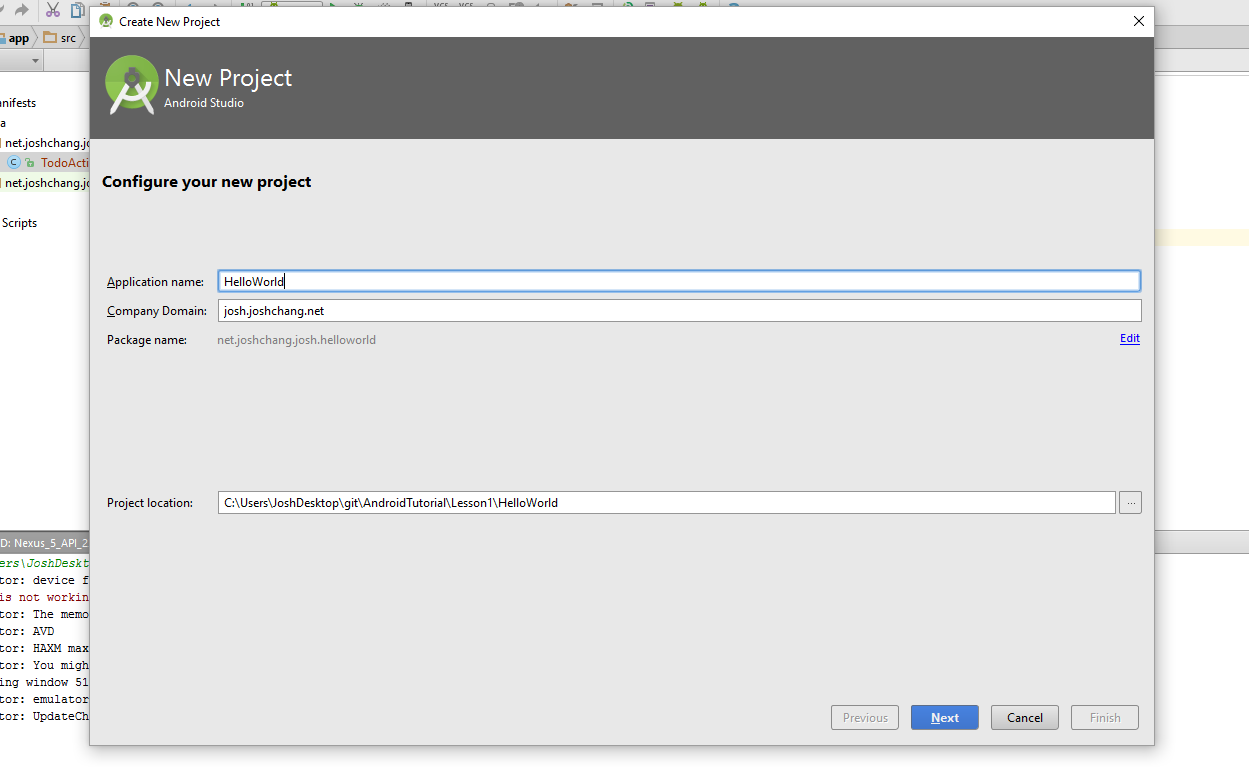
However I’m also assuming that you have some knowledge in java or other programming languages. If not I highly recommend going through [Code Academy for Java](https://www.codecademy.com/learn/learn-java)

# Setting up the project

If you don’t have Android Studio already installed, go here and [install Android Studio](http://developer.android.com/sdk/installing/index.html?pkg=studio)

Have Android Studio installed? Good!

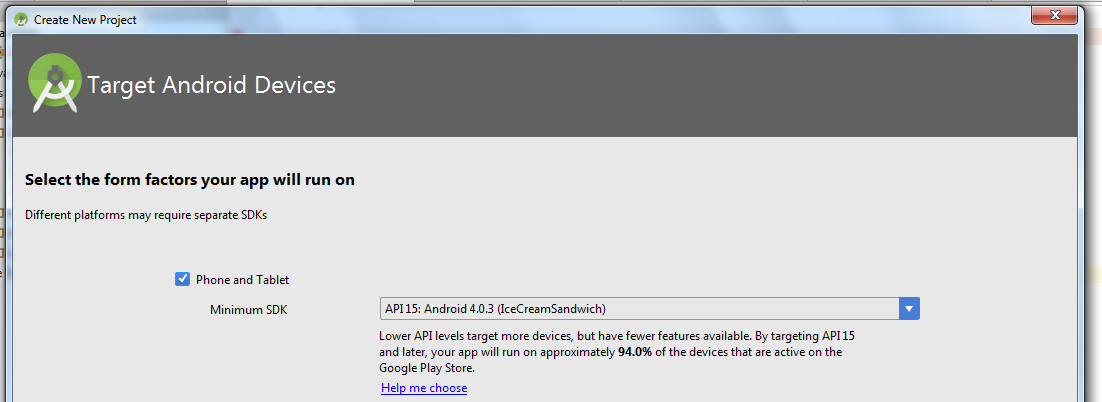
First thing to do is to create a new Project. Go to File -> New -> New Project...



You’ll see 2 fields,

1. application name – This is the name of our project. In our case I called the project **HelloWorld**
2. domain name – This is a unique identifier for your app if you decide to submit it to the play store. It can be anything and it can be changed to something else later.

