BASIC OVERVIEW OF AN APP (Refer app MyApplication)

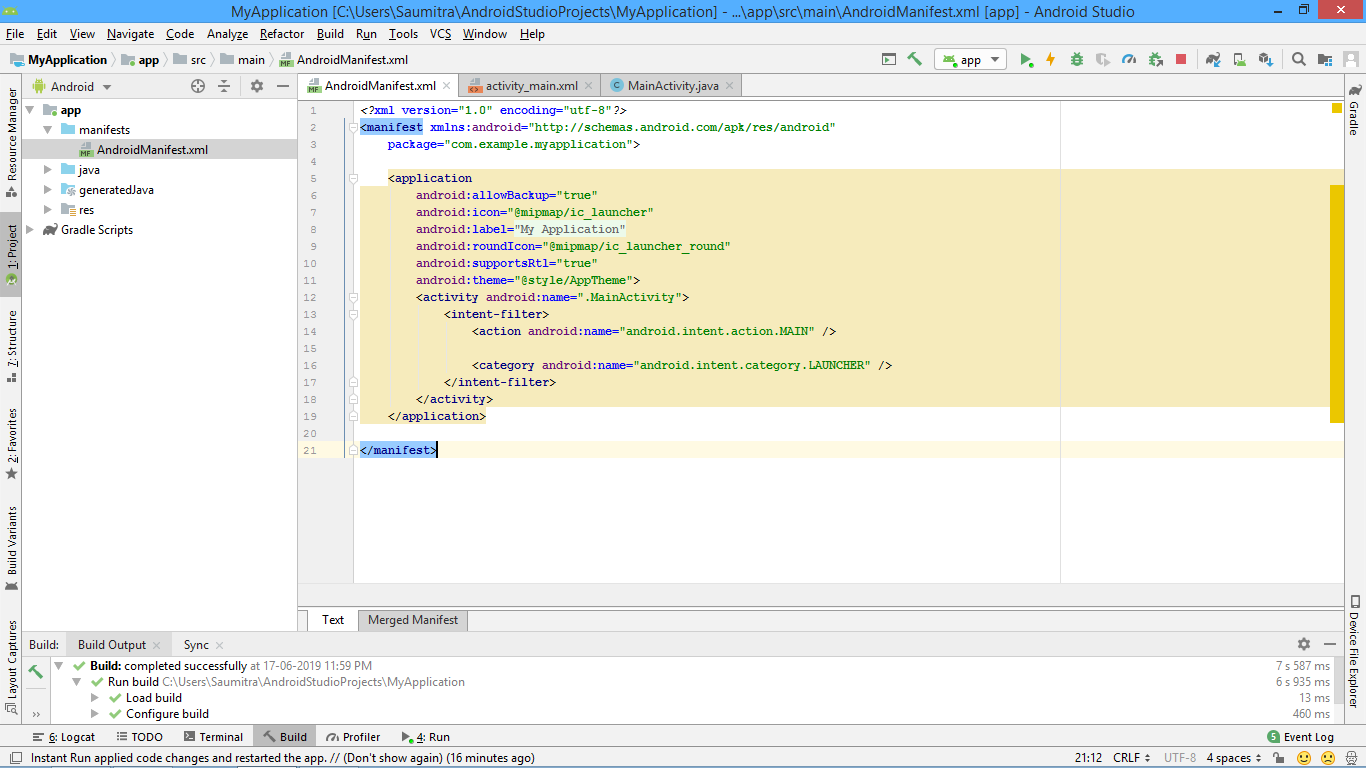
Your app has 3 very basic and very important files:

1>MainActivity.java

2>activity\_main.xml

3>AndroidManifest.xml

Of these, the first file that the phone looks for whenever you are launching an app is the AndroidManifest.xml. This file contains information about what files to refer to in your android app. The following is a screenshot of the AndroidManifest.xml file:



The manifest file contains a record of all the activities in the application.

Here we can see that the manifest file points to an activity which the programmer has defined to be the LAUNCHER. This identification of the launcher activity is defined in the MainActivity.java file. The java file contains a method to show the layout of our app. The layout of the Main Activity which is stored in Activity\_Main.xml is then shown to the user.

Usually the activity which is defined as the LAUNCHER activity is the first thing that the user sees when he clicks the app. Thus, the java file has the code which displays the launcher activity when the onCreate() method is called (i.e. when the app is first launched).

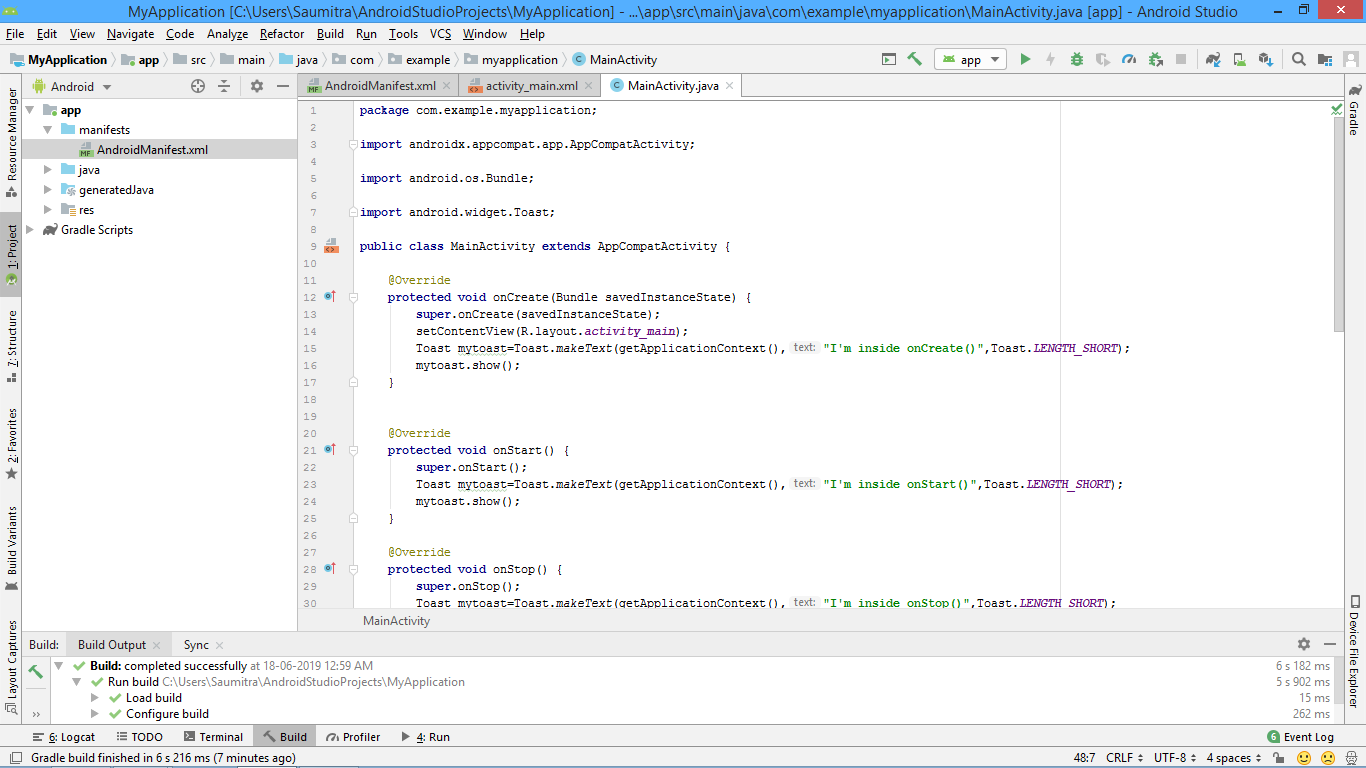
Thus, the manifest file contains the record of which activity to show as the launcher activity, and thus also refers to the java file.

