# Designing User Interface

The user interface is a system by which users interact with the application. It can be designed using one of two methods: **procedural** and **declarative**. Procedural simply means in code. You write code to create and manipulate all the user interface objects. The following example shows the procedural means for creating and manipulating UI objects using java code.

//getting the main layout

LinearLayout mainLayout = (LinearLayout) findViewById(R.id.main\_layout);

//finding the view

TextView txtHelloWorld = (TextView)findViewById(R.id.hello\_world);

//setting the text string value

txtHelloWorld.setText("Hello CET");

//text color for text view

txtHelloWorld.setTextColor(Color.CYAN);

//creating a button

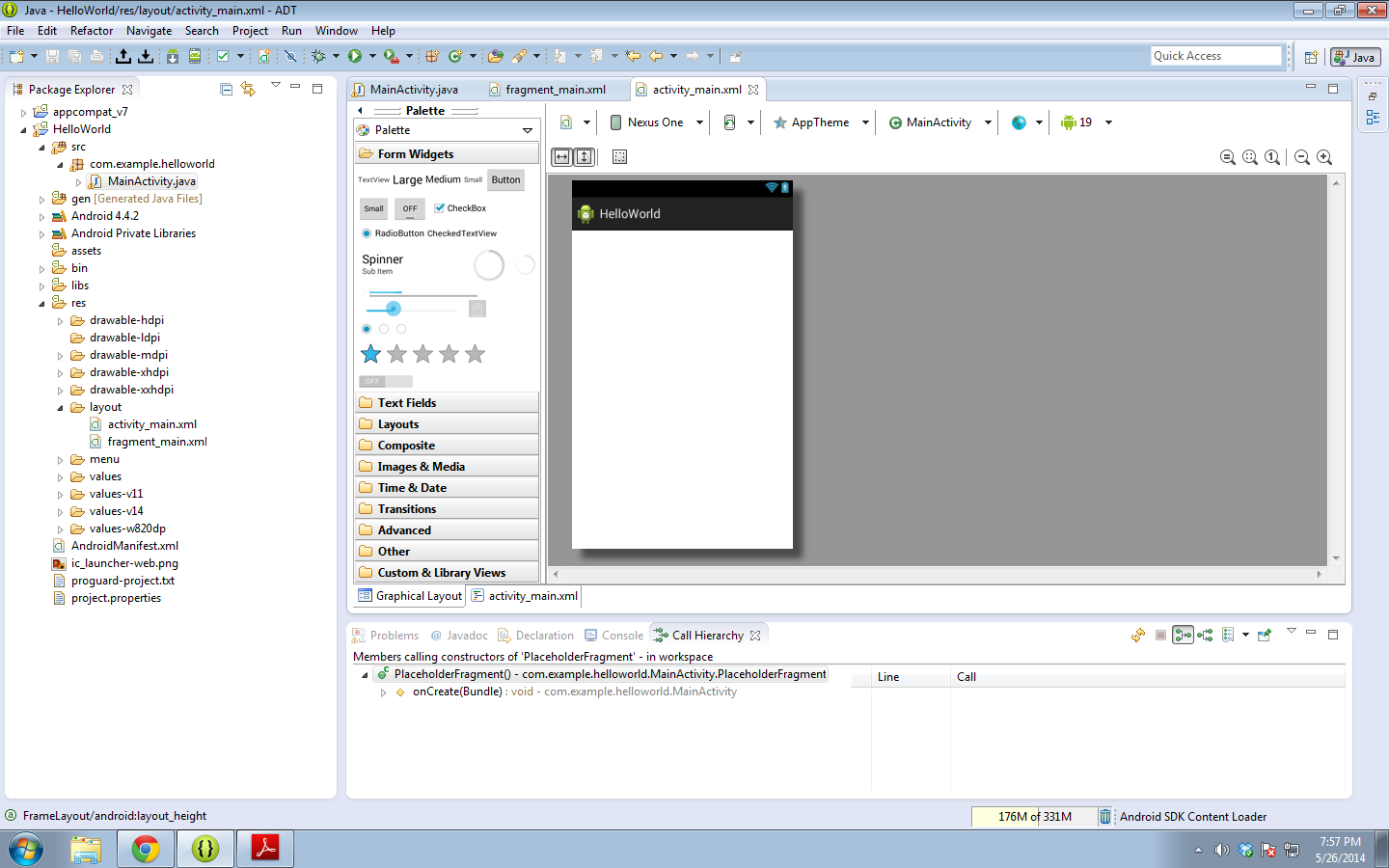
Button helloButton = new Button(this);

helloButton.setText("Hello!");

//add a view to the main layout

mainLayout.addView(helloButton);

Declarative design, on the other hand does not involve any code. You use a tool that will describe what you want to see, not how you want to do it. Google provided WYSIWYG (What You See Is What You Get) tools to build powerful XML-based UI, where in you can visually place UI elements and manipulate them.

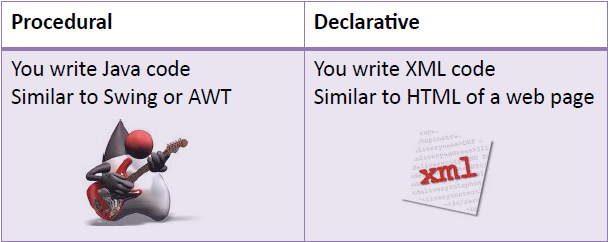


Android provides user interface components to utilize and customize to the needs of specific application. The Android user interface (UI) consists of screen views, screen touch events, and key presses. Each UI object has three definable attributes that customize the look and feel of the UI: the dimension of the object, text in the object, and the color of the object.

Android also relies on an extensible XML framework. To create liquid layouts, android sets resource qualifiers. These can adapt to environmental changes.

## Declarative Vs. Procedural

Which is better? Procedural or Declarative style of creating and manipulating of UI?

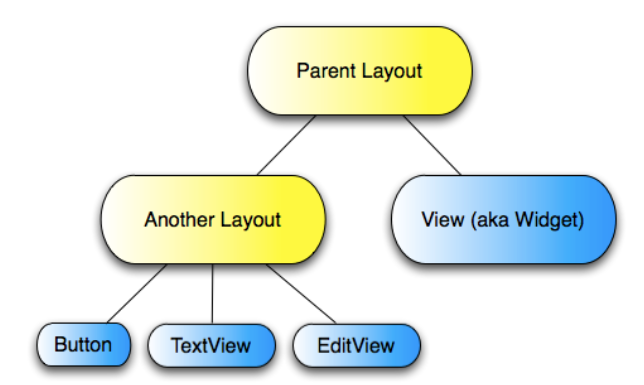


Mix and match styles to suit your development needs.

* Start XML and declare most of UI.
* Switch to Java and implement the UI logic. Example during runtime you want to change the texts of Buttons, TextView etc. or changing their background colors.

