eBook Version 1.3

Android Material Design

By AbhiAndroid

Awesome

Thank you for downloading this guide…

Hi,

My name is Abhishek. I'm 26 year old guy from India.

I'm on mission and my mission is to help people learn and grow as an Android Developer.

So, that's why I created AbhiAndroid to reach more and more people like you.

I have created this eBook with love for those developer who want to take a deeper dive in Android Material Design and want to drive their skills to the next level.

With love & respect,

Abhishek Saini

You can connect with me on Facebook

Live Android App Project Source Code

Before you start reading this guide, I would like to share premium live Android App project source code build specially for you. These source code are available at a very affordable price giving chance for developer and student to understand how complex Android App are built.

With each source code you will get step by step documentation and 6 months of FREE email support.

You can grab the source code here: http://abhiandroid.com/sourcecode/

Service

I also offer Android App development service and you can consider hiring me. Some of my services include developing App from scratch, adding new feature in App, fixing bugs, enhancing Android App or customization.

You can know more about my services here: http://abhiandroid.com/service/

To reach us please email: info@abhiandroid.com

Important Message

This copy is purely dedicated to you. You can use it in several ways. Save it in your laptop, mobile, take a printout, and please, no need to say thanks. But you can't sell it or you can't make a change in it because all rights of this copy is with AbhiAndroid.com. If want some changes in it or some addition to it, you can mail me at info@abhiandroid.com. And, if you like this guide, don't forget to share it with your buddies. I'm sure they will appreciate it.

Tables Of Content

Introduction to Material Design

Material Design In Android:

In Android, Material Design is comprehensive guide for motion, visual and interaction design across platforms and devices. Its introduced in Android Lollipop version 5.0(API level 21) by Google at the 2014 Google I/O Conference on 25th June of 2014. With Introduction of Material design a lot of new things were introduced like new widgets, animations, material themes and others. Google has proposed some rules and regulations while adding this Material Design to our application's to improvise its standards.

Special Note: In Material Design, the visual specifics are amusing and the material objects have x, y, and z dimensions which allows us to create/build an incredible 3D world. Material Design is just not about how to use the best images, dazzling colors, nor the elevation of the object; it's about how we can create the amazing experience to users with the positive brand reality. At the end, we can say that Material Design is clear, straightforward, and brilliant.

Elements Of Material Design In Android:

Here are some common elements of Material Design which makes it better.

1. The material theme: A theme is a style applied to activity or application. In Material Design we have new themes like Material Light Theme, Material Dark Theme and Material Light with Dark Action Bar Theme to be precise.

a. @android:style/Theme.Material: Dark version.

b. @android:style/Theme.Material.Light: Light version.

c. @android:style/Theme.Material.Light.DarkActionBar.

2. Widgets for cards and lists: Material Design introduced new widgets like RecyclerView, CardView and other. RecyclerView is a replacement of ListView and GridView. CardView is used to display the data in the form of Cards with use of elevation, corners and others properties.

3. Custom shadows and view clipping: Widgets had an X and Y positions on the screens but now they have Z position. The new Z position defines the virtual elevation of the widget from the screen surface in order to give it the appearance as if it is hovering and thus projecting a shadow of the view to the surface below. Clipping is used to give the views the appropriate shape.

4. SVG drawables: SVG(Scalar vector graphics) is supported now in Android Lollipop. Now we can create resources which define the graphics using SVG.

5. Custom animations: Now we can also animate our items(RecyclerView items).

Goals Of Material Design In Android:

Here are the common goals of Material Design.