Afterwards you’ll see a list of targeted API level:



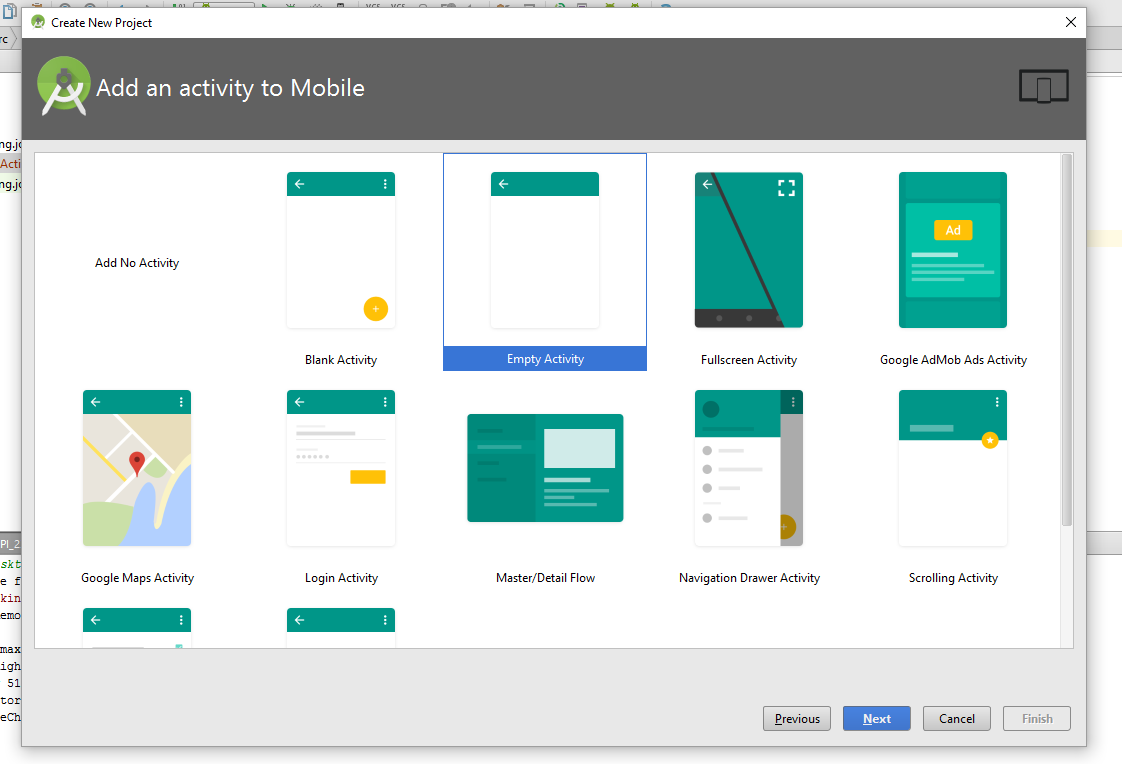
Every phone has an API level based off of the current update they’re on. The newer updates have performances fixes and new features.

So why would you ever want to use older API’s you might ask? Depending on what phone you have, you can’t upgrade to the newer API levels.

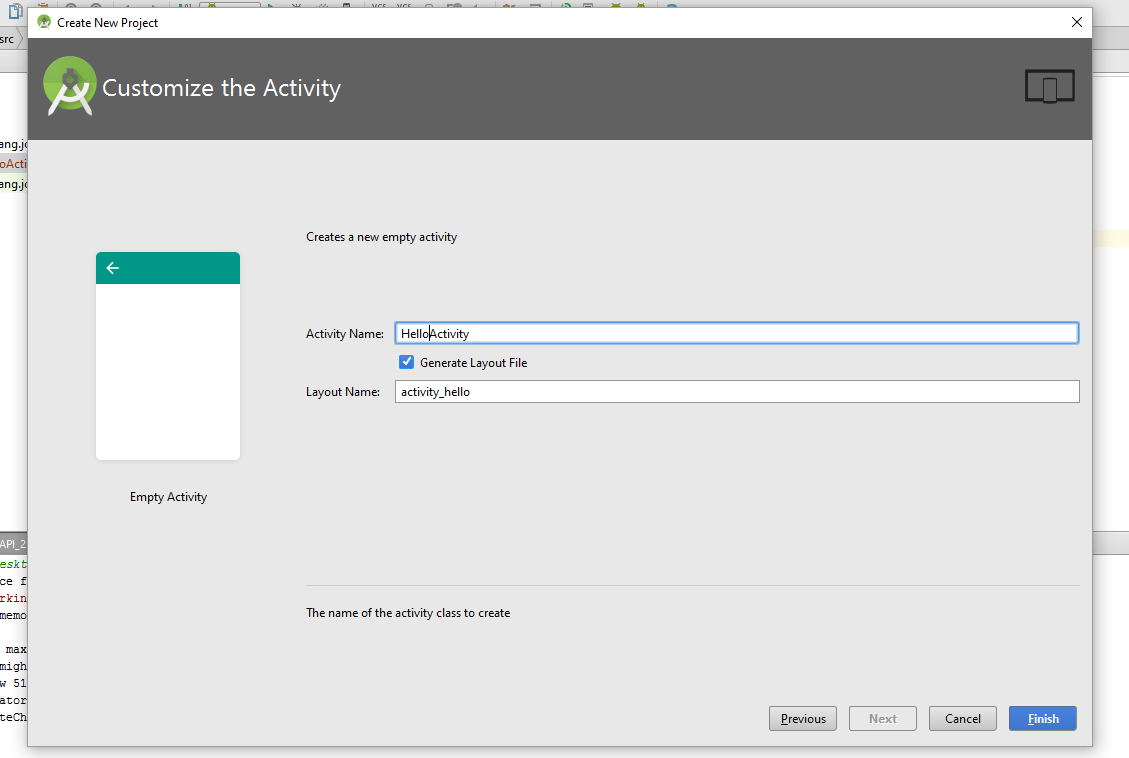
As a result, if developers want to support more people, they’re going to have to develop at a lower level API.