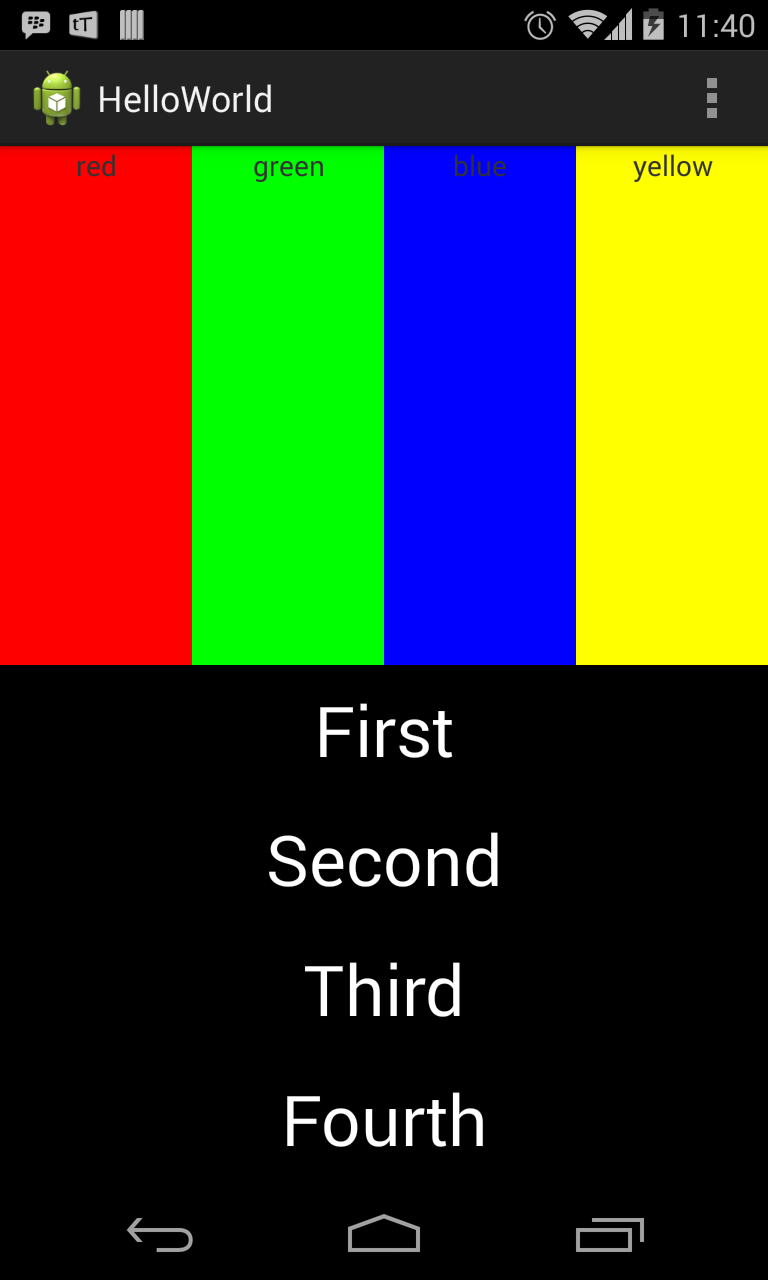
## Android Layouts

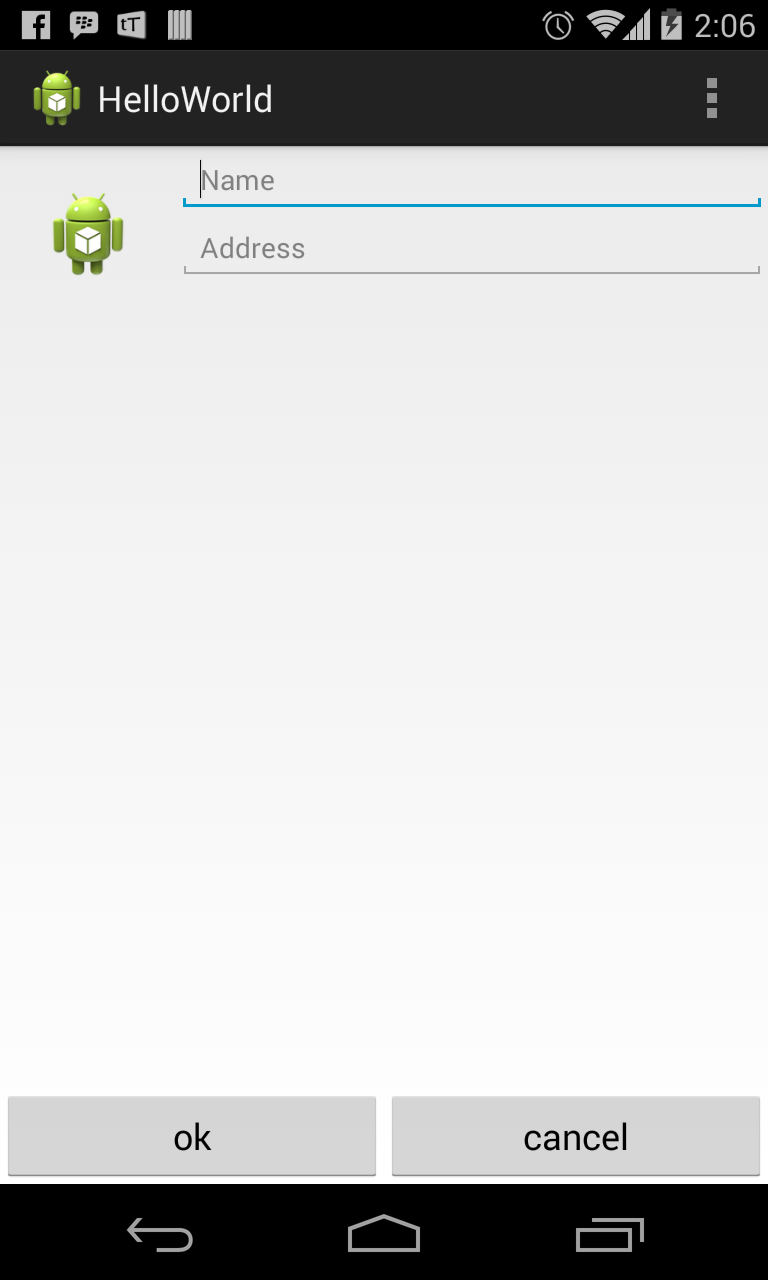
Android Layouts can contain widgets and other layouts forming a composite pattern. Layouts are called View Groups that extends Views that can contain multiple child Views.