ACTIVITY STATES:

An android application –similar to an applet, has many activity states. All the activities in our program are inherited from a class called the “android.app.Activity” class. This class has a wide variety of standard methods whose declaration has already been provided. These methods can be inherited (overridden) by our program. Some of these methods are executed directly whenever the activity reaches a certain stage.

Many methods of the activity states like onCreate(), onDestroy(), onResume(), etc. can be overridden in order to verify the android activity lifecycle.

Essentials for running the app: <https://www.javatpoint.com/android-toast-example>



<https://www.javatpoint.com/android-life-cycle-of-activity>

(Use alt+insert for code completion)

**Android Activity Lifecycle** is controlled by 7 methods of android.app.Activity class. The android Activity is the subclass of ContextThemeWrapper class.

An activity is the single screen in android. It is like window or frame of Java.

By the help of activity, you can place all your UI components or widgets in a single screen.

|  |  |
| --- | --- |
| **onCreate()** | called when activity is first created. |
| **onStart()** | called when activity is becoming visible to the user. |
| **onResume()** | called when activity will start interacting with the user. |
| **onPause()** | called when activity is not visible to the user. |
| **onStop()** | called when activity is no longer visible to the user. |
| **onRestart()** | called after your activity is stopped, prior to start. |
| **onDestroy()** | called before the activity is destroyed. |

<https://www.youtube.com/watch?v=NMDPxN8FgXM&list=PL6gx4Cwl9DGBsvRxJJOzG4r4k_zLKrnxl&index=9>

<https://www.youtube.com/watch?v=9LbETUPM_sY&list=PL6gx4Cwl9DGBsvRxJJOzG4r4k_zLKrnxl&index=10>

To get a better feel of the activities that occur during the lifecycle of an app, we can create simple android app. In this app, as the user interacts with it, Android studio actually maintains the log of the activity that occurs. We can see the different activities of the app in the Logcat section of android studio. For this, we actually need to import a new package called android.util.Log.

This package lets us interacts with a wide variety of log elements and also lets us create our own log messages whenever a particular function is called.

We then need to create a public static final String variable called TAG. It should be named TAG as it is because it is mentioned so in the documentation of android. We shall set it equal to whatever identifier we want. This variable will serve as the identifier which lets us filter the logcat information.

Now the Log class has a static method called ‘i’ . This method is used to display a log message having a particular identifier. This method accepts 2 parameters—the ‘tag’ of type String which was used to identify our log message, and our actual log message.

By doing so, we can add a new filter in the logcat section of android studio. This filter identifies only those logs which have the TAG which we defined earlier and displays them. This makes it easier to identify whenever the onCreate(), or onResume() functions have been called.

ADDING NEW ACTIVITIES MANUALLY: (REFER APP FruitFly)

Let us start with a new project having NO activity by default. When android studio is done setting up the project, we are left with an application which does not have ANY ACTIVITY. In order to add activities manually, we usually go to the java directory/com.example.MyApplication. Then we right click here and select “Add new activity”. This brings us back to the initialisation screen where we had to choose the activities to add to the project.

We can choose any activity from these, and because currently we are making apps with only ONE ACTIVITY, it is important that we choose the checkbox “Launcher Activity”. This will specify that the current activity is the FIRST thing that the user sees when the app is launched.

MORE ON USER INTERFACE:

We can add widgets to the app either by using the design view of the xml file or using the text view. We can also change many properties of the widgets by using many of the inbuilt attributes. For example, in order to change the width of a particular textedit, we use the attribute:

android:width= “200dp”

Here dp stands for “device pixels”. This is the standard measurement of size across many mobile handsets as well as TV’s.

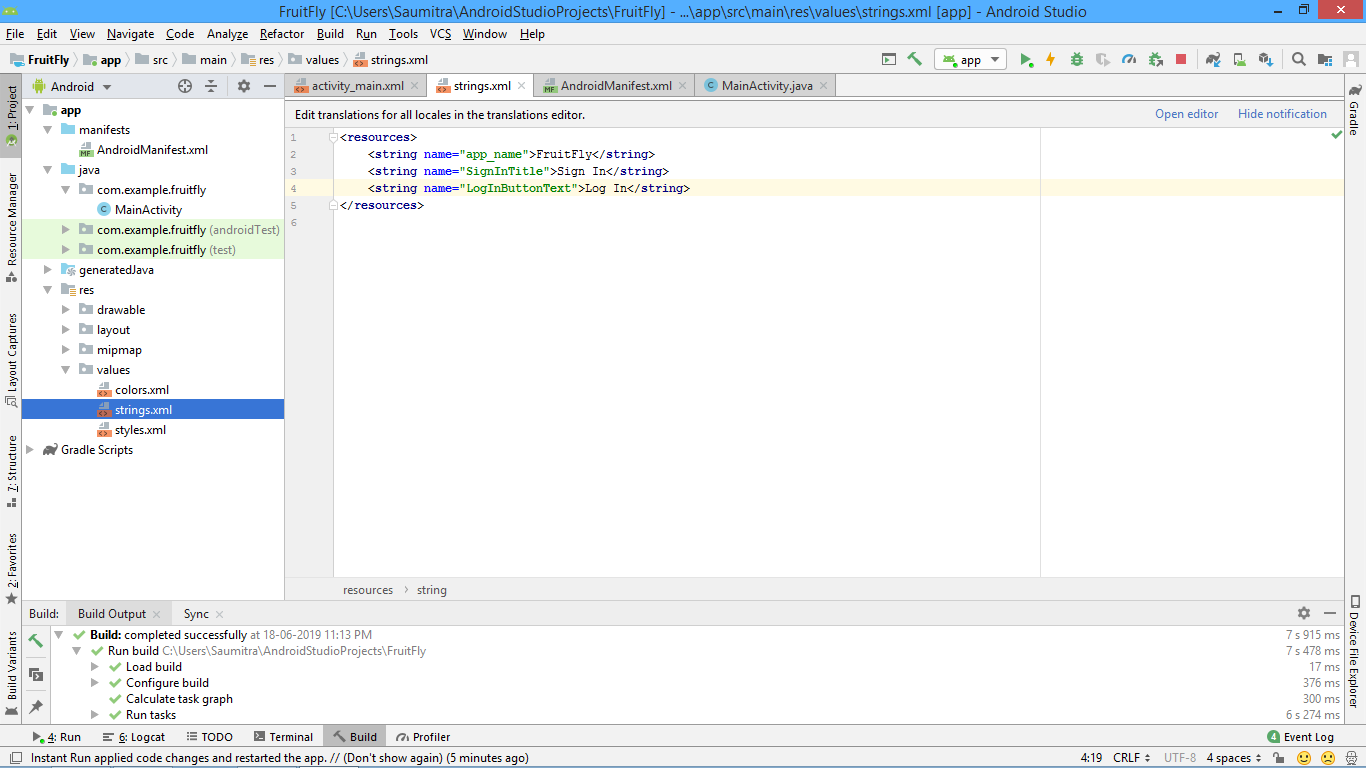
Whenever we change the text string in any widget of the user interface, we need to give a name to that particular string resource. This is essential to do because whenever, we write any string in our app, it gets stored inside a strings.xml file.