I recommend using API 15 as your baseline. At the time of the writing API 15 (IceCreamSandwich) supports 94% of the whole Google Play Store.

Next you’ll need to select to add an activity to your project. Activities are your java files where you put your code in.



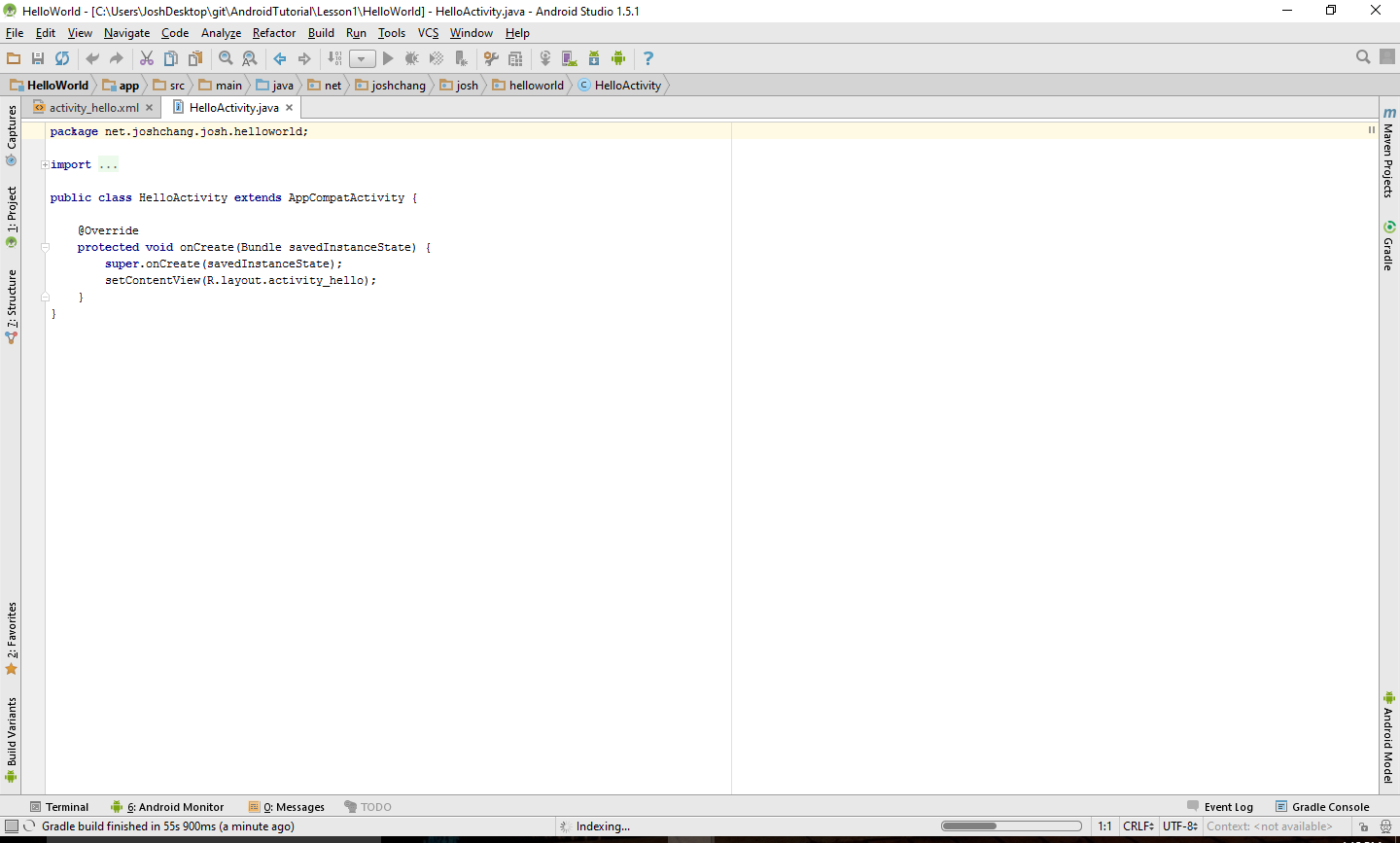
For now just select an **Empty Activity** and you’ll see:

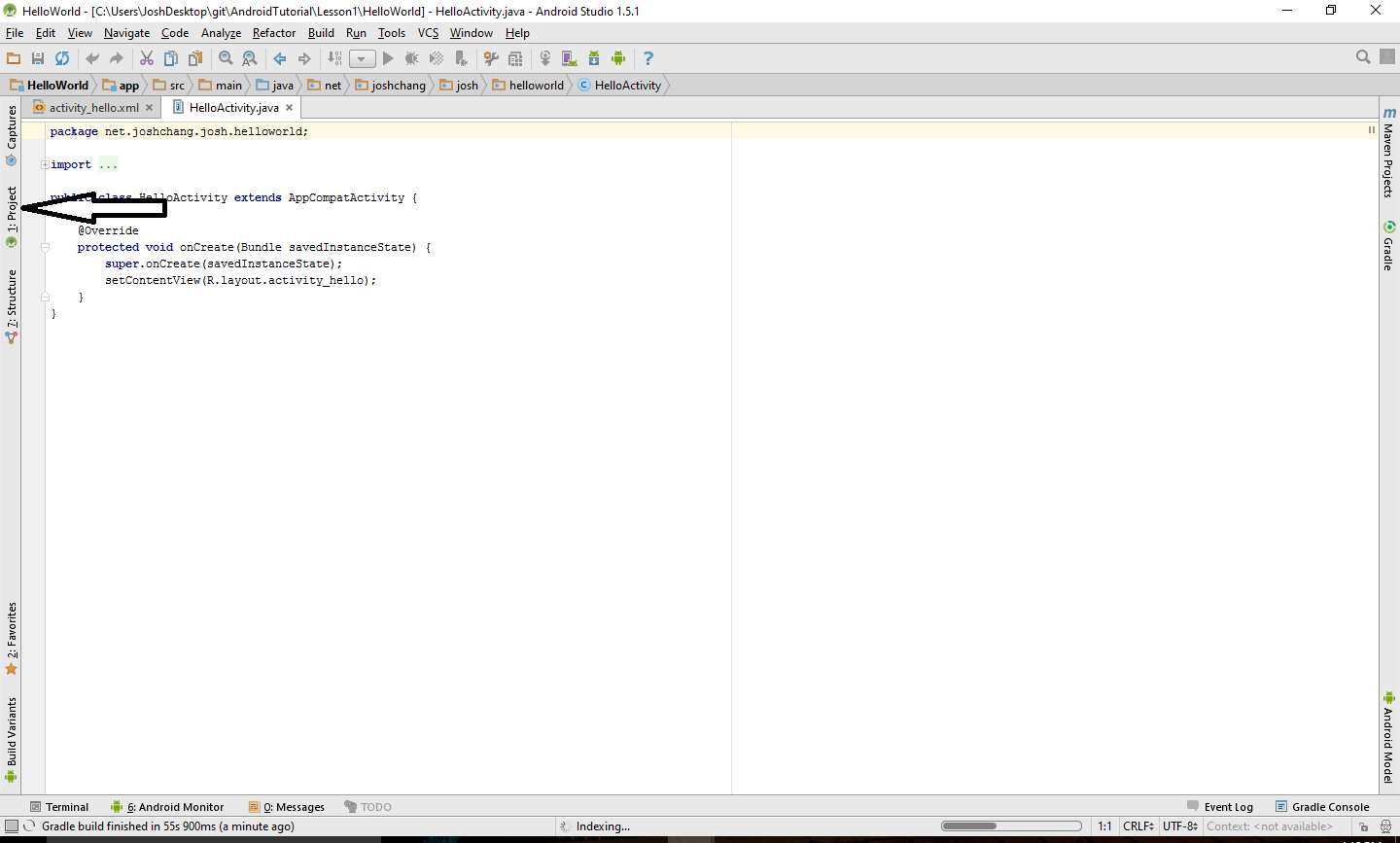
****

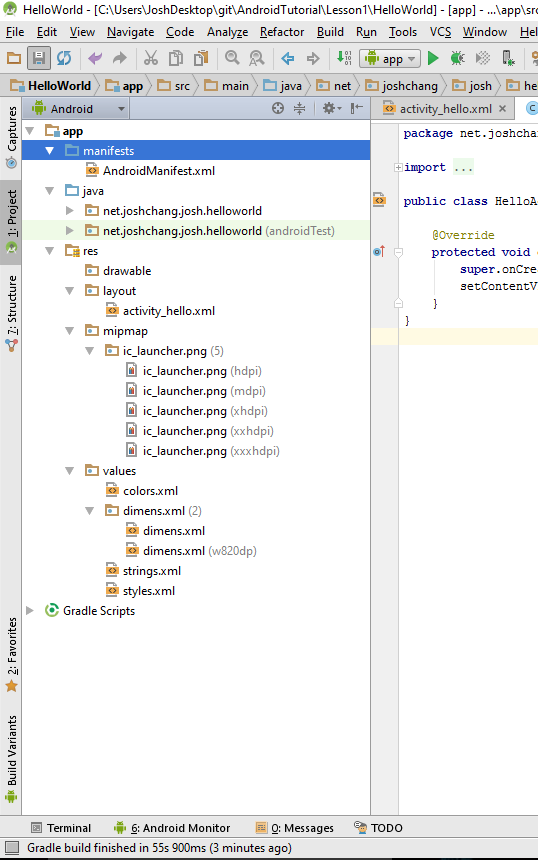
You’ll see 2 fields on the next page:

1. Activity Name – the name of your activity, or your java file where you put your logic. We’re just calling it **HelloActivity**.
2. Layout Name – Every activity will have a layout associated with it. We’ll talk more about what layouts, are but they’re just the GUI (Graphic User Interface) to your app. The app will change it. Just leave it to be **activity\_hello**

Now that you’re done you should have something like this:

Now if you click on the project bar on the left you’ll see a lot of folders that makes no sense to you at all.