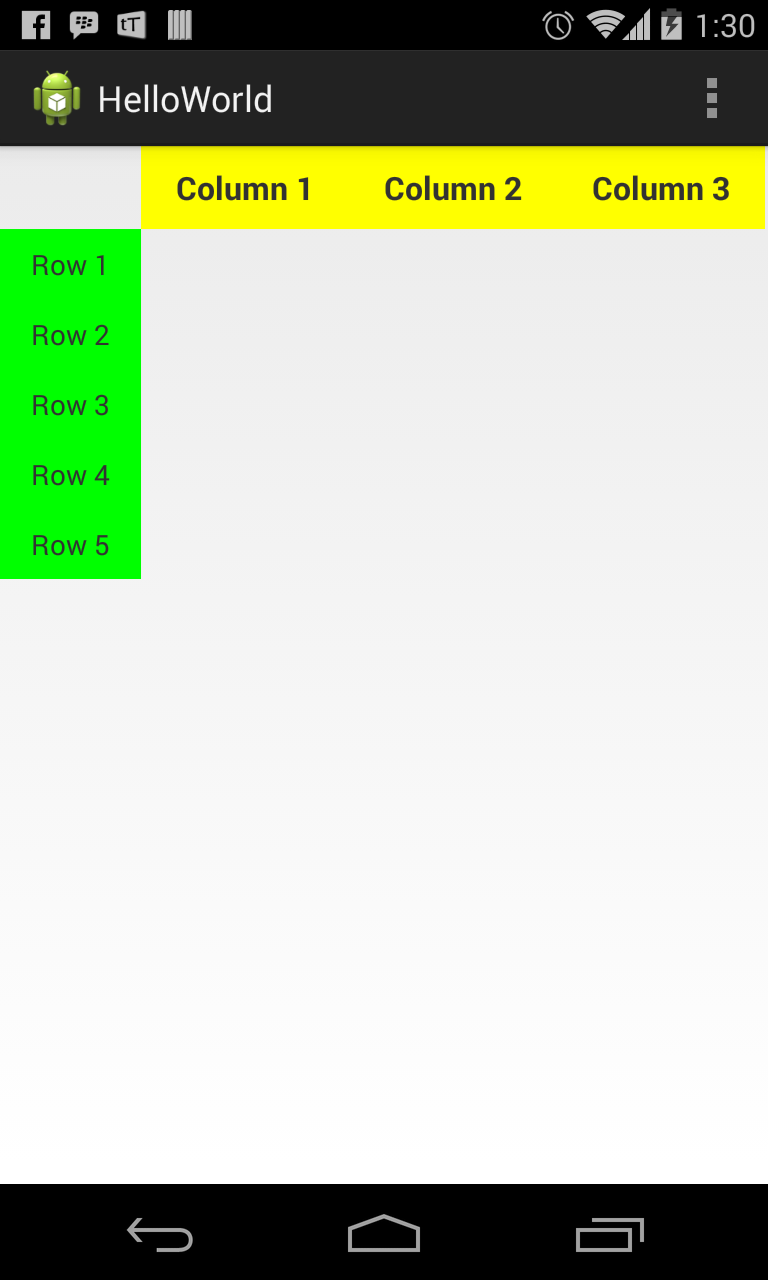
## Commonly Used Layouts

Android provided tools necessary to create types of layouts. The following are the common types of layouts that are available in the Android Software Development Kit (SDK):

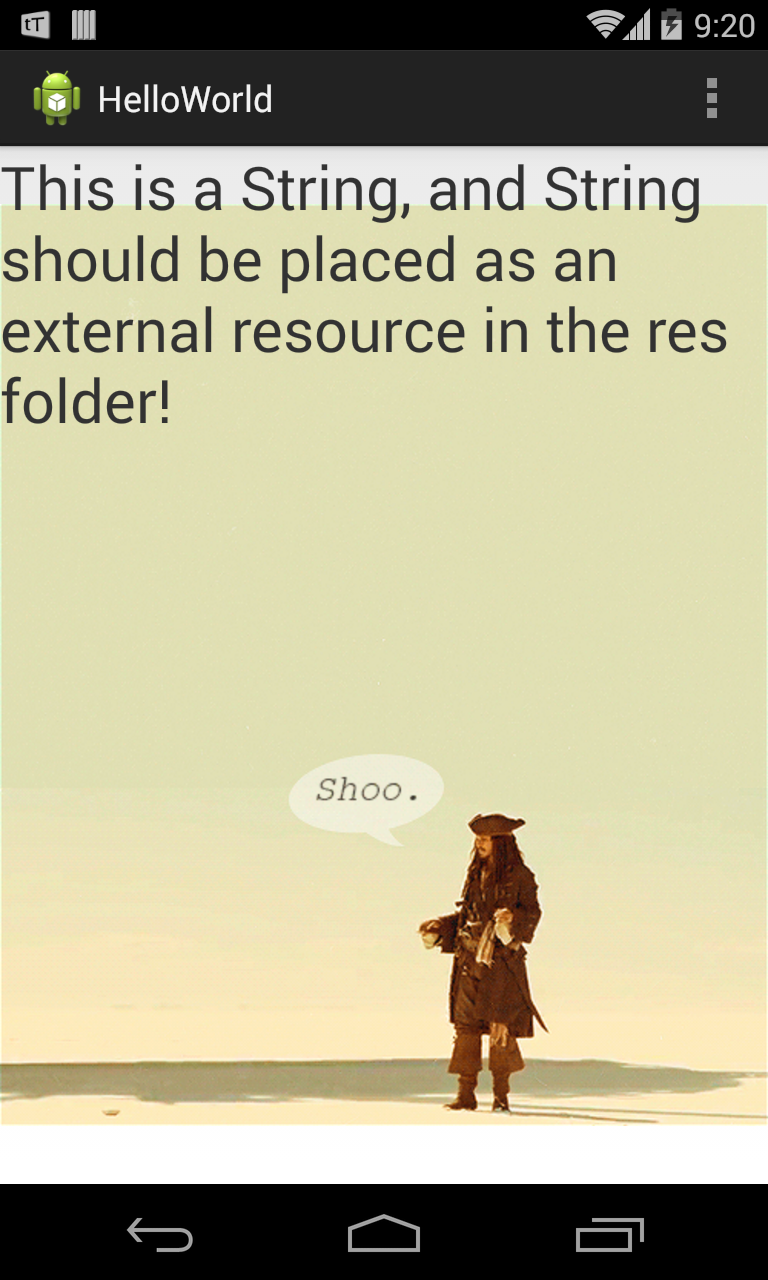
* **LinearLayout** - This layout lays it’s children next to each other, either horizontally and vertically. A Linear Layout aligns each child View in either a vertical or a horizontal line. A vertical layout has a column of Views, whereas a horizontal layout has a row of Views. The Linear Layout supports a weight attribute for each child View that can control the relative size of each child View within the available space.
* **RelativeLayout** - A Layout where the positions of the children can be described in relation to each other or to the parent. Much efficient that LinearLayout - <http://android-developers.blogspot.com/2009/02/android-layout-tricks-1.html>