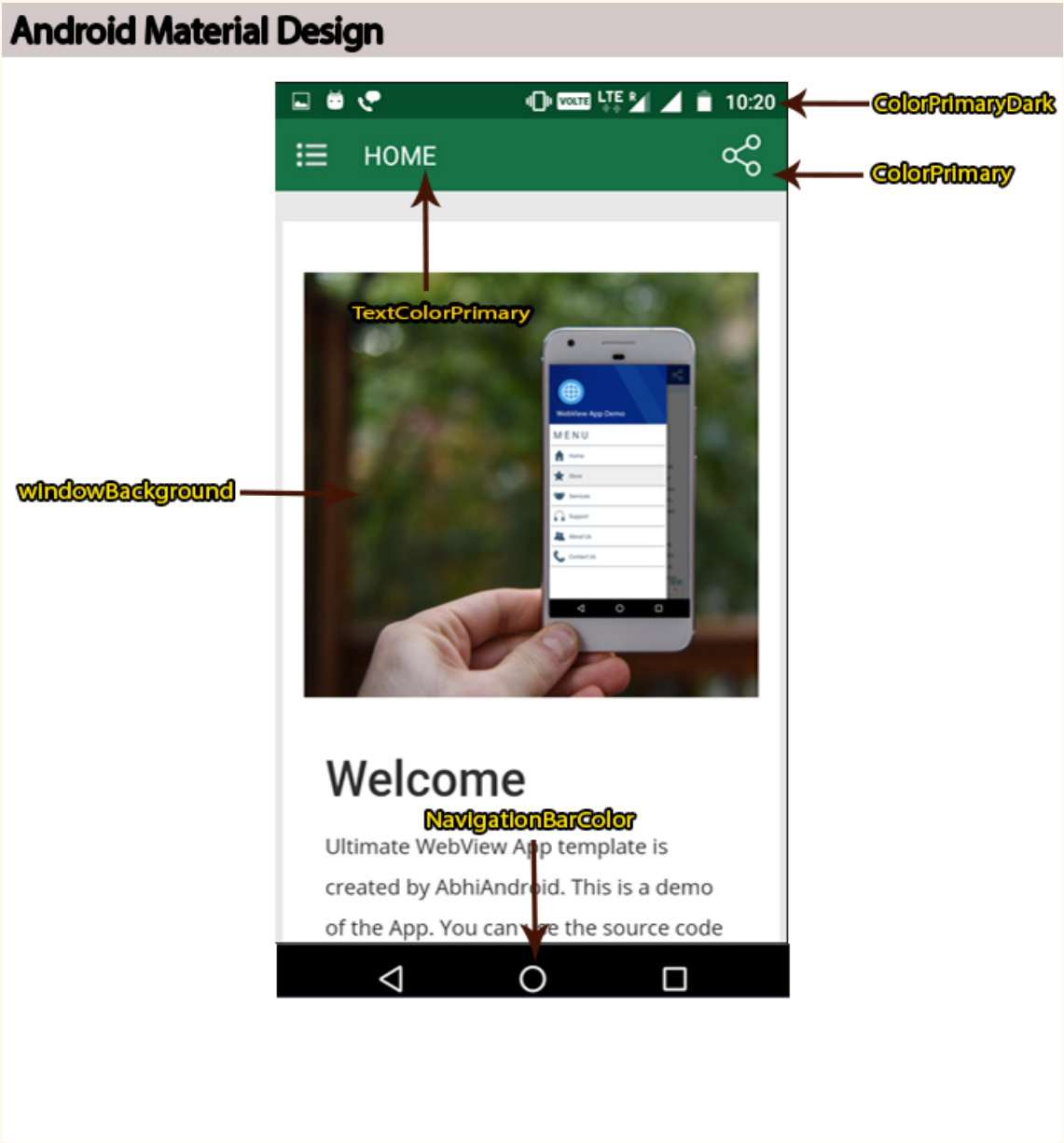
1. Design an Application UI like a magical paper means something like real and appropriate.

2. Animations have been pulled to make the experience more lively by safeguarding the maximum amount of content is always visible to the users.

3. In Material Design Google also determined to robotize experience for users.

Material Design Color Customization In Android:

In Android, Material Design provides us a set of properties to customize the Color theme but we use five primary attributes to customize overall theme.



1. colorPrimaryDark – This is darkest primary color of the app mainly applies to notification bar background.

2. colorPrimary – This is the primary color of the app. This color will be applied as toolbar background.

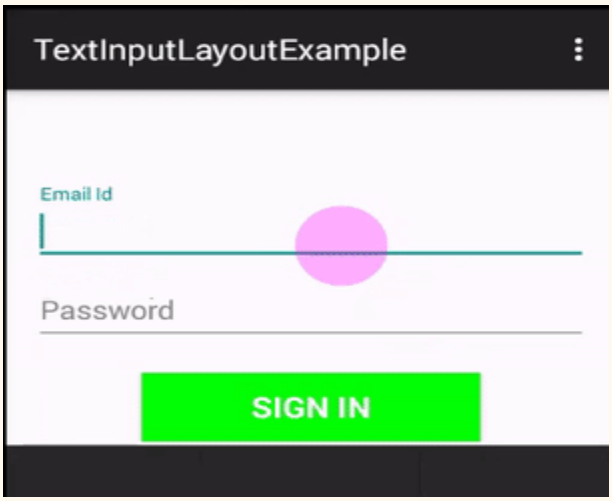
3. textColorPrimary – This is the primary color of text. This applies to toolbar title.

4. windowBackground – This is the default background color of the app.

5. navigationBarColor – This is the color defines the background color of footer navigation bar.

TextInputLayout / Floating Labels In EditText

TextInputLayout is a new element introduced in Design Support library to display the floating label in EditText. To display floating label in EditText, TextInputLayout needs to wrapped the EditText. We can also display the error message to EditText by using setError() and setErrorEnabled() methods. It takes the value of hint assigned to EditText and displays it as floating label. Android Design Support Library introduced some important new widgets that helps us to create consistent UI.



Floating Labels: Floating labels first introduced in Android design support library to display floating label over EditText. Firstly it acts as hint in the EditText when the field is empty. After that when a user start inputting the text it starts animating by moving to floating label position.

Special Note: In Android, one of the most basic UI element or widgets is an EditText. It is generally used to take the any kind of input from the user but what it lacks was a label attached to it. Therefore in most of the implementations hint was used as a label for the EditText. From the time material design was released, a new concept of floating labels was introduced. In this

concept initially showed a label as a hint and when a user enters a value in the EditText that hint moves on to the top of the EditText as a floating label.

Basic TextInputLayout XML Code:

```
<android.support.design.widget.TextInputLayout
android:id="@+id/simpleTextInputLayout"
android:layout_width="wrap_content"
android:layout_height="wrap_content">

<EditText
android:id="@+id/simpleEditText"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:hint="Enter Your Name" />
</android.support.design.widget.TextInputLayout>
```

Important Methods Of TextInputLayout:

Let's we discuss some important methods of TextInputLayout that may be called in order to manage the TextInputLayout.

1. setError(CharSequenceerror): This method is used to set an error message that will be displayed below our EditText. In this method we set CharSequence value for error message.