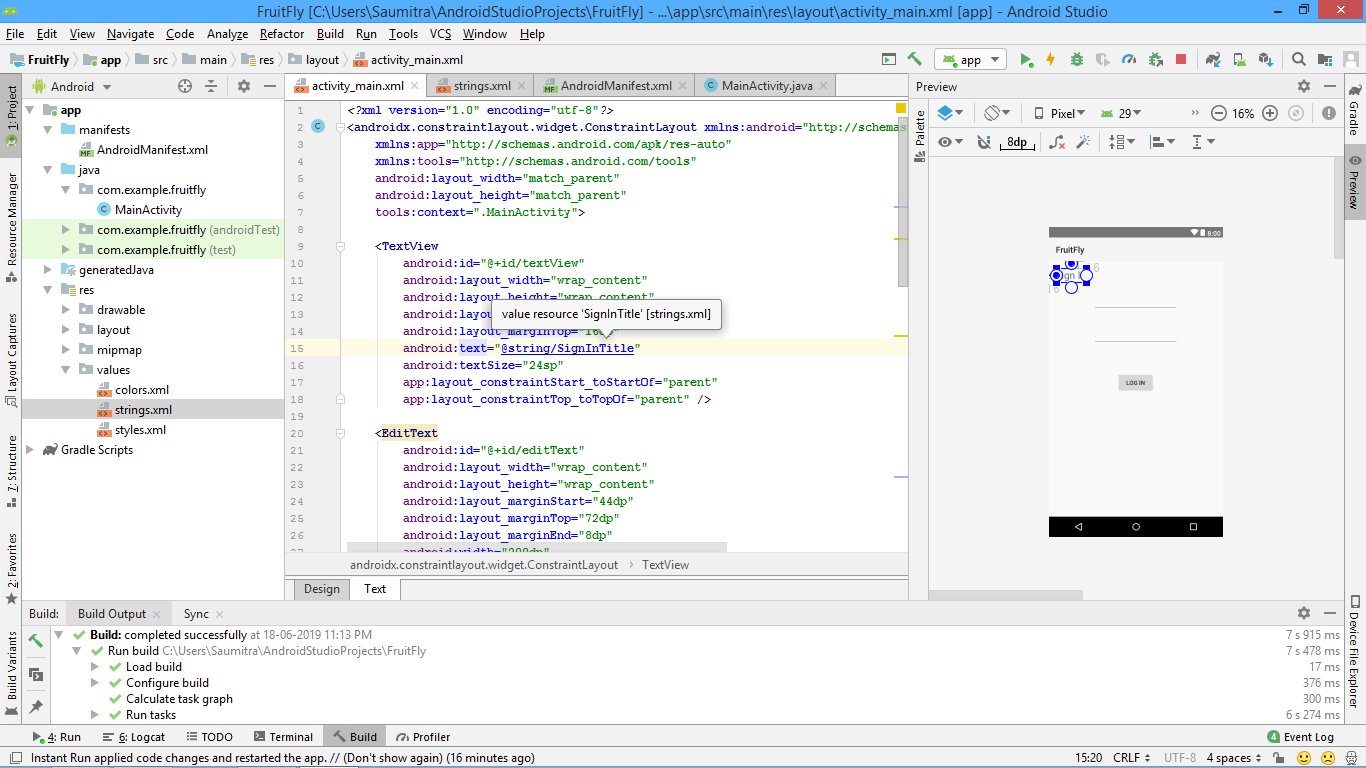
This is essential in those scenarios where suppose the display language of the phone changes to some language other than English, then the os needs to translate all the strings in the app. In order to do so, the os need no search all the activities of the app. It just needs to access only one xml file—strings.xml.

In order to assign string resource, we go to the text view and hover over our string text attribute until we see a lightbulb icon. On clicking this icon, it will give us the option of extracting string resources. We are essentially giving an identifier by which the string can be referred by. All of the strings are present inside the strings.xml file and can by identified by the string resource name.

Whenever we assign names to the string resources, the value of the text field in the xml file will get changed to the resource name, and we will not be able to see the “Actual string”. We can only see the actual string in the strings.xml file.

Thus, in order to locate our string in the strings.xml file, we press ctrl and hover our mouse over the text field. We see that the string attribute now becomes clickable. Upon clicking the attribute, it will redirect us directly to the strings.xml file which contains the actual value of the string.





Ctrl+click here will take you to strings.xml file

CREATING USER INTERFACE WITH JAVA (REFER APP Timber)

Till now we have created the UI with the help of the xml file and the design editor. This approach is very useful whenever you want to create a static layout where the elements don’t usually move much. However, it is possible to create thee layout entirely using Java.

In order to do so, we first need to change our layout in the xml file to the relative layout from the constraintLayout which is there by default.

If we want to switch from the now default ConstraintLayout to RelativeLayout, follow these steps.

1. Go to your activity\_main.xml file 2)
2. Switch from the design tab to the text tab so you can see the source code 3)
3. find this text at the top <android.support.constraint.ConstraintLayout xmlns:android="[http://schemas.android.com/apk/res/android](https://www.youtube.com/redirect?redir_token=w5IngdPeDk8bb52-_68MWvCCWht8MTU2MDk2NzU2NUAxNTYwODgxMTY1&stzid=UginlGsNBhf0RngCoAEC&event=comments&q=http%3A%2F%2Fschemas.android.com%2Fapk%2Fres%2Fandroid)" and replace it with this <android.widget.RelativeLayout xmlns:android="[http://schemas.android.com/apk/res/android](https://www.youtube.com/redirect?redir_token=w5IngdPeDk8bb52-_68MWvCCWht8MTU2MDk2NzU2NUAxNTYwODgxMTY1&stzid=UginlGsNBhf0RngCoAEC&event=comments&q=http%3A%2F%2Fschemas.android.com%2Fapk%2Fres%2Fandroid)" also replace the closing tag at the bottom with this </android.widget.RelativeLayout>
4. You should now be using RelativeLayout

Then, we need to import 2 packages—

android.widget.RelativeLayout

android.widget.Button

We are currently creating a program which will just show a button on the screen. We first need to look at the onCreate() method. This method has a function call to setContentView. For the time being, we just comment that.