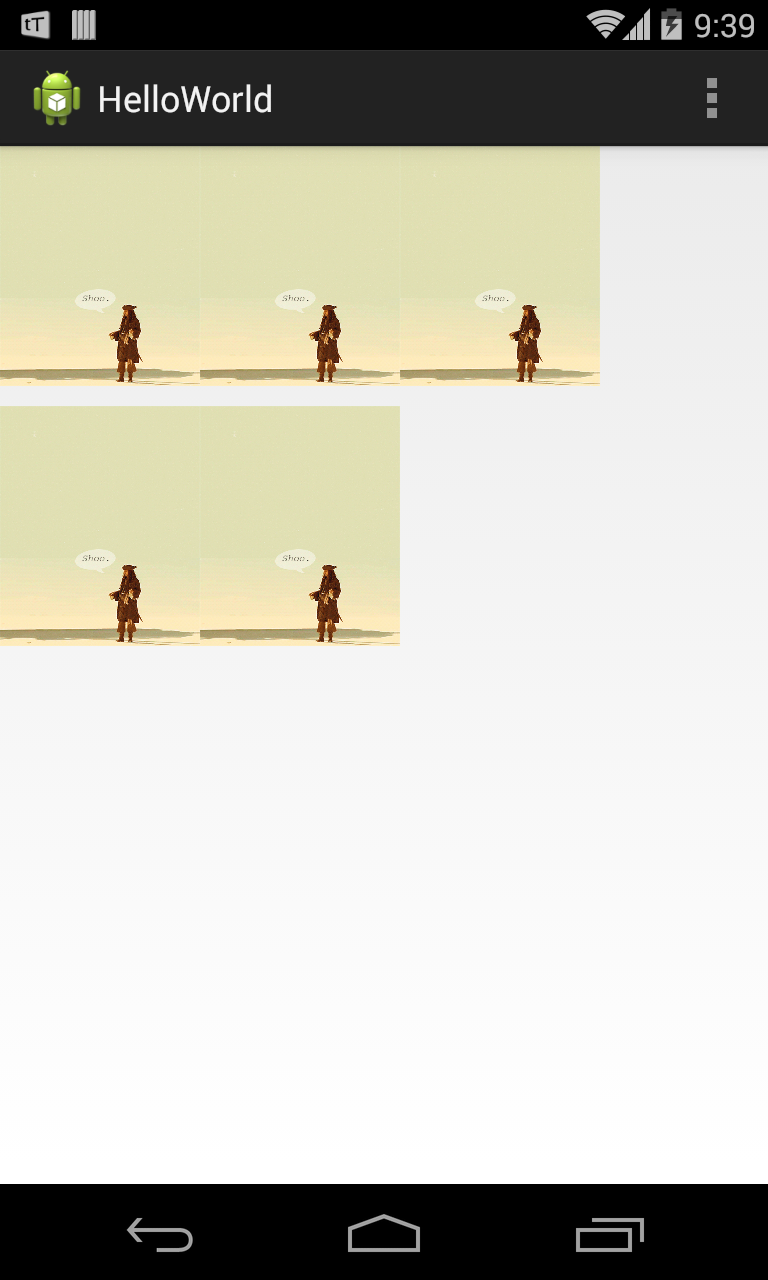
* **TableLayout** - This layout arrange it’s children into table rows and columns. It is similar to a HTML table.



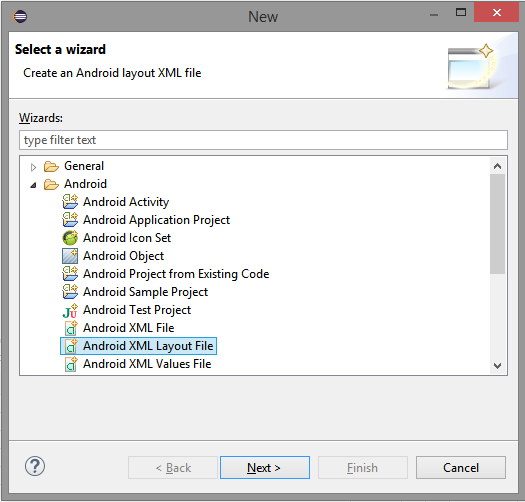
* **FrameLayout** - This layout is commonly used as a way to layout views in an absolute position. The simplest of the Layout Managers, the Frame Layout pins each child view within its frame. The default position is the top-left corner, though you can use the gravity attribute to alter its location. Adding multiple children stacks each new child on top of the one before, with each new View potentially obscuring the previous ones.



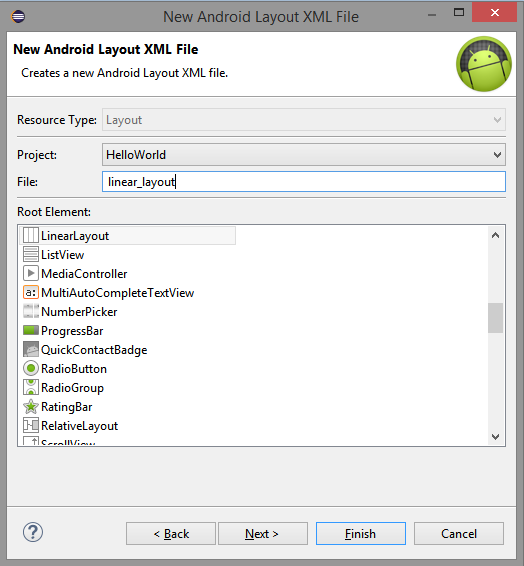
* **GridLayout** - Uses a rectangular grid of infinitely thin lines to lay out Views in a series of rows and columns. It i incredibly flexible and can be used to greatly simplify layouts and reduce or eliminate the complex nesting. It’s good practice to use the Layout Editor to construct your Grid Layouts rather than relying on tweaking the XML manually.



## How to Create Layouts



1. **File** -> **New** -> **Other** -> Choose Android XML Layout File, then click **Next**.



1. Choose the **Root Element** of the layout and provide the name of the layout file and click **Finish**.

## Assigning User Interfaces to Activities

A new Activity starts with a temptingly empty screen onto which you place your UI. To do so, call setContentView, passing in the View instance, or layout resource, to display. Because empty screens aren’t particularly inspiring, you will almost always use **setContentView** to assign an Activity’s UI when overriding its **onCreate** handler.

The **setContentView** method accepts either a layout’s resource ID or a single View instance. This lets you define your UI either in code or using the preferred technique of external layout resources.

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

**setContentView(R.layout.main);**

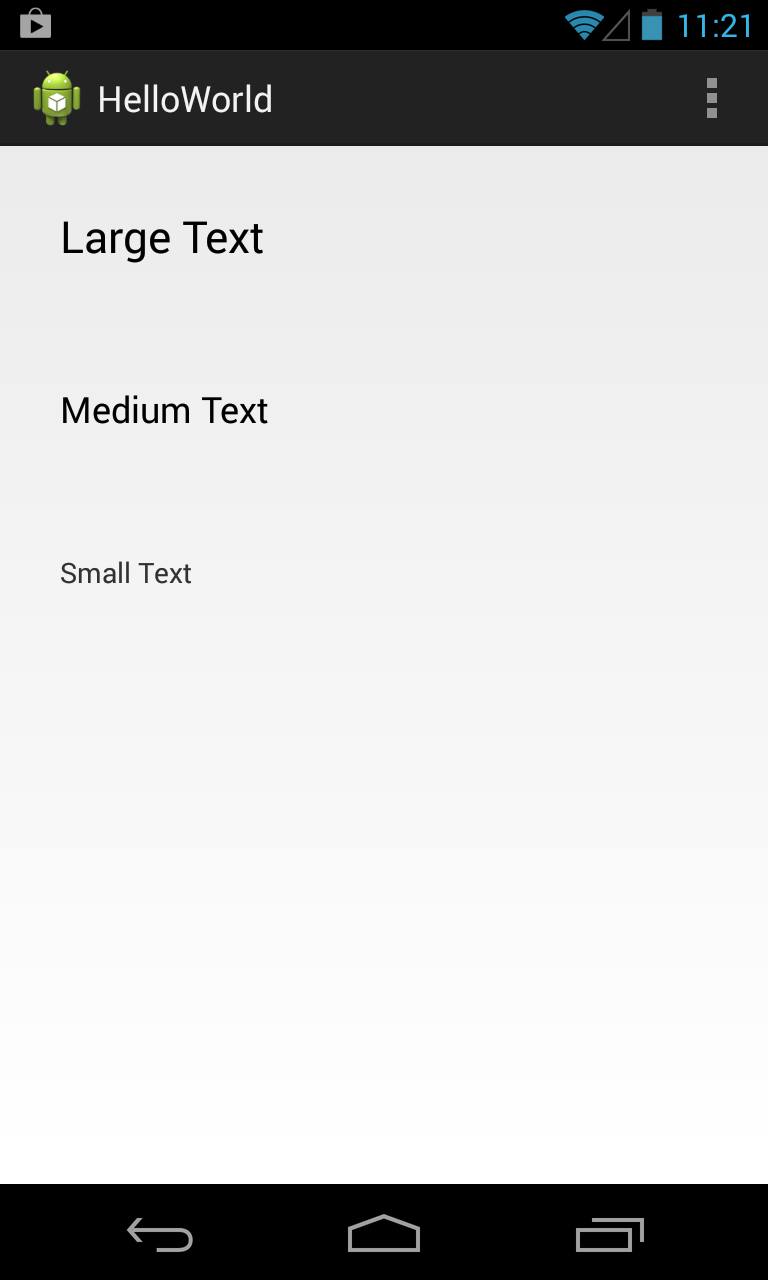
}

## Views

Views in Android are the basic building blocks for user interface components. User interface component such as Layout, TextView, and so on are considered as a view.