Below we set the error message that will be displayed below our EditText.

```
TextInputLayout simpleTextInputLayout = (TextInputLayout)
findViewById(R.id.simpleTextInputLayout); // get the reference of TextInputLayout
simpleTextInputLayout.setError("Please Check Field"); // set the error message that will be
displayed below our EditText
```

2. getError(): This method is used for getting the error message that was set to be displayed using setError(CharSequence). It returns null if no error was set or error displaying is not enabled. This method returns CharSequence type value.

Below we firstly set the error and then get the error message that was set to be displayed using setError(CharSequence) method.

```
TextInputLayout simpleTextInputLayout = (TextInputLayout)
findViewById(R.id.simpleTextInputLayout); // get the reference of TextInputLayout
simpleTextInputLayout.setError("Please Check Field"); // set the error message that will be
displayed below our EditText
CharSequence errorMessage=simpleTextInputLayout.getError(); // get the error message that
was set to be displayed using setError(CharSequence) method.
```

3. setErrorEnabled(boolean enabled): This method is used to set whether the error functionality is enabled or not in this layout. We set true for enabled and false for disabled error functionality.

Below we set the true value that enabled the error functionality.

```
TextInputLayout simpleTextInputLayout = (TextInputLayout)
findViewById(R.id.simpleTextInputLayout); // get the reference of TextInputLayout
simpleTextInputLayout.setErrorEnabled(true); // set the true value that enabled the error
functionality
```

4. isErrorEnabled(): This method is used to check whether the error functionality is enabled or not in this layout. This method returns Boolean type value which we set using setErrorEnabled (boolean enabled) method.

Below we firstly enabled the error functionality and then check whether the error functionality is enabled or not.

```
TextInputLayout simpleTextInputLayout = (TextInputLayout)
findViewById(R.id.simpleTextInputLayout); // get the reference of TextInputLayout
simpleTextInputLayout.setErrorEnabled(true); // set the true value that enabled the error
functionality
Boolean isEnabled = simpleTextInputLayout.isErrorEnabled(); // check whether the error
functionality is enabled or not
```

5. setHint(CharSequence hint): This method is used to set the hint to be displayed in the floating label, if enabled. In this method we set CharSequence value for displaying hint.

Below we set the hint to be displayed in the floating label.

```
TextInputLayout simpleTextInputLayout = (TextInputLayout)
findViewById(R.id.simpleTextInputLayout); // get the reference of TextInputLayout
simpleTextInputLayout.setHint("Enter UserName"); // set hint displayed in floating label.
```

6. getHint(): This method is used to get the hint which is displayed in the floating label, if enabled. This method returns CharSequence type value.

Below we set the hint and then get the hint which is displayed in the floating label

```
TextInputLayout simpleTextInputLayout = (TextInputLayout)
findViewById(R.id.simpleTextInputLayout); // get the reference of TextInputLayout
simpleTextInputLayout.setHint("Enter UserName"); // set the hint to be displayed in the
floating label.
CharSequence hintValue = simpleTextInputLayout.getHint(); // get the hint which is displayed
in the floating label
```

7. setCounterMaxLength(int maxLength): This method is used to set the max length value to display at the character counter. In this method we pass int type value for setting max length value.

Below we set the max length value to display at the character counter.

```
TextInputLayout simpleTextInputLayout = (TextInputLayout)
findViewById(R.id.simpleTextInputLayout); // get the reference of TextInputLayout
simpleTextInputLayout.setCounterMaxLength(10); // set 10 max length value to display at the
character counter
```

8. getCounterMaxLength(): This method is used for getting the max length shown at the character counter. This method returns int type value which we set through setCounterMaxLength(int maxLength) method.

Below we firstly set the max length value and then get the max length value that shown at the character counter.

```
TextInputLayout simpleTextInputLayout = (TextInputLayout)
findViewById(R.id.simpleTextInputLayout); // get the reference of TextInputLayout
simpleTextInputLayout.setCounterMaxLength(10); // set 10 max length value to display at the
character counter
int maxLengthValue= simpleTextInputLayout.getCounterMaxLength(); // get the max length value
that shown at the character counter.
```

9. setCounterEnabled(boolean enabled): This method is used to set whether the character counter functionality is enabled or not in this layout. . We set true for enabled and false for disabled the counter functionality.

Below we set the true value that enabled the counter functionality in this layout.

```
TextInputLayout simpleTextInputLayout = (TextInputLayout)
findViewById(R.id.simpleTextInputLayout); // get the reference of TextInputLayout
simpleTextInputLayout.setCounterEnabled(true); // set the true value that enabled the
counter functionality in this layout
```

10. setTypeface(Typeface typeface): This method is used to set the typeface to use for both the expanded and floating hint.

Below we set Sans – Serif typeface to use for the expanded and floating hint.

```
TextInputLayout simpleTextInputLayout = (TextInputLayout)
findViewById(R.id.simpleTextInputLayout); // get the reference of TextInputLayout
simpleTextInputLayout.setTypeface(Typeface.SANS_SERIF); // set Sans - Serif typeface to use
for the expanded and floating hint.
```

11. getTypeface(): This method is used for getting the typeface used for both the expanded and floating hint. This method returns Typeface type value which we set using setTypeface(Typeface typeface) method.