I’ll give a brief overview of what they’re for:

* **manifests** – Holds your AndroidManifiest.xml which is where you add permissions that you need to run your app on your phone. For example: GPS, internet access, etc
* **java** – Inside the java you’ll see your domain name that you selected. There are 2 of them, the one that says androidTest is there for you to write tests for your app and the other folder that contains your TodoActivity is where we put all of our files.
* **res**
  + **drawable –** This is where you put all of your pictures
  + **layout –** Layouts contains the xml files that creates the GUI for each of your activitiy
  + **mitmap –** Similarto drawable except mitmap is where you put your icons
  + **values –** The files in here are used to make your life easier by setting constant values that you can call on later. It might seem like a pain in the butt, but if you ever decide to change anything in the future you don’t have to go throughout your codebase to change everything.
* **Gradle** – We won’t talk much about gradle, but gradle is used to compile and run your code. There are many useful features like using 3rd party libraries, but we won’t get to it in this tutorial.

Now that we have everything up and running let’s understand our code:

**package** net.joshchang.josh.todolist;  
  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
  
**public class** HelloActivity **extends** AppCompatActivity {  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_hello);  
 }  
}

First the class header: you’ll notice that your class extends something called **AppCombatActivity**.

Without going to deep into the technical details, **AppCombatActivity** and other **Activity** classes are the what displays the content of your app.

For now just understand everything you use in Android extends on an already existing class and you’re just overriding their methods to create the features that you want to use.

**A look into the Android Life Cycle**

Inside our class we have an **onCreate** method that’s already provided for us.

The onCreate method is one of the methods in the Android LifeCycle which you can see here:

I’ll talk more about the Life Cycle in the future, but just know that onCreate is the first method your **Activity** when you app opens up.

The first line of code we see is this:

**super**.onCreate(savedInstanceState);

This is us calling the onCreate function of our super class. Because of the complexity of Android we have to initialize our super class so that your app functions as it should. Generally speaking whenever you’re overriding a method in an activity or anywhere else you want call your super class to make sure your app runs properly.

The next line of code is initializing the GUI for your app:

setContentView(R.layout.***activity\_hello***);

**setContentView** is a method of your class that you inherit. What it does is that it chooses one of your **layout** in your res folder to be the GUI for the activity.

Passing into the method, you’re giving <R.layout.activity\_todo> R is a class that contains the path to the contents inside your **res** folder.

For simplicity sakes just think of **R** as **res**.

Every **Activity** class you create you’ll need to create an xml **layout** to display a UI to the user.

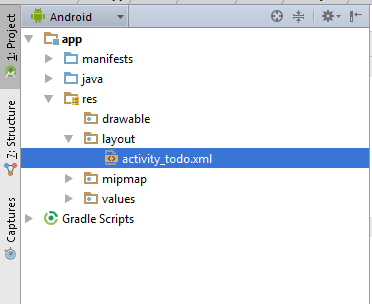
**Note**: Now I know it’s a pain to figure out what methods you inherit especially when Android is completely new to you. But as you continue to develop in Android you’ll come to recognize these methods. Of course you could always look through the [Android Activity Documentation](http://developer.android.com/reference/android/app/Activity.html)!

**First Line of Android Code**

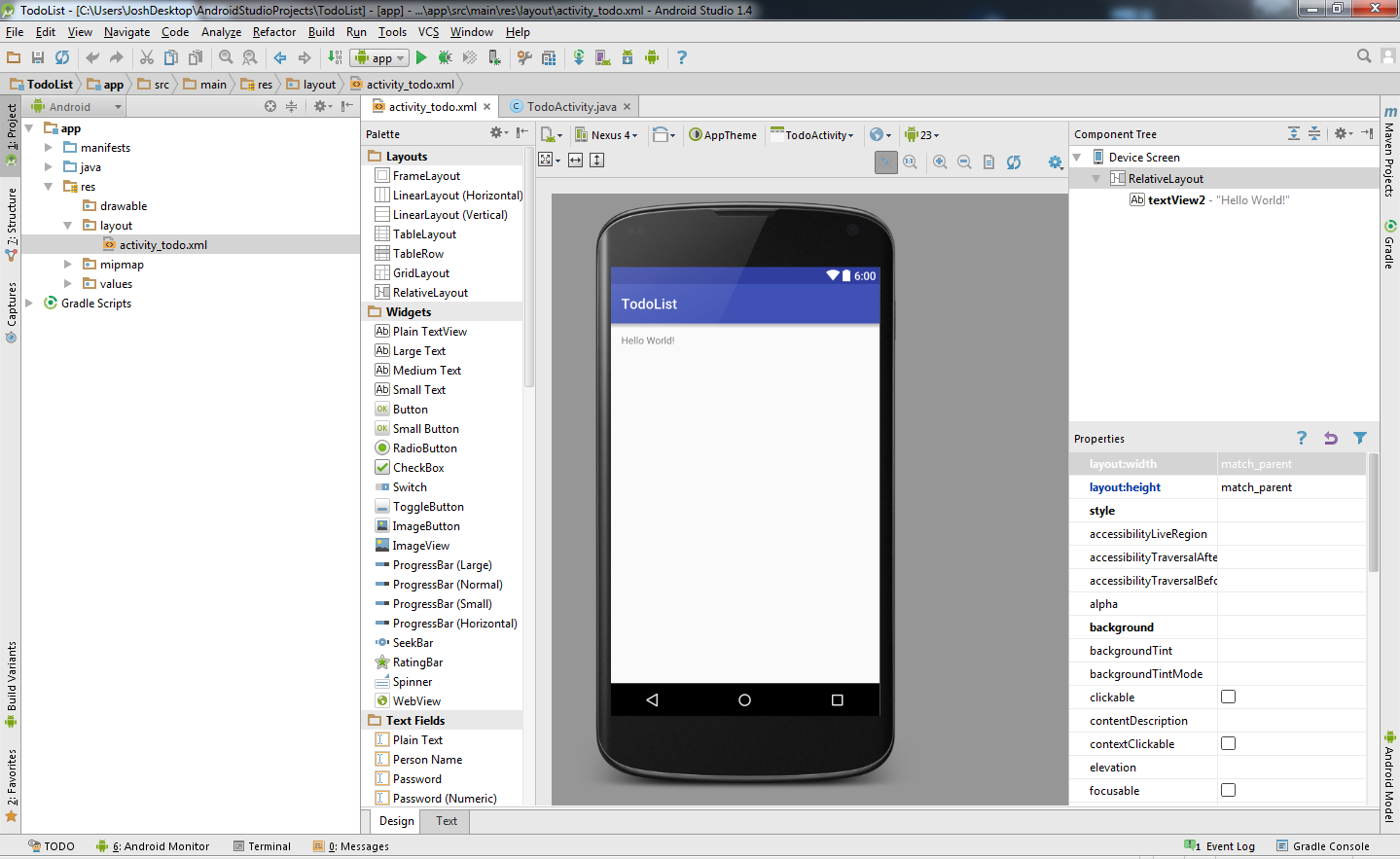
So now that our long winded explanation of 4 lines of code is over, let’s actually start writing code. To create our simple HelloWorld app we’ll need 3 widgets:

* **TextView –** To display our text
* **Button** – To change what’s currently being displayed
* **EditText** – To allow you to type in something new to display

Head over to **app** -> **res** -> **activity\_hello.xml**:

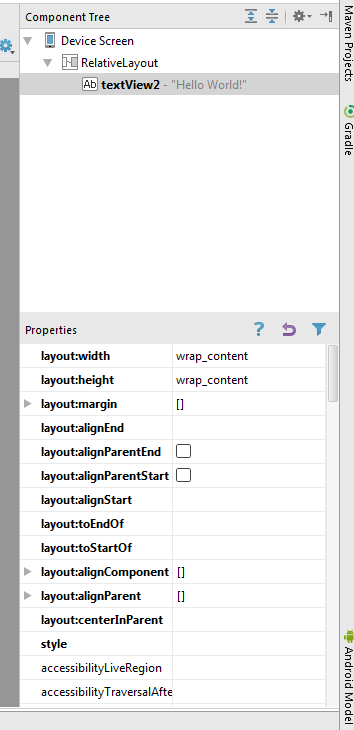


You’ll see something like this:



This is the design editor for your layout. With this, you can easily drag, drop, and edit your widgets without code. It’s really handy for getting something out quickly regardless of your experience.

On the right pane, you’ll see the **Component Tree Pane**. It allows you to see all the properties that your widget contains that you can change.

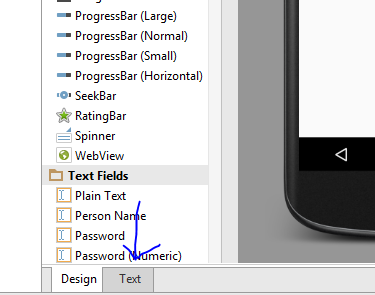
<7PropertyPane>

The more important properties are even highlighted for your convenience.

However if you already know what you want, it might be faster to go in and edit the properties of your widgets directly in your layout xml.

For my convenience we’ll be working with the xml source code, however you can just as easily drag the widgets in the **Design Editor** to get similar results

Click Text at the bottom at left corner of your activity\_todo.xml window and you’ll go to the source code of your xml file:

<8>

Your code should look like this:

*<?***xml version="1.0" encoding="utf-8"***?>*<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools" android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent" android:paddingLeft="@dimen/activity\_horizontal\_margin"  
 android:paddingRight="@dimen/activity\_horizontal\_margin"  
 android:paddingTop="@dimen/activity\_vertical\_margin"  
 android:paddingBottom="@dimen/activity\_vertical\_margin" tools:context=".TodoActivity"** >

*<!-- Note that the padding above is actually using a shortcut, specifically  
 @dimen/activity\_vertical\_margin and @dimen/activity\_vertical\_margin  
 if you don't have these values, you can also hardcode 16dp  
 -->*

<**TextView android:text="Hello World!" android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:id="@+id/textView2"** />  
  
</**RelativeLayout**>

Looks confusing? Don’t worry, after you have made a couple of layouts you’ll realize that it’s not really as complicated as you might think.

In layout folders everything is represented inside a pair of <> and ends with another pair </> such as:

**<Button> </Button>** or you can even do this **<Button />** to open and close the tag

Inside each of these tags you can add attributes inside them to change what your widget will look like.

Overtime you’ll learn more and more widgets and how to use them.