**Basic Widgets or Views in Android**

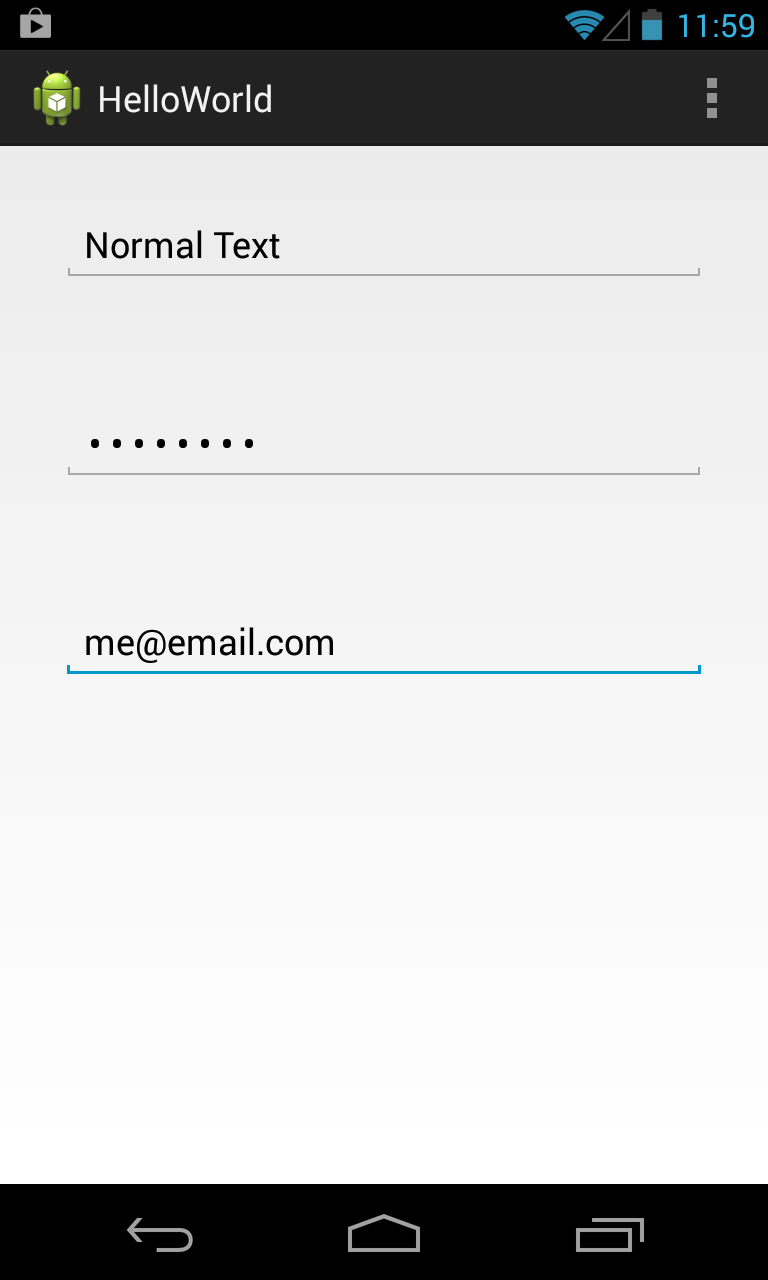
* **TextView** - Displays text to the user and optionally allows them to edit it. A TextView is a complete text editor, however the basic class is configured to not allow editing; see [EditText](http://developer.android.com/reference/android/widget/EditText.html) for a subclass that configures the text view for editing.



* **Button** - Represents a push-button widget. Push-buttons can be pressed, or clicked, by the user to perform an action.



* **EditText** - An editable text entry box that accepts multi line entry, word-wrapping, and hint text.

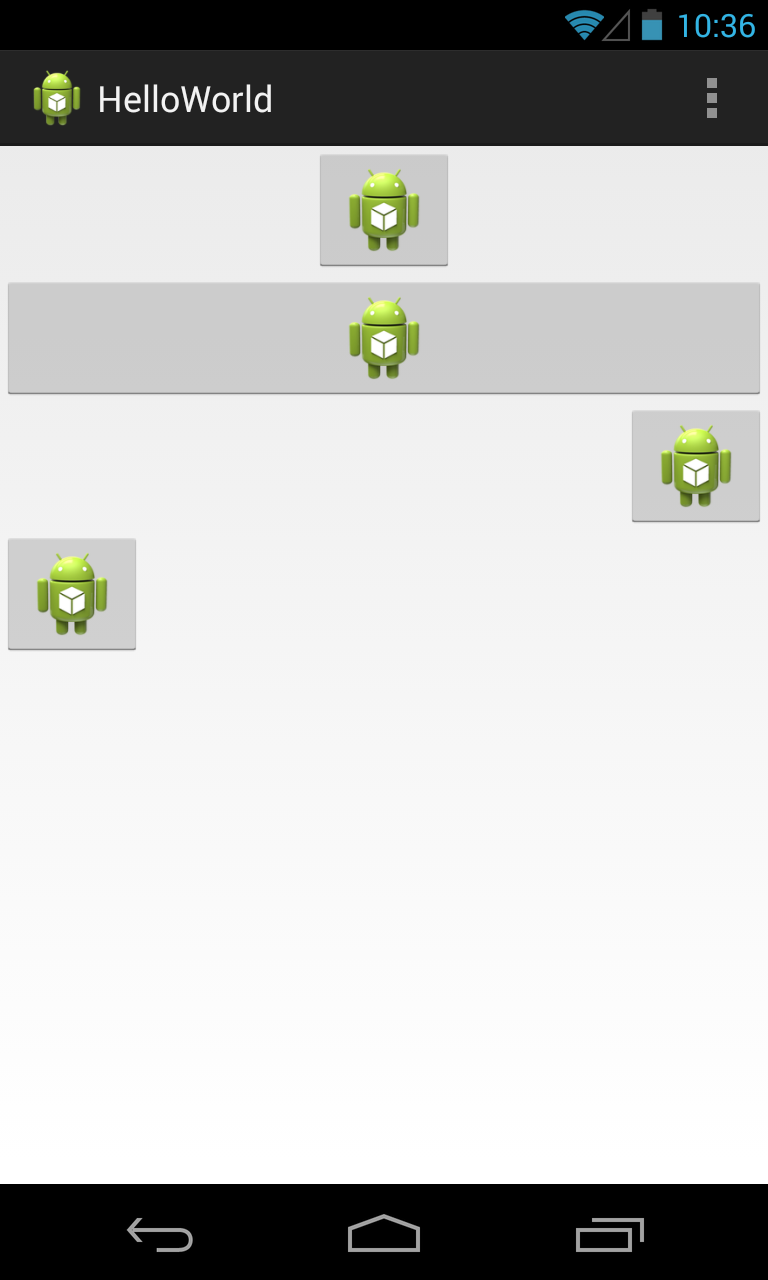


* **ToggleButton** - A two-state button that can be used as an alternative to a check box. It’s particularly appropriate where pressing the button will initiate an action as well as changing a state (such as when turning something on or off).