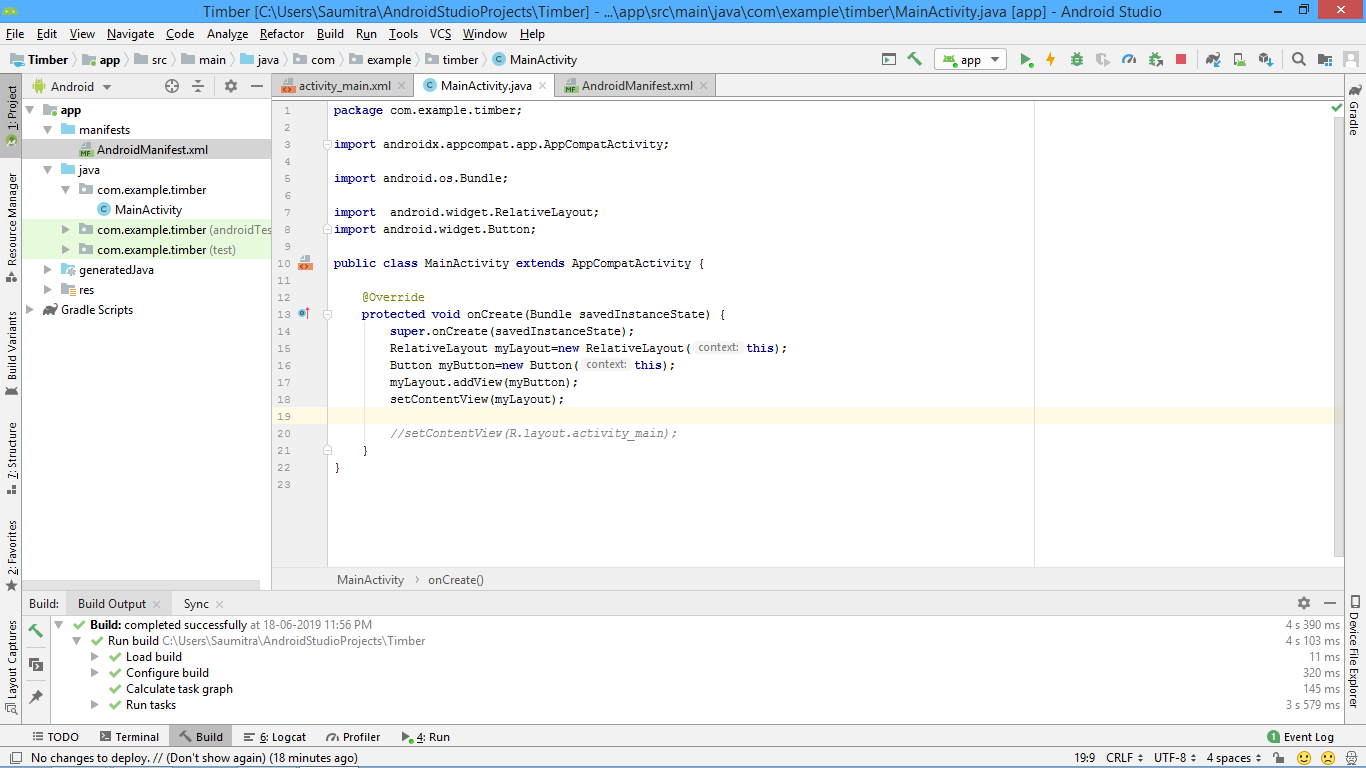
We first need to create a RelativeLayout where all the different widgets can be arranged. In order to do so, we create a new object of the RelativeLayout class.It takes the current context as the parameter in the constructor. Similarly, we also need to create an object of the Button class which will serve as the button on the display. It also takes the current context as the parameter in the constructor.

RelativeLayout myLayout=new RelativeLayout(this);

Button newButton=new Button(this);

Now, the RelativeLayout and the Button currently exist as 2 separate entities. We need to connect the button to the RelativeLayout. In order to do so, we use a method of the RelativeLayout class called addView().This method takes in an object of a class in the android.widget package. This method essentially, adds the widget to the layout.

After adding the widget, we also need to display the layout to the user, whenever he opens the app. This can be done with the help of an inherited method called setContentView(). It takes an object of the a class belonging to the View package .This view package contains many commonly used layouts.



In order to position the button, we need to create a container and assign the layout parameters to it. Then, we need to position the container at a specific location. Finally, we need to add the button to the container.

In order to create a container, we first create a LayoutParams object and assign the size of the container as parameters to its constructor.

RelativeLayout.Layoutparams buttonDetails=new RelativeLayout.Layoutparams(

RelativeLayout.LayoutParams.WRAP\_CONTENT

RelativeLayout.LayoutParams.WRAP\_CONTENT

);

We then assign the position to this container. This can be done with the help of the addRule() method.

buttonDetails.addRule(RelativeLayout.CENTER\_HORIZONTAL);

buttonDetails.addRule(RelativeLayout.CENTER\_VERTICAL);

Finally, we assign the button to this container in the RelativeLayout with the help of addView method.

RelativeLayout.addView(myButton,buttonDetails);

We can also do a similar thing with another widget and position it RELATIVE to this one. Suppose we want to add an EditText widget. For this we’ll have to import the package android.widget.EditText.

Now we create a new instance of the widget.

EditText myText=new EditText (this);

In order to position the EditText RELATIVE to the button, we need to assign id’s to the widgets. This can be done with the help of the setId() method. Thus:

myButton.setId(1);

myText.setId(2);

We again create a new container for the EditText and then assign the size of the container as the parameters to the constructor.

RelativeLayout.Layoutparams textDetails=new RelativeLayout.Layoutparams(

RelativeLayout.LayoutParams.WRAP\_CONTENT

RelativeLayout.LayoutParams.WRAP\_CONTENT

);

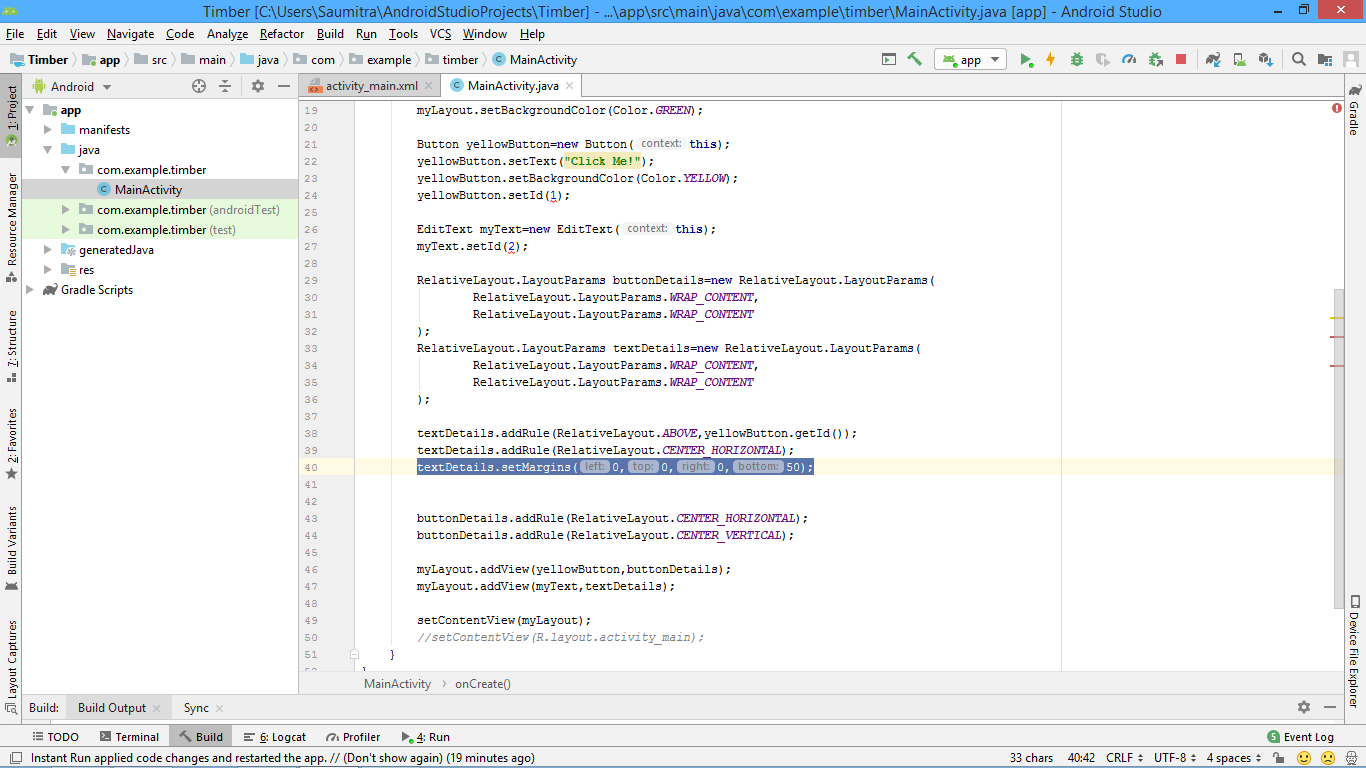
Now we have to position this EditText RELATIVE to the button. This can be done by adding rules to the layoutparameters. The first rules that we’ll add is to position the EditText ABOVE the button. This can be done by specifying the ID of the button ABOVE WHICH we have to place the EditText.

textDetails.addRule(RelativeLayout.ABOVE,myButton.getId());

textDetails.addRule(RelativeLayout.CENTER\_HORIZONTAL);

Finally, we add margins to the EditText so that it is slightly ABOVE the button.

textDetails.setMargins(0,0,0,50);



Finally, we assign the EditText to this container in the RelativeLayout with the help of addView method.