What we have here are:

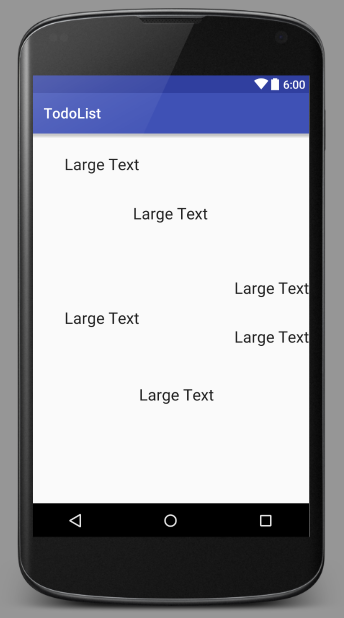
* **Relative Layout –** A container that’s used to hold widgets or other containers
* **TextView –** A widget that allows you to display words

**Lesson On Layouts: Relative Layout vs. Linear Layout**

On a side note, if you have been seen other tutorials you’re bound to encounter **Linear Layouts**. What’s the difference?

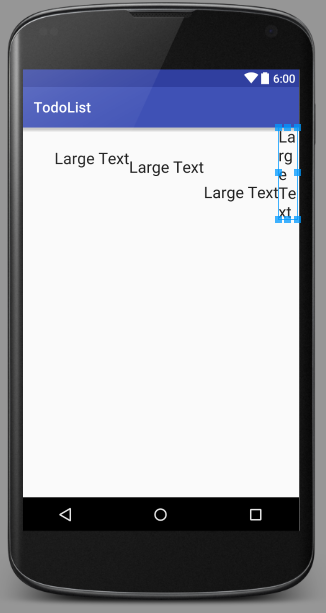
**Relative Layout** allows you to position items anywhere in your container relative to where other widgets or the relative layout are located.

An example:



**Linear Layouts** doesn’t have the feature. Linear layouts are more restricted in that your widgets are ordered linearly either horizontally or vertically where widgets won’t be stacked on each other.

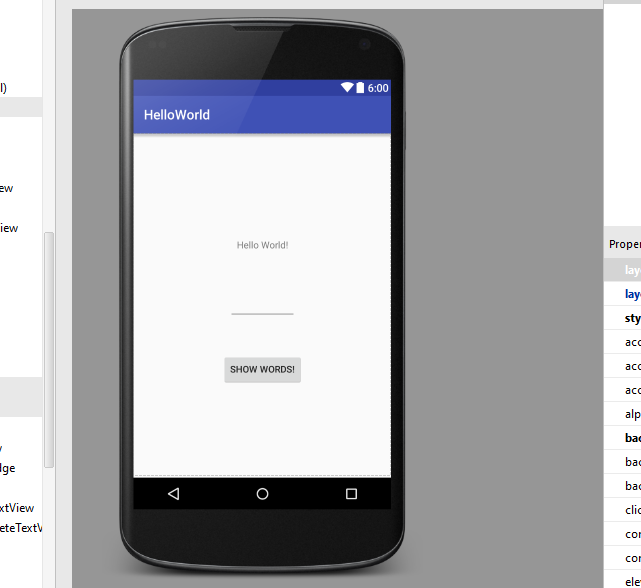
An example of linear layout with the exact same widgets the only difference is that it’s using Linear Layout instead of Relative Layout:



Now for our first line of layout code! We’re going to start adding our components inside our xml.

*<?***xml version="1.0" encoding="utf-8"***?>*<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:paddingBottom="@dimen/activity\_vertical\_margin"  
 android:paddingLeft="@dimen/activity\_horizontal\_margin"  
 android:paddingRight="@dimen/activity\_horizontal\_margin"  
 android:paddingTop="@dimen/activity\_vertical\_margin"  
 tools:context="net.joshchang.josh.helloworld.HelloActivity"**>  
 *<!-- Note that the padding above is actually using a shortcut, specifically  
 @dimen/activity\_vertical\_margin and @dimen/activity\_vertical\_margin  
 if you don't have these values, you can also hardcode 16dp  
 -->* <**TextView  
 android:id="@+id/tvDisplayText"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Hello World!"  
 android:layout\_above="@+id/etShowWord"  
 android:layout\_centerHorizontal="true"  
 android:layout\_marginBottom="60dp"** />  
  
 <**EditText  
 android:id="@+id/etShowWord"  
 android:layout\_width="100dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_centerVertical="true"  
 android:layout\_centerHorizontal="true"** />  
  
 <**Button  
 android:id="@+id/btnShowWord"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Show Words!"  
 android:layout\_below="@+id/etShowWord"  
 android:layout\_centerHorizontal="true"  
 android:layout\_marginTop="50dp"** />  
</**RelativeLayout**>

And you should have something that looks like this:



So what do these properties mean? Well let’s go through them.

* **android:id –** The ID of the widget. This has to be a unique name so you can refer to this specific object in your code.
* **android:layout\_width** and **android:layout\_height** – These attributes describe the width and height of your widget. **wrap\_content** wraps makes the widget as big as the text/size you set. **match\_parent** makes your widget match the size of its container which can either be the layout or the size of your screen
* **android:layout\_alignParentBottom, Top, Left, Right (Only used with Relative Layout) –** Aligns the widget to the locations that you set inside the parent, which can be the layout or screen.
* **android:layout\_toLeftOf, android:layout\_above android:layout\_above –** Aligns the widget next to another widget by using their id.
* android:layout\_marginBottom="60dp" – Moves the object from the bottom up 60dp. Like the other styles, we can move left, right, top, and bottom
* dp – dp is a standard a measurement that’s used in Android so that your app will always look the same no matter the screen size of your user.
* **android:layout\_centerHorizontal="true", android:layout\_centerVertical="true" –** As you might expect, this centers your component horizontally and vertically.