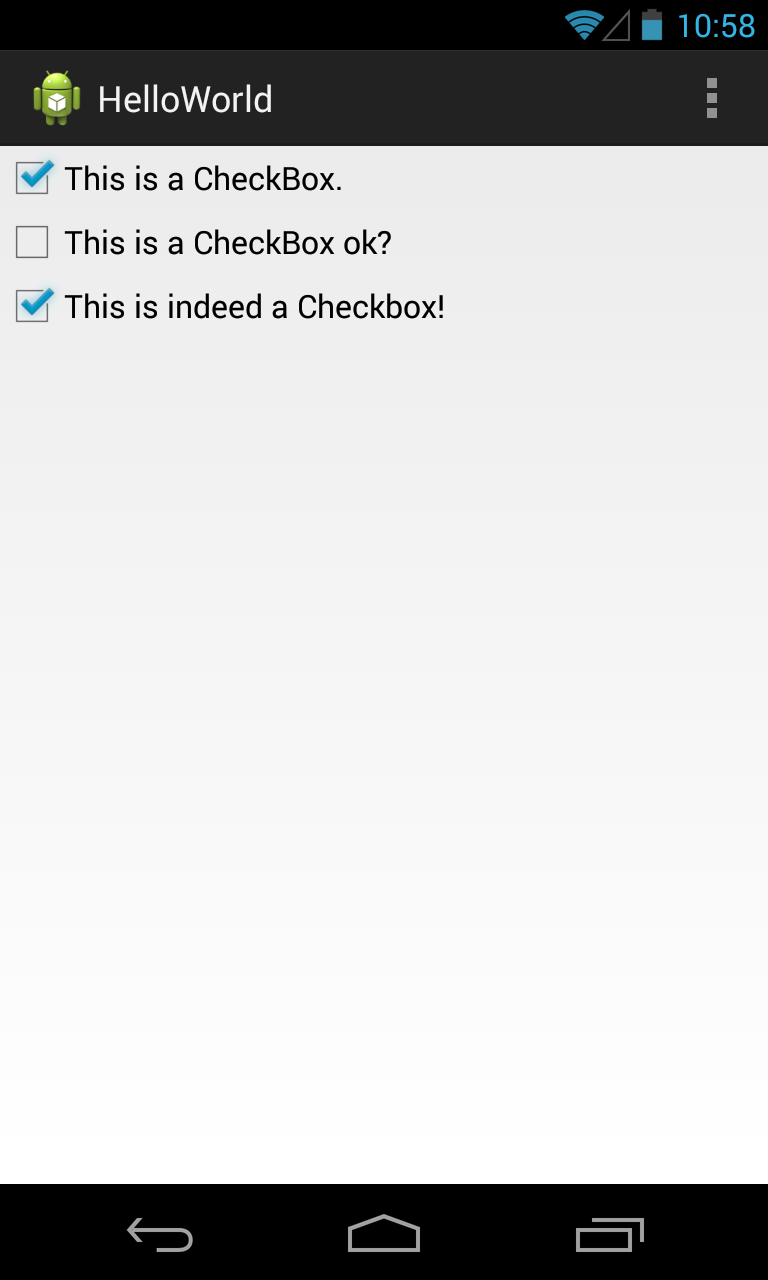


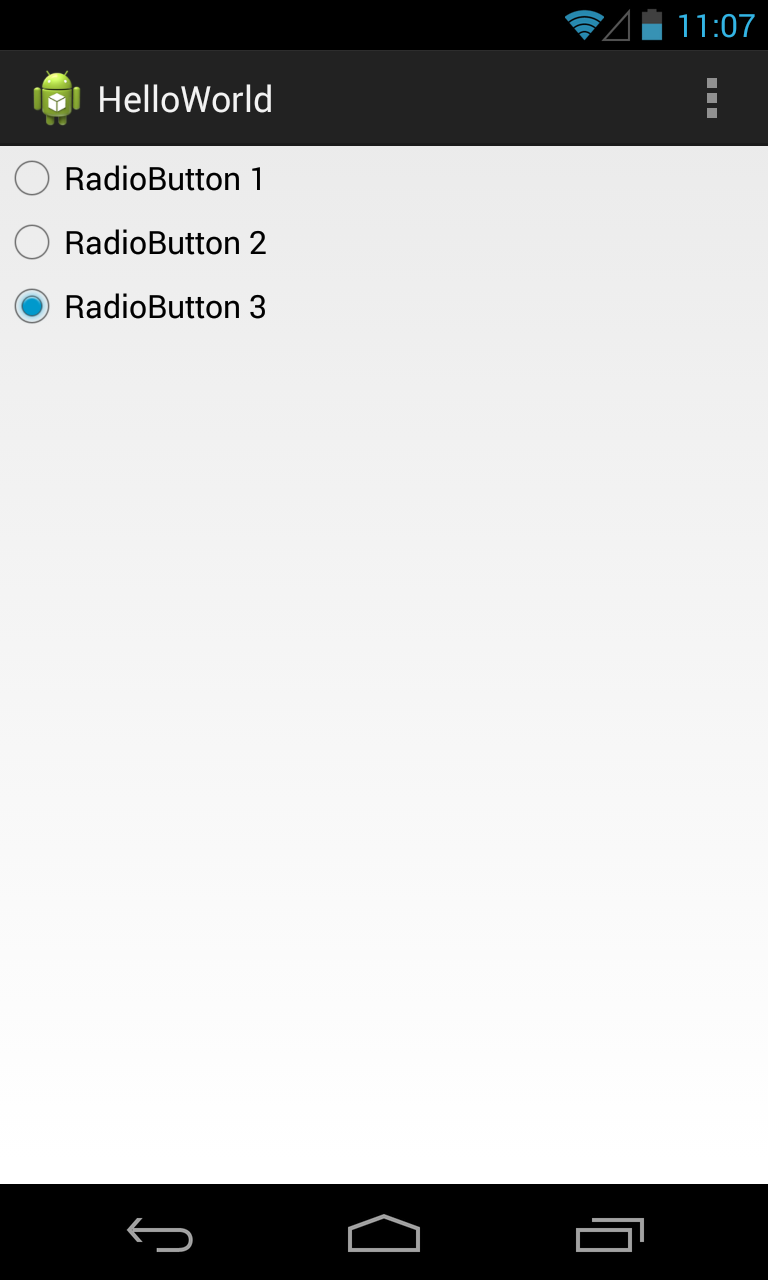
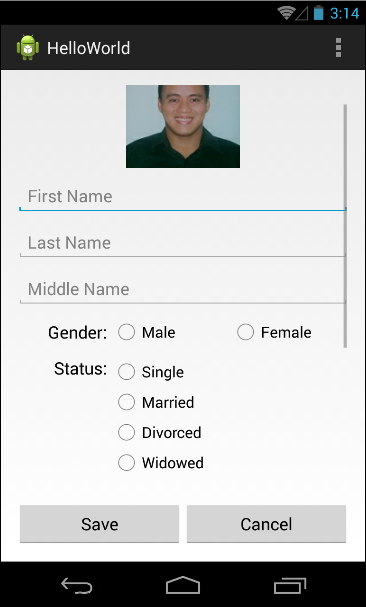