RelativeLayout.addView(myText,textDetails);

CONVERTING FROM DEVICE INDEPENDENT PIXELS (DIP) TO PIXELS:

Now, in order to change the width of the EditText box, we need to use a method called setWidth. However, this method only accepts pixel values. This might create a discrepancy whenever we are working with devices having different display sizes. In order to avoid this discrepancy we need to use DIP (Device/Density Independent Pixels).

As stated earlier, the setWidth method only accepts ‘pixel’ values and not DIP. Thus, we need to convert from DIP to pixels, so that the method can be applied. In order to do so, we need to import 2 new packages—android.content.res.resources

android.util.TypedValue.

Now we first need to get the dimensions of the device that we’re working on. In order to do so, we first create a new reference variable of the resources class and set it equal to the value returned by the method getResources().

Resources r=getResources();

Now, we need to create an int variable which stores the int pixel value. We also need to typecast the RHS to int so that the result returned by the converter method is always int.

Now we use the method TypedValue.applyDimension. This method takes 3 parameters.

1>The unit that we want to convert from

2> The value of DIP that we want to convert.

3>The display details of the target device

**int** pixels=(**int**)TypedValue.*applyDimension*(TypedValue.***COMPLEX\_UNIT\_DIP***,200,r.getDisplayMetrics());

The value in pixels is stored in the variable ‘pixels’

Then finally:

myText.setWidth(pixels);

GRID LAYOUT (Refer App: Fragment)

<https://www.youtube.com/watch?v=4bXOr5Rk1dk&list=PL6gx4Cwl9DGBsvRxJJOzG4r4k_zLKrnxl&index=17>

For Android Studio 3.0, the following steps should work: 1. Adding GridLayout: As per the video.

2. GridLayout wrap\_content: In the Attributes for the GridLayout, set layout\_width and layout\_height to wrap\_content. (There should be a drop-down option.)

3. Adding CENTER button: Drag the Button onto the GridLayout within the Component Tree. Text can be set from the text field in Attributes.

4. Adding the RIGHT button: Drag the Button onto the CENTER button added previously, within the Component Tree.

5. Adding the LEFT button: Drag the button onto the GridLayout in the Component Tree. This should insert the button before the CENTER button.

6. Adding the BOTTOM button: Drag the Button onto the RIGHT button in the Component Tree. This will put it on the same row in the 4th column. To change this, select the GridLayout and view all attributes. Set columnCount to 3. BOTTOM should now be wrapped to the next row.

7. Column / row spans: The attributes are called layout\_rowSpan and layout\_columnSpan.

8. Layout gravity: As per the video.

EVENT HANDLERS, EVENT LISTENERS AND CALLBACK METHODS: (Refer app :Rocket)

Till now, we have seen apps with a wide variety of user interface elements, but these were not so interactive to the user. Here, we shall see how to make UI elements LISTEN for events, and create CALLBACK methods which the environment can execute whenever the said event has occurred.

We shall be creating a simple app with a button and a textview. Whenever the user clicks on the button, the text above it changes. In order to do so, we first need to import a few classes:

import android.view.View

import android.widget.Button

import android.widget.TextView

The View class monitors the functionality of all the widgets inside the app. It therefore is a very important class.

Before, we start writing the java code, we first need to assign id’s to out widgets (i.e. the button and the textview).This can be done by changing the id attribute in the ‘All Attributes’ section of the corresponding widget. The id is the thing that the java code can identify the widget with, and implement different operations on it by treating it like an object.

Now, we need to create an object of the Button class and identify it by the id that we set earlier. This way we can execute methods for our button. This can be done by:

Button btngo= findViewById(R.id.***myButton***);

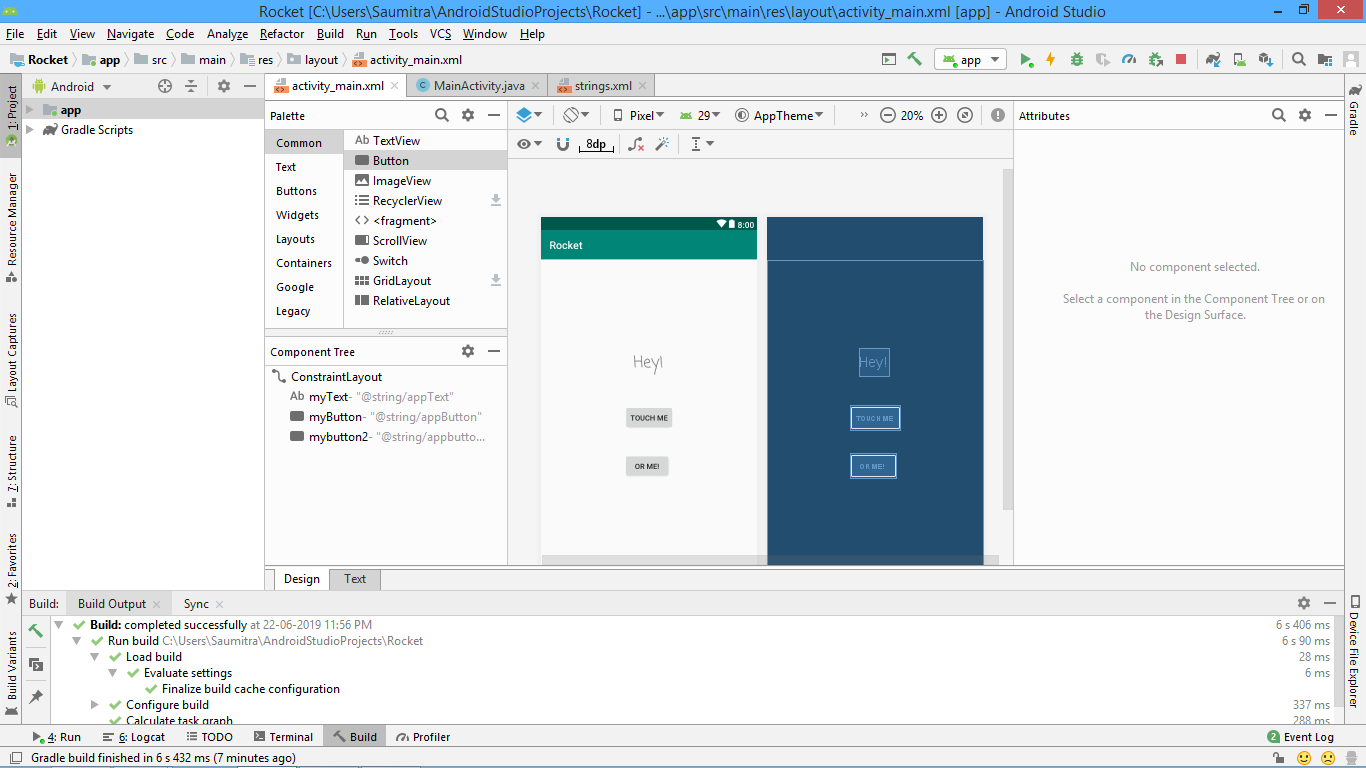
Now, we need to “tell” the button to wait and listen for a click. This can be done with the help of a method called ‘setOnClickListener’. This method accepts an Interface which tells the button what to do once it is successfully clicked. This method will be our callback method.