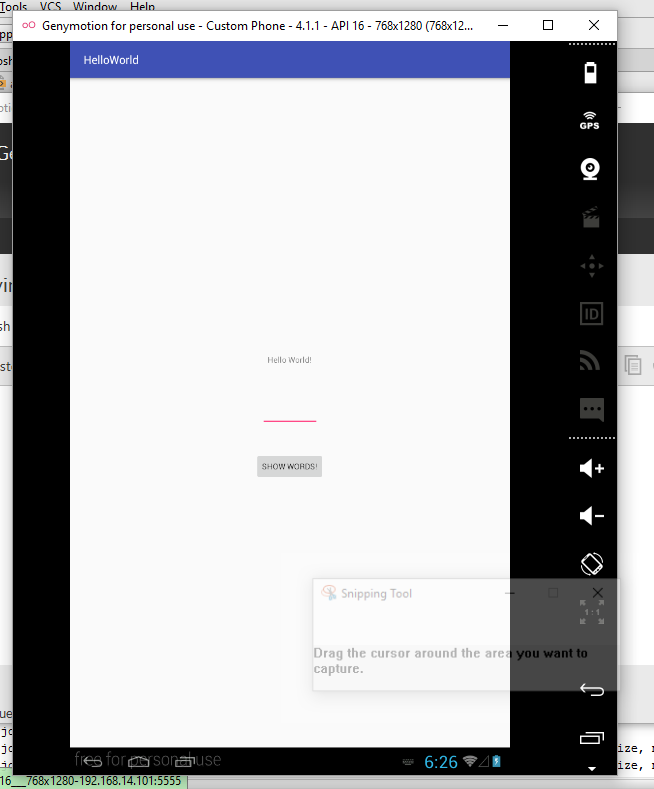
So now it’s time to write some code in our HelloActivity that we looked at before. We’re adding on to what we have here.

**package** net.joshchang.josh.helloworld;  
  
  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.EditText;  
**import** android.widget.TextView;  
  
  
**public class** HelloActivity **extends** AppCompatActivity {  
 *// Use fields so you can access your widgets at any point in the code* **private** EditText **mEtText**;  
 **private** Button **mBtnShow**;  
 **private** TextView **mTvDisplay**;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_hello***);  
  
 *// 1) Gets access to the widget in the layout via their unique ID's* **mEtText** = (EditText) findViewById(R.id.***etShowWord***);  
 **mBtnShow** = (Button) findViewById(R.id.***btnShowWord***);  
 **mTvDisplay** = (TextView) findViewById(R.id.***tvDisplayText***);  
  
 *// 2) Sets a listener to run your code whenever someone clicks  
 // your button.* **mBtnShow**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 String text = **mEtText**.getText().toString();  
 **mTvDisplay**.setText(text);  
 }  
 });  
 }  
}

So let’s step through the new code at each numbered section in the code

1. findViewById(R.id. R.id.***etShowWord***)gets the object inside the layout with the ID etShowWord
   1. R is a shortcut for your res folder, which as you recall, includes styles, Strings, images, and Id’s
   2. It is also important to note that you have to cast the object you found to the appropriate object. If you don’t you won’t be able to use the proper functions of the widget.
2. Here we create an on click listener on our button. What that means is that every time someone clicks our button, we would execute the code we wrote.

Go ahead and run your code on the emulator. You should see something like this.



And there we go! Our simple List app is finished. But before we finish, there are some things that we can do to help make our code a little bit more robust!

**package** net.joshchang.josh.helloworld;  
  
  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.EditText;  
**import** android.widget.TextView;  
  
**public class** HelloActivity **extends** AppCompatActivity {  
 *// Use fields so you can access your widgets at any point in the code* **private** EditText **mEtText**;  
 **private** Button **mBtnShow**;  
 **private** TextView **mTvDisplay**;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_hello***);  
  
 *// 1) Gets access to the widget in the layout via their unique ID's* **mEtText** = (EditText) findViewById(R.id.***etShowWord***);  
 **mBtnShow** = (Button) findViewById(R.id.***btnShowWord***);  
 **mTvDisplay** = (TextView) findViewById(R.id.***tvDisplayText***);  
  
 *// 2) Sets a listener to run your code whenever someone clicks your button.* **mBtnShow**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 *// gets the text that the user added in the Edit Text* String text = **mEtText**.getText().toString();  
  
 *// 3) only set the TextView to the new word if it's not empty* **if** (text.length() != 0) {  
 *// sets the textview to be the new text* **mTvDisplay**.setText(text);  
 *// 4) empty your Edit Text for convenience* **mEtText**.setText(**""**);  
 }  
 }  
 });  
 }  
}

We’ve made 2 changes:

1. We check to see if the user didn’t just give us an empty file and if they did, we don’t do anything
2. Whenever the user inputs something, we empty our Text Edit so that the users don’t have to manually delete it themselves if they wanted to change the text to something new.

If you have any questions or suggestions, please drop a message in the comment!