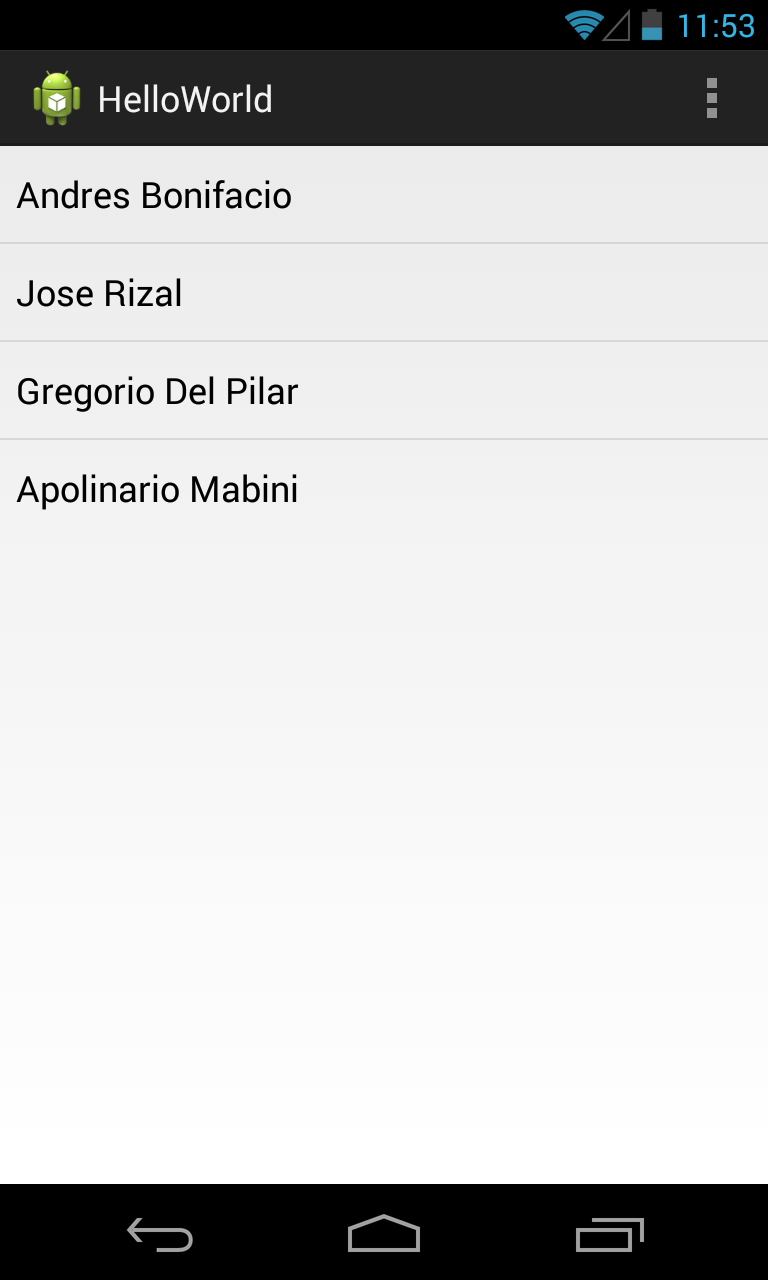
* **ImageButton** - A push button for which you can specify a customized background image (Drawable).



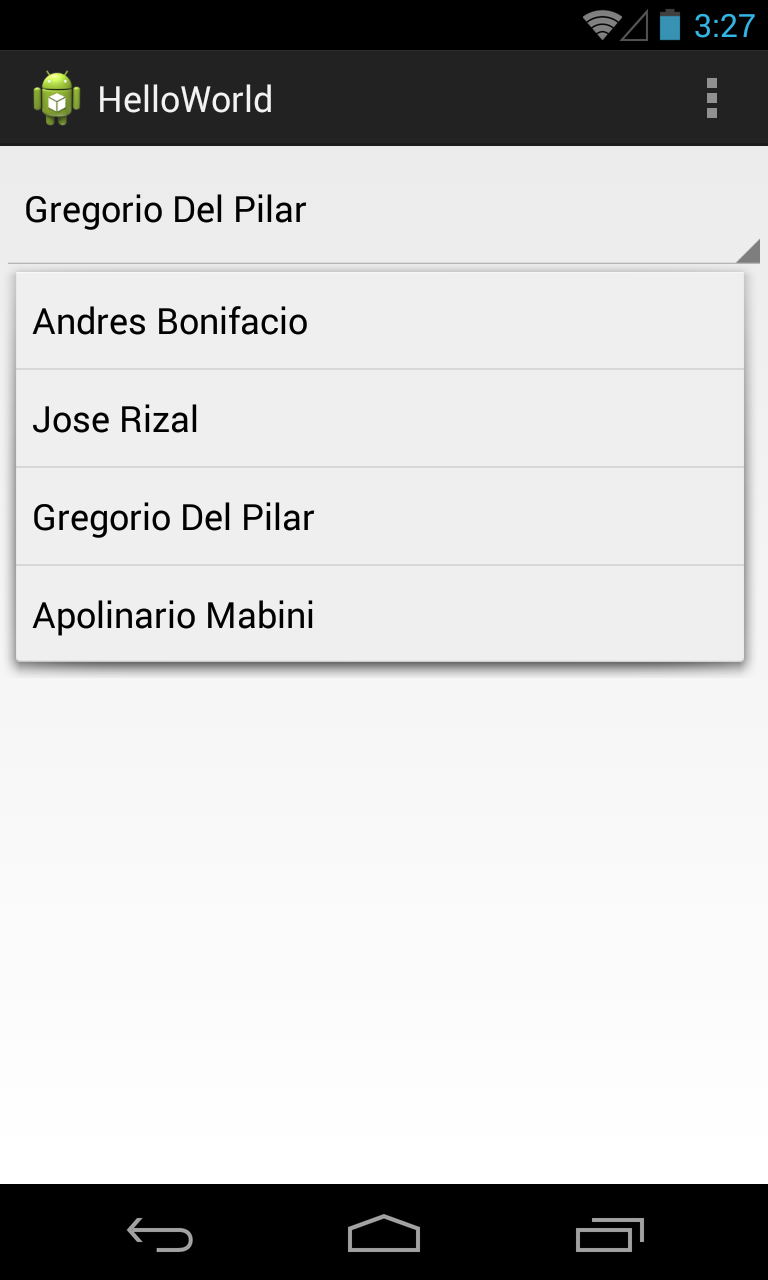
* **CheckBox** - A two-state button represented by a checked or unchecked box.



* **RadioButton** - A two-state grouped button. A group of these presents the user with a number of possible options, of which only one can be enabled at a time.
* **ScrollView** - Layout container for a view hierarchy that can be scrolled by the user, allowing it to be larger than the physical display. A ScrollView is a FrameLayout, meaning you should place one child in it containing the entire contents to scroll; this child may itself be a layout manager with a complex hierarchy of objects. 
* **ViewFilpper** - A ViewGroup that lets you define a collection of Views as a horizontal row in which only one View is visible at a time, and in which transitions between visible views can be animated.
* **ListView** - A ViewGroup that creates and manages a vertical list of Views, displaying them as rows within the list. The simplest ListView displays the *toString* value of each object in an array, using a TextView for each item.