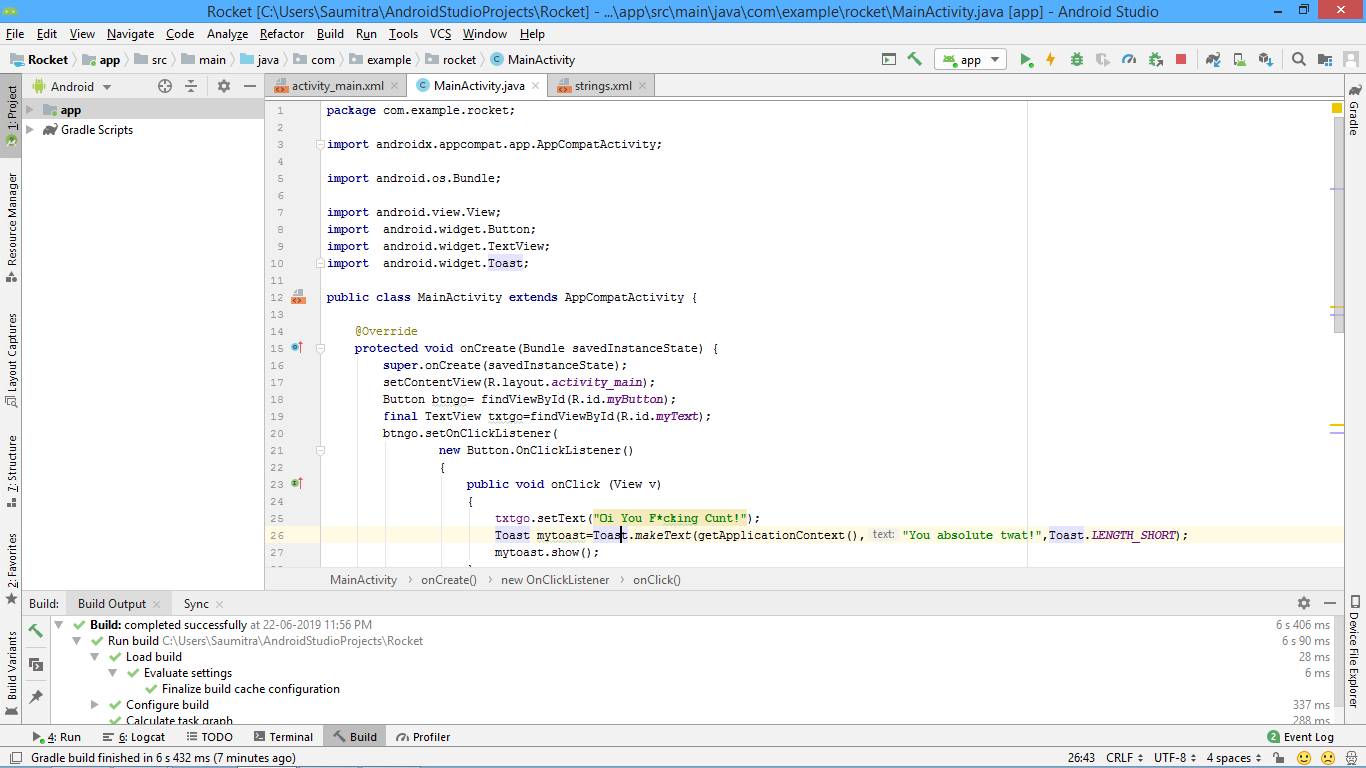
The java code can be written as follows:

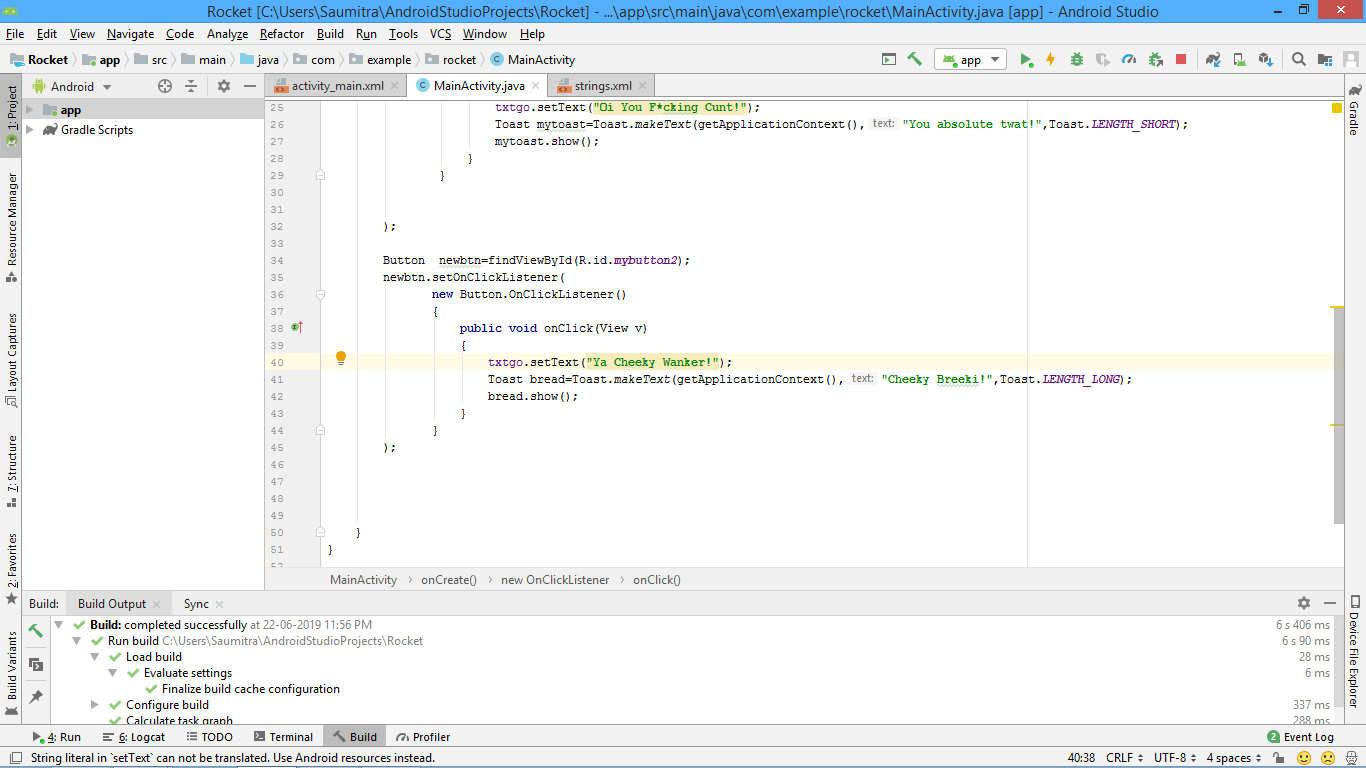
btngo.setOnClickListener(  
 **new** Button.OnClickListener()  
 {  
 **public void** onClick (View v)  
 {