Below we firstly we set Sans – Serif typeface and then get the typeface used for both the expanded and floating hint.

```
TextInputLayout simpleTextInputLayout = (TextInputLayout)
findViewById(R.id.simpleTextInputLayout); // get the reference of TextInputLayout
simpleTextInputLayout.setTypeface(Typeface.SANS_SERIF); // set Sans - Serif typeface to use
for the expanded and floating hint.
Typeface typeface = simpleTextInputLayout.getTypeface(); // get the typeface used for both
the expanded and floating hint.
```

Attributes of TextInputLayouts:

Now let's we discuss some common attributes of a TextInputLayout that helps us to configure it in our layout (xml).

1. support.design:counterMaxLength: This attribute is used to set the max length to display in the character counter. In this attribute we set int type value for setting max length value. We can also do this programmatically using setCounterMaxLength(int maxLength) method.

Below we set the max length value to display at the character counter.

```
<android.support.design.widget.TextInputLayout
android:id="@+id/simpleTextInputLayout"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android.support.design:counterMaxLength="10">

<EditText
android:id="@+id/simpleEditText"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:hint="Enter Your Name" />
</android.support.design.widget.TextInputLayout>

<!--   we set the max length value to display at the character counter -->
```

2. support.design:counterEnabled: This attribute is used to set whether the character counter functionality is enabled or not in this layout. We set true for enabled and false for disabled the counter functionality. We can also do this programmatically using setCounterEnabled(boolean enabled) method.

Below we set the true value that enabled the counter functionality in this layout.

```
<android.support.design.widget.TextInputLayout
android:id="@+id/simpleTextInputLayout"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android.support.design:counterEnabled="true"
android.support.design:counterMaxLength="10">

<EditText
android:id="@+id/simpleEditText"
android:layout_width="match_parent"
android:layout_height="wrap_content"
```

```
android:hint="Enter Your Name" />
</android.support.design.widget.TextInputLayout>
<!-- set the true value that enabled the counter functionality in this layout-->
```

3. support.design:errorEnabled: This attribute is used to set whether the error functionality is enabled or not in this layout. We set true for enabled and false for disabled error functionality. We can also do this programmatically using setErrorEnabled(boolean enabled) method.

Below we set the true value that enabled the error functionality.

```
<android.support.design.widget.TextInputLayout
android:id="@+id/simpleTextInputLayout"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android.support.design:counterEnabled="true"
android.support.design:errorEnabled="true">

<EditText
android:id="@+id/simpleEditText"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:hint="Enter Your Name" />
</android.support.design.widget.TextInputLayout>
<!-- set the true value that enabled the error functionality in this layout-->
```

4. android:hint: This attribute is used to set the hint to be displayed in the floating label. We can also set hint programmatically using setHint(CharSequence hint) method.

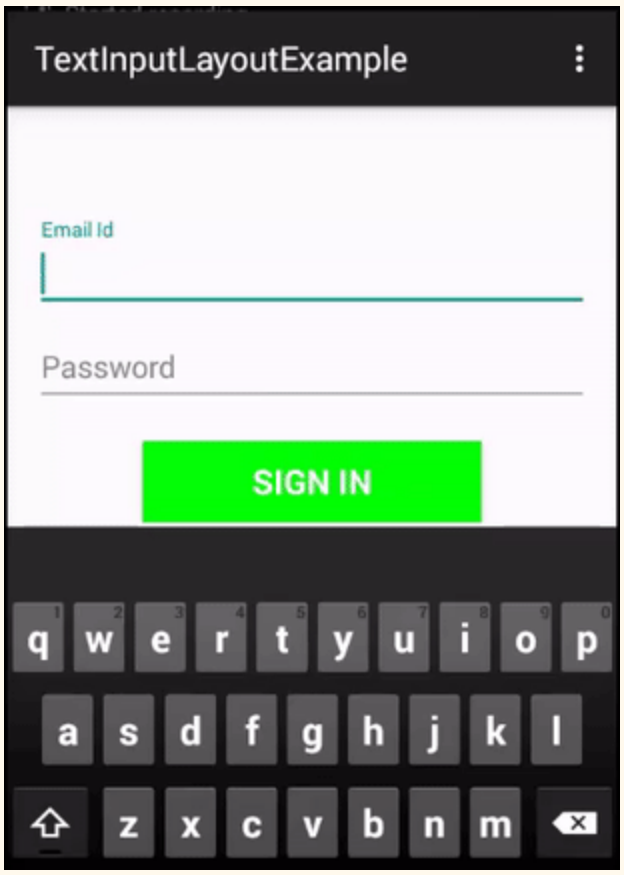
Below we set the hint to be displayed in the floating label.

```
<android.support.design.widget.TextInputLayout
android:id="@+id/simpleTextInputLayout"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:hint="User Name">

<EditText
android:id="@+id/simpleEditText"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
</android.support.design.widget.TextInputLayout>
<!-- set the hint to be displayed in the floating label-->
```

TextInputLayout/ Floating Labels In EditText Example In Android Studio:

Below is the example to show the usage of TextInputLayout where we create a login form with floating labels, input validations and error messages enabled. In this we also display a Sign In Button and perform click event on it so whenever a user click on Button we check the Fields and if a field is empty we display the error message otherwise we display “Thank You” message by using a Toast.