* **Spinner** - A composite control that displays a TextView and an associated List View that lets you select an item from a list to display in the textbox. It’s made from a Text View displaying the current selection, combined with a button that displays a selection dialog when pressed.



## Procedural UI Manipulation

Very often in Android Development controlling or manipulating UI elements dynamically is required when you are dealing with dynamic and runtime changes for your Application’s Views, creating UI views and adding them to ViewGroups, composite controls and getting data from the UI itself. The best practice is to declare all Views in XML layout, make sure IDs are properly and correctly added, obtain the object instance of those views by finding by ID and finally casting the View objects to their designated proper classes. Once the UI objects are found you can do whatever you want to those View objects.

//data set

private String[] mgaBayani = new String []{

"Andres Bonifacio",

"Jose Rizal",

"Gregorio Del Pilar",

"Apolinario Mabini"

};

/\*ListView\*/

//find the ListView from the layout\_with\_views

mListView = (ListView) findViewById(R.id.list\_view);

//create a basic adapter to connect data set to ListView

ArrayAdapter<String> adapter = new ArrayAdapter<String>(

this, android.R.layout.simple\_list\_item\_1, mgaBayani );

//set the adapter to the ListView

mListView.setAdapter(adapter);

/\*Spinner\*/

//find the Spinner from the layout\_with\_views

mSpinner = (Spinner)findViewById(R.id.spinner);

//create a basic adapter to connect data set to Spinner

ArrayAdapter<String> adapter = new ArrayAdapter<String>(

this, android.R.layout.simple\_list\_item\_1, mgaBayani );

//set the adapter to the Spinner

mSpinner.setAdapter(adapter);

/\*ViewFlipper\*/

//find the ListView from the layout\_with\_views

mViewFlipper = (ViewFlipper)findViewById(R.id.view\_flipper);

//add each item in the data set to the ViewFlipper

for(String bayani:mgaBayani){

TextView textView = new TextView(this);

textView.setText(bayani);

textView.setGravity(Gravity.CENTER);

//add the TextView

mViewFlipper.addView(textView);

}

//set the flip interval in milliseconds

mViewFlipper.setFlipInterval(1500); //1.5 seconds

//start the flipping through views

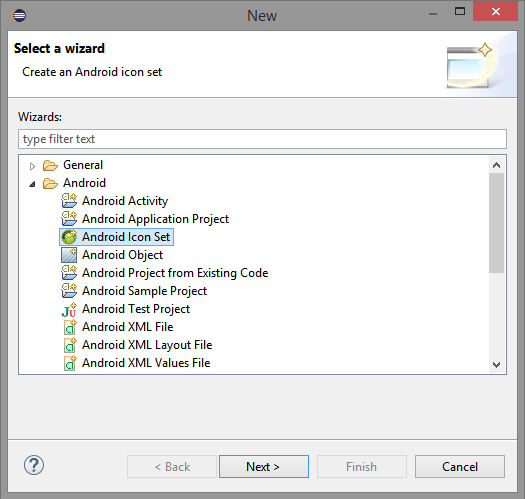
mViewFlipper.startFlipping();

## Launcher Icon

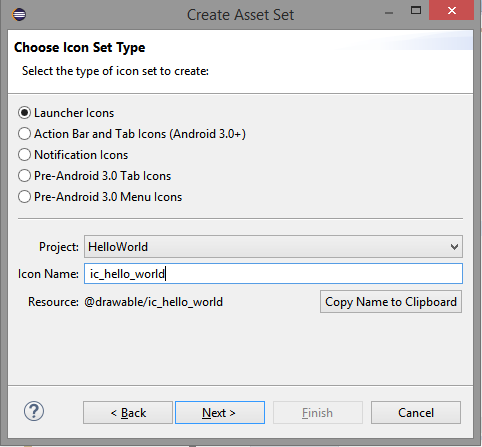
The launcher icon is the visual representation of your app on the Home or All Apps screen. Since the user can change the Home screen's wallpaper, make sure that your launcher icon is clearly visible on any type of background.

### Creating Launcher Icon

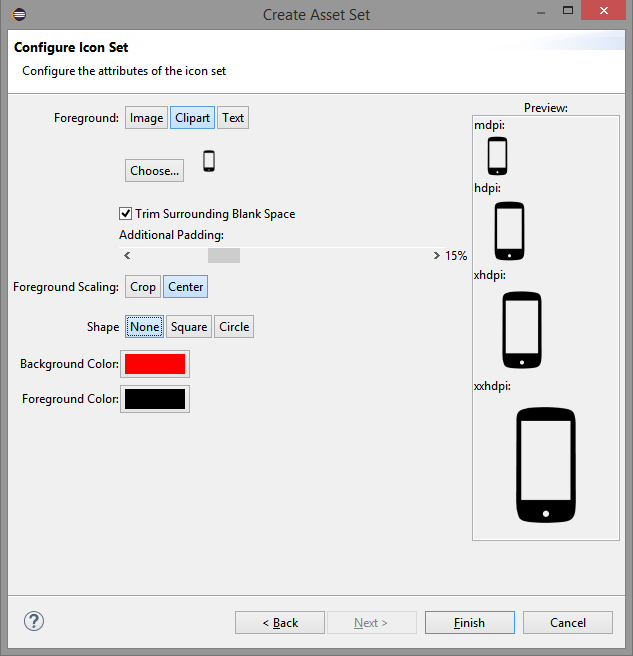
1. In the package explorer of your Eclipse click your project.
2. Right Click the project, choose **New** -> **Other**.
3. Choose **Android Icon Set**.



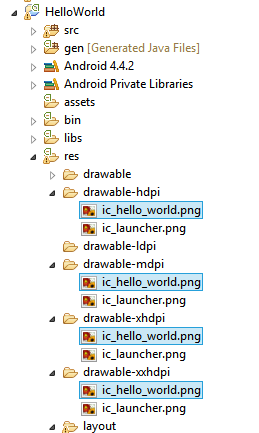
1. Click **Next**.



1. Choose **Launcher Icons**, provide the **Icon Name** and click **Next**.
2. Either use your **Image**, **Clip-art** or **Text** to generate launcher icon.



1. Click **Finish**. Four copies of icon is generated which will be placed in the res (resources) folder under drawables with different resource qualifiers which is scree density. Each copy are in a unique folders **drawable-hdpi**, **drawable-mdpi**, **drawable-xhdpi** and **drawable-xxhdpi**. The android system is using these folders to get proper resources depending on the current configuration, situation or context of the application. For example the launcher icons are placed in drawable folders with screen density qualifier, if the application is installed in a device with resolution device the system will use the **drawable-mdpi, dpi** is a comparative measure of the pixel density of a screen in dots per inch, therefore **mdpi** is for medium density devices, **hdpi** is high, **xhdpi** is extra high and **xxhdpi** for extremely high density devices. Note this is not an exact measure of screen density for a particular devices, just general groupings of screen densities.



1. Apply the newly generated icon set to the application by going to the manifest (**AndroidManifest.xml**), look for the **application** tag and change the **android:icon** value to **@drawable/ic\_icon\_name.png**. What happens here the system will look for a suitable icon file in drawable folders that fits the qualifier.