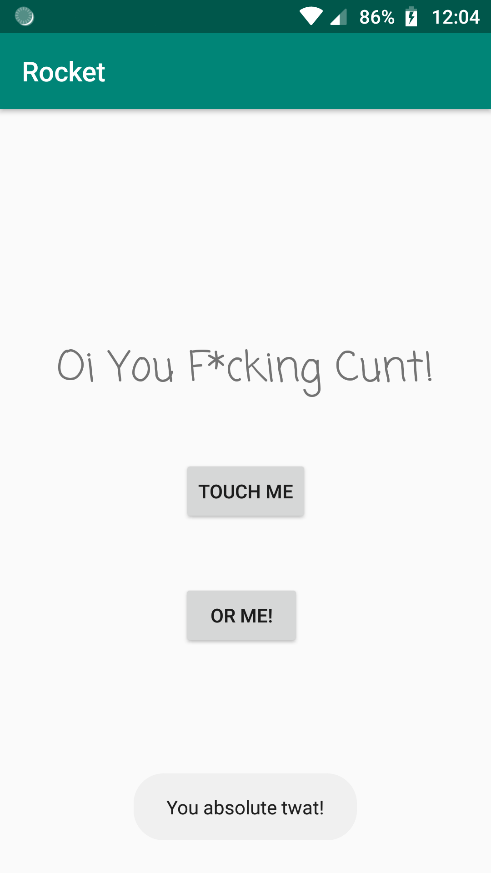
TextView txtgo=findViewById(R.id.***myText***);  
 txtgo.setText(**"Oi You F\*cking Cunt!"**);  
 Toast mytoast=Toast.*makeText*(getApplicationContext(),**"You absolute twat!"**,Toast.***LENGTH\_SHORT***);  
 mytoast.show();  
 }  
 }  
  
);

We see that the name of the method should COMPULSORILY be onClick and it should COMPULSORILY accept a reference variable to the View class as the parameter.







On Clicking this

On Clicking this

MULTIPLE EVENT LISTENERS:

We can also tell a widget (say a button) to listen for multiple events. This can be done be setting multiple event listeners for a button. The process is very similar to that of an onClickListener. Here we shall set an onLongClickListener. Thus, the button will listen if the user clicks the button as well as when the user clicks and HOLDS the button. A different action will be executed then.

Let us see the code for setting up an onLongClickListener().

btngo.setOnLongClickListener(  
 **new** Button.OnLongClickListener()  
  
 {  
 **public boolean** onLongClick(View v)  
 {  
 txtgo.setText(**"AAAH! Leave me or I'll beat the shit out of you!"**);  
 Toast rottenegg=Toast.*makeText*(getApplicationContext(),**"You mankey daft arse-licker!"**,Toast.***LENGTH\_SHORT***);  
 rottenegg.show();  
 **return false**; *//Returning false will temporarily execute onLongClick method and on release will switch back to onClick* }  
 }  
  
  
);

We see that the code is very similar to the original onClick code except for ONE IMPORTANT DIFFERENCE. We see that the our callback method: public boolean onLongClick(View v) returns BOOLEAN VALUE. Let us see what happens when we return ‘true’ and ‘false’ from the method.

1. RETURNING ‘TRUE’:

Whenever we return true from the method, it signifies that the onLongClick event has been handled successfully. Thus, the runtime environment DOES NOTHING whenever the user releases from the long click. The event has been caught and resolved. Again when the user simply touches the button the onClick method will be executed and STOPPED. Similarly when the user touches and holds the button, the onLongClick method will be executed and STOPPED. In both cases the event has been successfully handled.

1. RETURNING ‘FALSE’:

Whenever we return true from the method, it signifies that the onLongClick event has been handled unsuccessfully. Thus, the runtime environment SWITCHES BACK TO THAT METHOD WHICH SUCCESSFULLY HANDLES THE EVENT (in this case, the onClick method) whenever the user releases from the long click. Thus, the event has been caught but has again been thrown to another catch handler which successfully resolves the event.

When the user simply touches the button the onClick method will be executed and STOPPED. But, when the user touches and holds the button, the onLongClick method will be executed and on release, the onClick method will be executed.

GESTURES (REFER APP: ChickenNugget)

<https://developer.android.com/training/gestures/detector#detect>

<https://developer.android.com/reference/android/view/GestureDetector.OnGestureListener.html#onDown(android.view.MotionEvent)>

Similar to onClickListeners, we can also add gesture detectors to our app. Gestures are simply actions that you do on the screen rather than on a particular widget. It includes single tap, double tap, scroll (scroll and hold), fling (quick scroll and release), single tap up,etc.

Here we are building an app which displays what gesture you are currently performing .

In order to do so, we need to import a few standard packages viz:

**import** android.widget.TextView;  
**import** android.view.MotionEvent;  
**import** android.view.GestureDetector;

**import** androidx.core.view.GestureDetectorCompat;

All of the above packages are necessary in order to in order to detect (i.e LISTEN FOR) gestures as well as to specify the callback function which will perform an action in response to the function. In order to do so, we need our MainActivity class to implement certain Interfaces which we have imported in the packages. These interfaces are actually LISTENERS for the gestures which we want the app to detect.

This might seem familiar to the onClickListeners that we have used earlier, wherein we manually defined the Interfaces and wrote the callback function manually. The advantage of implementing such listener interfaces in our MainActivity class is that all the standard methods which absolutely and compulsorily must be overidden can be written using the Code AutoComplete feature of Android Studio.

Thus we shall just add the keywords: **implements**GestureDetector.OnGestureListener,GestureDetector.OnDoubleTapListener

Which will make the MainActivity class implement the required functions. We then create reference variables for the TextView as well as the GestureDetectorCompat classes. The GestureDetectorCompat object detects various gestures and events using the supplied MotionEvents. The GestureDetector.OnGestureListener callback will notify users when a particular motion event has occurred. This class should only be used with MotionEvents reported via touch (don't use for trackball events).

This compatibility implementation of the framework's GestureDetector guarantees the newer focal point scrolling behavior from Jellybean MR1 on all platform versions.