Step 1: Create a new project and name it TextInputLayoutExample

Step 2: Open Gradle Scripts > build.gradle and add Design support library dependency.

```
apply plugin: 'com.android.application'

android {
compileSdkVersion 23
buildToolsVersion "23.0.2"

defaultConfig {
applicationId "example.abhiandroid.textinputlayoutexample"
minSdkVersion 15
targetSdkVersion 23
versionCode 1
versionName "1.0"
}
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
}
}
}

dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
compile 'com.android.support:appcompat-v7:23.1.1'
compile 'com.android.support:design:23.1.0' // design support library

}
```

Step 3: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:android.support.design="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
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=".MainActivity">
<!-- first TextInputLayout -->
<android.support.design.widget.TextInputLayout
android:id="@+id/emailTextInputLayout"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginTop="50dp"
android.support.design:counterMaxLength="3">

<EditText
android:id="@+id/emailEditText"
android:layout_width="match_parent"
android:layout_height="wrap_content"
```

```
android:hint="Email Id" />
</android.support.design.widget.TextInputLayout>
<!-- first TextInputLayout -->

<android.support.design.widget.TextInputLayout
android:id="@+id/passwordTextInputLayout"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:hint="Password">

<EditText
android:id="@+id/passwordEditText"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
</android.support.design.widget.TextInputLayout>
<!-- sign In Button -->
<Button
android:id="@+id/signInButton"
android:layout_width="200dp"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="20dp"
android:background="#0f0"
android:text="Sign In"
android:textColor="#FFF"
android:textSize="20sp"
android:textStyle="bold" />
</LinearLayout>
```

Step 4: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code for initiate the views (TextInputLayout and other views). After that we perform click event on Button so whenever a user click on Button we check the Fields and if a field is empty we display the error message otherwise we display “Thank You” message by using a Toast.

```
package example.abhiandroid.textinputlayoutexample;

import android.graphics.Typeface;
import android.support.design.widget.TextInputLayout;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

TextInputLayout emailTextInputLayout, passwordTextInputLayout;
EditText email, password;
Button signIn;

@Override
protected void onCreate(Bundle savedInstanceState) {
```

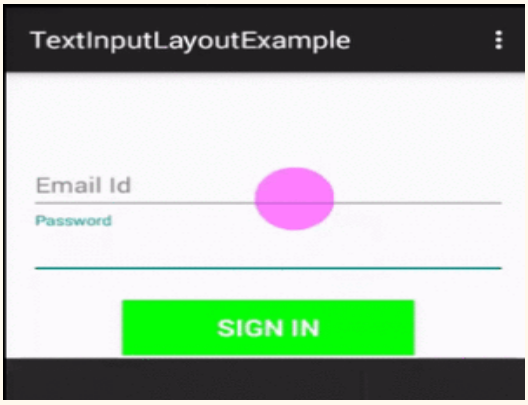
```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
// get the reference of View's
emailTextInputLayout = (TextInputLayout) findViewById(R.id.emailTextInputLayout);
passwordTextInputLayout = (TextInputLayout) findViewById(R.id.passwordTextInputLayout);
email = (EditText) findViewById(R.id.emailEditText);
password = (EditText) findViewById(R.id.passwordEditText);
signIn = (Button) findViewById(R.id.signInButton);
// perform click event on sign In Button
signIn.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
if (validate(email, emailTextInputLayout) && validate(password, passwordTextInputLayout)) {
//  display a Thank You message
Toast.makeText(getApplicationContext(), "Thank You", Toast.LENGTH_LONG).show();

}
}
});
}

// validate fields
private boolean validate(EditText editText, TextInputLayout textInputLayout) {
if (editText.getText().toString().trim().length() > 0) {
return true;
}
editText.requestFocus(); // set focus on fields
textInputLayout.setError("Please Fill This.!!!"); // set error message
return false;
}
}
```

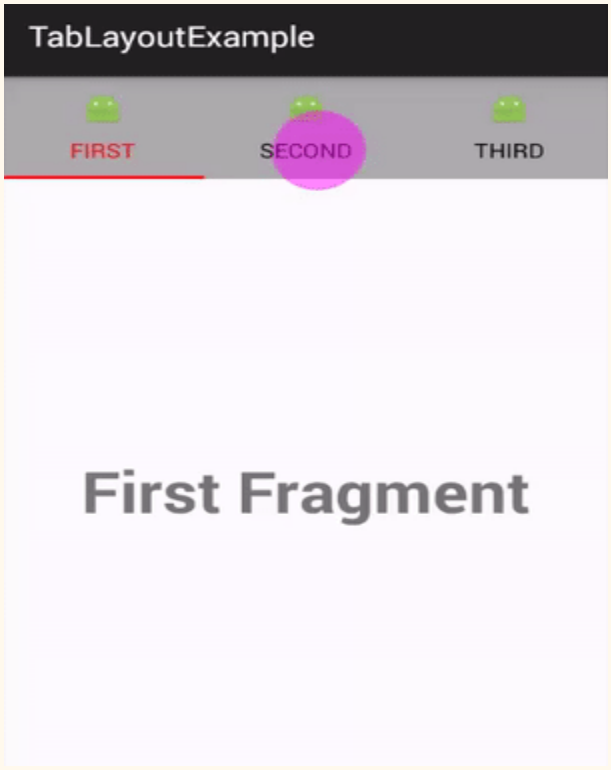
Output:

Now run the app and you will see login form on the screen. Click on email & password and you will see label text is Floating which looks beautiful.



TabLayout

In Android TabLayout is a new element introduced in Design Support library. It provides horizontal layout to display tabs on the screen. We can display more screens in a single screen using tabs. We can quickly swipe between the tabs. TabLayout is basically view class required to be added into our layout(xml) for creating Sliding Tabs. We use different methods of TabLayout to create, add and manage the tabs.



Special Note: TabLayout is used to display tabs on the screen. We can create sliding as well as non sliding tabs by using TabLayout. If we need simple tabs without sliding then we replace the layout with the fragment on tab selected listener event and if we need sliding tabs then we use Viewpager.

Basic TabLayout XML Code:

```
<android.support.design.widget.TabLayout
android:id="@+id/simpleTabLayout"
android:layout_width="match_parent"
android:layout_height="wrap_content"
app:tabMode="fixed"
app:tabGravity="fill"/>
```

Important Methods Of TabLayout:

Let's we discuss some important methods of TabLayout that may be called in order to manage the TabLayout.

1. newTab(): This method is used to create and return a new TabLayout.Tab.

Below we create a new tab and set the text and icon for the tab.

```
TabLayout   tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
TabLayout.Tab firstTab = tabLayout.newTab(); // Create a new Tab names
firstTab.setText("First Tab"); // set the Text for the first Tab
firstTab.setIcon(R.drawable.ic_launcher); // set an icon for the first tab
```

2. addTab(Tab tab): This method is used to add a tab in the TabLayout. By using this method we add the tab which we created using newTab() method in the TabLayout. The tab will be added at the end of the list and If it is the first tab to be added then it will become the selected tab.

Below we firstly create a new tab and then add it in the TabLayout.

```
TabLayout   tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
TabLayout.Tab firstTab = tabLayout.newTab(); // Create a new Tab names "First Tab"
firstTab.setText("First Tab"); // set the Text for the first Tab
firstTab.setIcon(R.drawable.ic_launcher); // set an icon for the first tab
tabLayout.addTab(firstTab); // add  the tab to the TabLayout
```

3. addTab(Tab tab, boolean setSelected): This method is used to add a tab in the TabLayout and set the state for the tab. By using this method we add the tab which we created

using newTab() method in the TabLayout. In this method we also set the state of the tab whether it is selected or not.

Below we firstly create a new tab and then add it in the TabLayout and set the true value for setSelected parameter that makes it selectable.

```
TabLayout   tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
TabLayout.Tab firstTab = tabLayout.newTab(); // Create a new Tab names "First Tab"
firstTab.setText("First Tab"); // set the Text for the first Tab
firstTab.setIcon(R.drawable.ic_launcher); // set an icon for the first tab
tabLayout.addTab(firstTab,true); // add  the tab in the TabLayout and makes it selectable
```

4. addTab(Tab tab, int position): This method is used to add a tab in the TabLayout and set the state for the tab. By using this method we add the tab which we created using newTab() method in the TabLayout. The tab will be inserted at the given position. If it is the first tab to be added then it will become the selected tab.

Below we firstly create a new tab and then add it in the TabLayout at a specific position.

```
TabLayout   tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
TabLayout.Tab firstTab = tabLayout.newTab(); // Create a new Tab names "First Tab"
firstTab.setText("First Tab"); // set the Text for the first Tab
firstTab.setIcon(R.drawable.ic_launcher); // set an icon for the first tab
tabLayout.addTab(firstTab,2); // add  the tab in the TabLayout at specific position
```

5. addTab(Tab tab, int position, boolean setSelected): This method is used to add a tab at a specific position and set the state of the tab. By using this method we add the tab which we created using newTab() method in the TabLayout. The tab will be inserted at the defined position and a Boolean value used to set the state of the tab. True value is used to make the tab selectable.

Below we firstly create a tab and then add it in the TabLayout at a specific position and we also set true value to make the tab selectable

```
TabLayout   tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
TabLayout.Tab firstTab = tabLayout.newTab(); // Create a new Tab names "First Tab"
firstTab.setText("First Tab"); // set the Text for the first Tab
firstTab.setIcon(R.drawable.ic_launcher); // set an icon for the first tab
tabLayout.addTab(firstTab,2,true); // add  the tab at specified position in the TabLayout
and makes it selectable
```

6. getSelectedTabPosition(): This method is used to get the position of the current selected tab. This method returns an int type value for the position of the selected tab. It returns -1 if there isn't a selected tab.

Below we get the current selected tab position.

```
TabLayout   tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
int selectedTabPosition = tabLayout.getSelectedTabPosition(); // get the position for the
current selected tab
```

7. getTabAt(int index): This method is used to get the tab at the specified index. This method returns TabLayout.Tab.

Below we get the tab at 1th index.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
TabLayout.Tab tab = tabLayout.getTabAt(1); //  get the tab at 1th in index
```

8. getTabCount(): This method is used to get the number of tabs currently registered with the action bar. This method returns an int type value for the number of total tabs.

Below we get the total number of tabs currently registered with the action bar.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
```

```
of TabLayout
int tabCount= tabLayout.getTabCount(); // get the total number of tabs currently registered
with the action bar.
```

9. setTabGravity(int gravity): This method is used to set the gravity to use when laying out the tabs.

Below we set the gravity for the tabs.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
tabLayout.setTabGravity(TabLayout.GRAVITY_CENTER); // set the gravity to use when laying out
the tabs.
```

10. getTabGravity(): This method is used to get the current gravity used for laying out tabs. This method returns the gravity which we set using setTabGravity(int gravity) method.

Below we firstly set the gravity and then get the current gravity used for laying out tabs.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
tabLayout.setTabGravity(TabLayout.GRAVITY_CENTER); // set the gravity to use when laying out
the tabs.
int gravity=tabLayout.getTabGravity(); // get the current gravity used for laying out tabs
```

11. setTabMode(int mode): This method is used to set the behavior mode for the Tabs in this layout. The valid input options are:

MODE_FIXED: Fixed tabs display all tabs concurrently and are best used with content that benefits from quick pivots between tabs.

MODE_SCROLLABLE: Scrollable tabs display a subset of tabs at any given moment and it can contain longer tab labels and a larger number of tabs. They are best used for browsing contexts in touch interfaces when users don't need to directly compare the tab labels. This mode is commonly used with a ViewPager.

Below we set the behaviour mode for the tabs.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
tabLayout.setTabMode(TabLayout.MODE_SCROLLABLE); // set the behaviour mode for the tabs
```

12. getTabMode(): This method is used to get the current mode of TabLayout. This method returns an int type value which we set using setTabMode(int mode) method.

Below we firstly set the tab mode and then get the current mode of the TabLayout.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
tabLayout.setTabMode(TabLayout.MODE_SCROLLABLE); // set the behaviour mode for the tabs
int mode=tabLayout.getTabMode(); // get the current mode of the TabLayout
```

13. setTabTextColors(int normalColor, int selectedColor): This method is used to set the text colors for the different states (normal, selected) of the tabs.

Below we set the tab text colors for the both states of the tab.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
tabLayout.setTabTextColors(Color.RED,Color.WHITE); // set the tab text colors for the both
states of the tab.
```

14. getTabTextColors(): This method is used to get the text colors for the different states (normal, selected) of the tabs. This method returns the text color which we set using setTabTextColors(int normalColor, int selectedColor) method.

Below we firstly set the text colors and then get the text colors for the both states of the tab.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
tabLayout.setTabTextColors(Color.RED,Color.WHITE); // set the tab text colors for the both
states of the tab.
```

```
ColorStateList colorStateList=tabLayout.getTabTextColors(); // get the text colors for the
both states of the tab
```

15. removeAllTabs(): This method is used to remove all tabs from the action bar and deselect the current tab.

Below we remove all the tabs from the action bar and deselect the current tab.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
tabLayout.removeAllTabs(); // remove all the tabs from the action bar and deselect the
current tab
```

16. setOnTabSelectedListener(OnTabSelectedListener listener): This method is used to add a listener that will be invoked when tab selection changes.

Below we show how to use addOnTabSelectedListener of TabLayout.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
tabLayout.setOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
@Override
public void onTabSelected(TabLayout.Tab tab) {
// called when tab selected
}

@Override
public void onTabUnselected(TabLayout.Tab tab) {
// called when tab unselected
}

@Override
public void onTabReselected(TabLayout.Tab tab) {
// called when a tab is reselected
}
});
```

17. removeTab(Tab tab): This method is used to remove a tab from the layout. In this method we pass the TabLayout.Tab object to remove the tab from the layout. If the removed tab

was selected then it will be automatically deselected and another tab will be selected if present in the TabLayout.

Below we firstly create and add a tab and then remove it from the TabLayout.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
TabLayout.Tab firstTab = tabLayout.newTab(); // Create a new Tab names "First Tab"
firstTab.setText("First Tab"); // set the Text for the first Tab
firstTab.setIcon(R.drawable.ic_launcher); // set an icon for the first tab
tabLayout.addTab(firstTab); // add  the tab at in the TabLayout
tabLayout.removeTab(firstTab); // remove the tab from the TabLayout
```

18. removeTabAt(int position): This method is used to remove a tab from the layout. In this method we pass the position of the tab that we want to remove from the layout. If the removed tab was selected then it will be automatically deselected and another tab will be selected if present in the TabLayout.

Below we remove the tab from a specified position of TabLayout.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
tabLayout.removeTabAt(1); // remove the tab from a specified position of TabLayout
```

19. setSelectedTabIndicatorColor(int color): This method is used to set the tab indicator's color for the currently selected tab.

Below we set the red color for the selected tab indicator.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
tabLayout.setSelectedTabIndicatorColor(Color.RED); // set the red color for the selected tab
indicator.
```

20. setSelectedTabIndicatorHeight(int height): This method is used to set the tab indicator's height for the currently selected tab.

Below we set the height for the selected tab's indicator.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
tabLayout.setSelectedTabIndicatorHeight(2); // set the height for the selected tab's
indicator
```

21. setupWithViewPager(ViewPager viewPager): This method is used for setting up the TabLayout with ViewPager. ViewPager is mainly used for creating Sliding tabs.

Below we set the TabLayout with the ViewPager.

```
ViewPager viewPager = (ViewPager) findViewById(R.id.viewpager); // get the reference of
ViewPager
TabLayout tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout); // get the reference
of TabLayout
tabLayout.setupWithViewPager(viewPager); // set the TabLayout with the ViewPager.
```

Attributes of TabLayout:

Now let's we discuss some common attributes of a TabLayout that helps us to configure it in our layout (xml).

1. id: id attribute is used to uniquely identify a TabLayout.

2. support.design:tabBackground: This attribute is used to set the background of the tabs. We can set a color or drawable in the background of tabs.

Below we set the Black color for the background of the tabs.