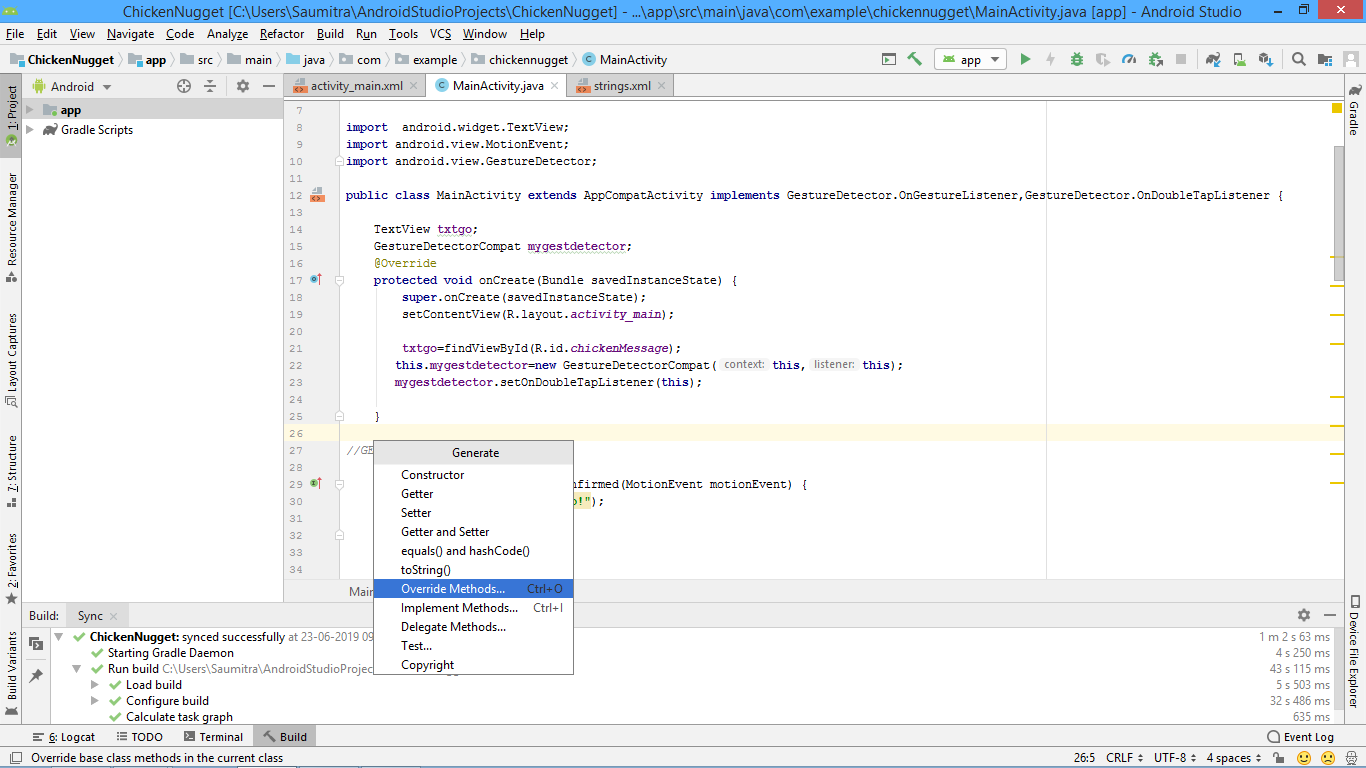
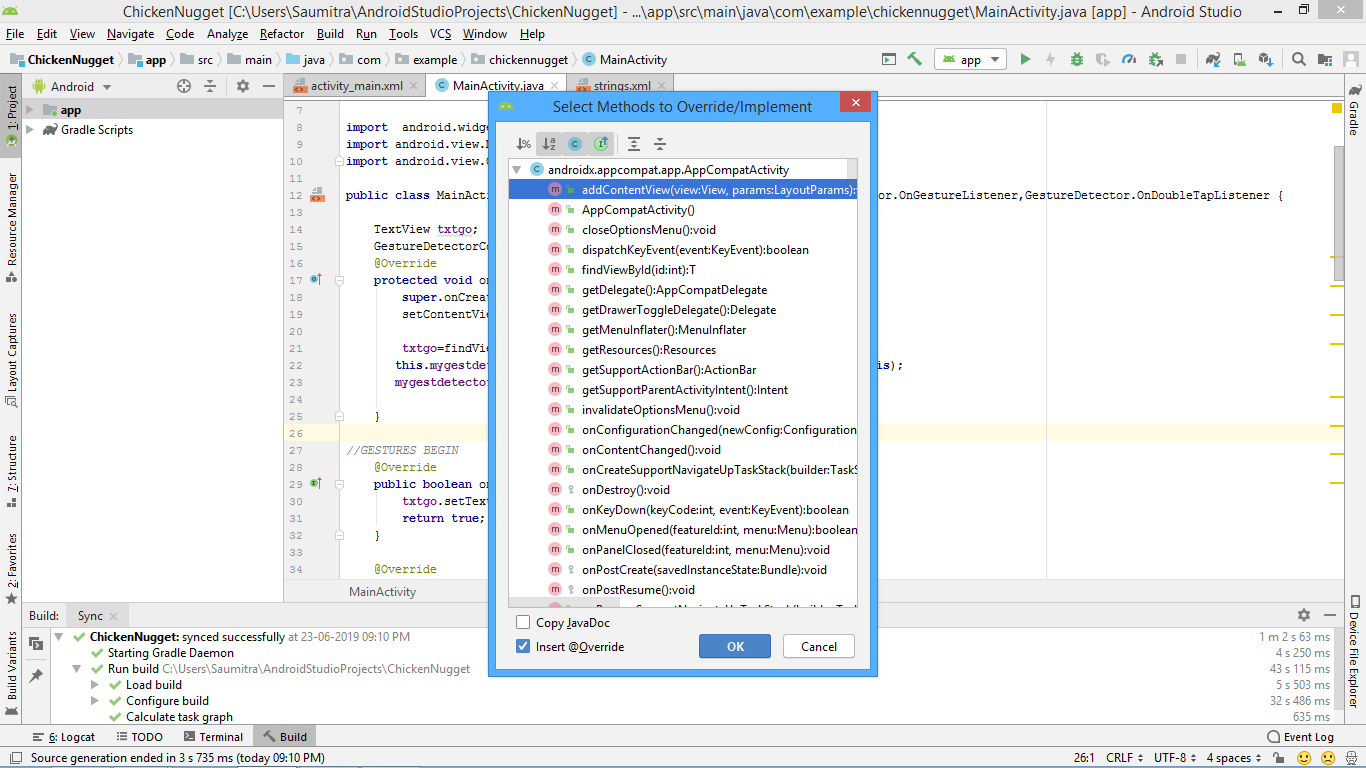
To use this class:

Create an instance of the GestureDetectorCompat for your View (which we have done)

In the View.onTouchEvent(MotionEvent) method ensure you call onTouchEvent(MotionEvent). The methods defined in your callback will be executed when the events occur.

TextView **txtgo**;  
GestureDetectorCompat **mygestdetector**;  
@Override  
**protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
  
 **txtgo**=findViewById(R.id.***chickenMessage***);  
 **this**.**mygestdetector**=**new** GestureDetectorCompat(**this**,**this**);  
 **mygestdetector**.setOnDoubleTapListener(**this**);  
  
}

We have created the objects in the onCreate method of the MainActivity class. After that we press Alt+insert, this brings up small menu. We select Override Methods option from the list which will bring up a window showing all the methods which we must compulsorily override in order to successfully implement the selected interfaces.

We change the return types of the methods we want to use to ‘TRUE’ indicating that the occurred event has been successfully handled. Otherwise runtime environment will continue to switch to that event hander (OR LISTENER) which successfully returns ‘TRUE’ and thus confirms that the event has been handled.

Finally we also need to override another method in our onCreate method which is the onTouchEvent method. This method accepts an object of the MotionEvent class (which detects ANY KIND OF MOVEMENT like trackball, pen, touch, ect). onTouchEvent method however will detect ONLY TOUCH EVENTS on the screen. We therefore pass the touch event that is present as a parameter in our method into the onTouchEvent method of the GestureDetectorCompat object.

We are essentially asking the method to verify whether whatever touch that just occurred is a gesture or not. This is the reason we are actually overriding this method. If is not one of the many gestures that we have defined, then it will continue its normal functionality.

@Override  
**public boolean** onTouchEvent(MotionEvent event) {  
  
 this.**mygestdetector**.onTouchEvent(event);  
  
 **return super**.onTouchEvent(event);  
}

### onTouchEvent

Added in [API level 1](https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels)

public boolean onTouchEvent ([MotionEvent](https://developer.android.com/reference/android/view/MotionEvent.html) event)

Implement this method to handle touch screen motion events.

If this method is used to detect click actions, it is recommended that the actions be performed by implementing and calling [performClick()](https://developer.android.com/reference/android/view/View.html#performClick()). This will ensure consistent system behavior, including:

* obeying click sound preferences
* dispatching OnClickListener calls
* handling [AccessibilityNodeInfo#ACTION\_CLICK](https://developer.android.com/reference/android/view/accessibility/AccessibilityNodeInfo.html#ACTION_CLICK) when accessibility features are enabled