```
<android.support.design.widget.TabLayout
android:id="@+id/simpleTabLayout"
android:layout_width="match_parent"
android:layout_height="wrap_content"
app:tabBackground="@android:color/darker_gray" /><!-- set the Black color for the background
of the tabs.-->
```

3. support.design:tabGravity: This attribute is used to set the gravity to use when laying out the tabs. We can also set gravity programmatically means in java class using setTabGravity(int gravity) method.

Below we set the gravity for the tabs.

```
<android.support.design.widget.TabLayout
android:id="@+id/simpleTabLayout "
android:layout_width="match_parent"
android:layout_height="wrap_content"
app:tabGravity="fill" /><!-- set the gravity for the tabs.-->
```

4. support.design:tabIndicatorColor: This attribute is used to set the Color of the indicator used to show the currently selected tab. We can also set color programmatically means in java class using setSelectedTabIndicatorColor(int color) method.

Below we set the red color for the selected tab indicator.

```
<android.support.design.widget.TabLayout
android:id="@+id/simpleTabLayout "
android:layout_width="match_parent"
android:layout_height="wrap_content"
app:tabIndicatorColor="#f00" /><!-- set the red color for the selected tab indicator.-->
```

5. support.design:tabIndicatorHeight: This attribute is used to set the height of the indicator used to show the currently selected tab. We can also set height programmatically means in java class using setSelectedTabIndicatorHeight(int height) method.

Below we set the height for the selected tab's indicator.

```
<android.support.design.widget.TabLayout
android:id="@+id/simpleTabLayout "
android:layout_width="match_parent"
android:layout_height="wrap_content"
app:tabIndicatorHeight="2dp" />
<!-- set the height for the selected tab's indicator -->
```

6. support.design:tabMaxWidth: This attribute is used to set the maximum width for the tabs.

Below we set the maximum width value for the tabs.

```
<android.support.design.widget.TabLayout
android:id="@+id/simpleTabLayout "
android:layout_width="match_parent"
android:layout_height="wrap_content"
app:tabMaxWidth="100dp" />
<!-- set the maximum width value for the  tabs. -->
```

7. support.design:tabMinWidth: This attribute is used to set the minimum width for the tabs.

Below we set the minimum width value for the tabs.

```
<android.support.design.widget.TabLayout
android:id="@+id/simpleTabLayout "
android:layout_width="match_parent"
android:layout_height="wrap_content"
app:tabMinWidth="50dp" />
<!-- set the minimum width value for the tabs  -->
```

8. support.design:tabMode: This attribute is used to set the behavior mode for the Tabs in this layout. We can also set the mode programmatically means in java class using setTabMode(int mode) method.

Below we set the behaviour mode for the tabs.

```
<android.support.design.widget.TabLayout
android:id="@+id/simpleTabLayout "
android:layout_width="match_parent"
android:layout_height="wrap_content"
app:tabMode="fixed" />
<!-- set the behaviour mode for the tabs  -->
```

9. support.design:tabPadding: This attribute is used to set the padding along all edges of tabs.

android.support.design:tabPaddingBottom: Set padding along the bottom edge of the tabs.

android.support.design:tabPaddingEnd: Set padding along the end edge of the tabs.

android.support.design:tabPaddingStart: Set padding along the start edge of the tabs.

android.support.design:tabPaddingTop: Set padding along the top edge of the tabs.

Below we set the padding along all edges of the tabs.

```
<android.support.design.widget.TabLayout
android:id="@+id/simpleTabLayout "
android:layout_width="match_parent"
android:layout_height="wrap_content"
app:tabPadding="5dp" />
<!-- set the padding along all edges of the tabs  -->
```

10. support.design:tabSelectedTextColor: This attribute is used to set the text color to be applied to the currently selected tab. We can also set this programmatically using setTabTextColors(int normalColor, int selectedColor) method.

Below we set the text color for the selected tab.

```
<android.support.design.widget.TabLayout
android:id="@+id/simpleTabLayout "
android:layout_width="match_parent"
android:layout_height="wrap_content"
app:tabSelectedTextColor="#fff" />
<!-- set the text color for the selected tab.  -->
```

11. support.design:tabTextColor: This method is used to set the default text color to be applied to tabs. . We can also set this programmatically using setTabTextColors(int normalColor, int selectedColor) method.

Below we set the default text color for the tabs.

```
<android.support.design.widget.TabLayout
android:id="@+id/simpleTabLayout "
android:layout_width="match_parent"
android:layout_height="wrap_content"
app:tabTextColor="#f00" />
<!-- set red as default text color for the text of tabs  -->
```

TabLayout Example In Android Studio:

Example 1 of TabLayout:

Below is the first example of TabLayout in which we display three non-sliding tabs. In this example we define a TabLayout and a FrameLayout in our xml file. In this example we create and add 3 tabs in the TabLayout with the help of different methods of TabLayout. After that we implement setOnTabSelectedListener event and replace the FrameLayout with the fragment's according to selected tab.

Step 1: Create a new project and name it TabLayoutExample

Step 2: Open build.gradle and add Design support library dependency.

```
apply plugin: 'com.android.application'

android {
compileSdkVersion 23
buildToolsVersion "23.0.2"

defaultConfig {
applicationId "com.abhiandroid.tablayoutexample"
minSdkVersion 15
targetSdkVersion 23
versionCode 1
versionName "1.0"
}
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
}
}
}

dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
compile 'com.android.support:appcompat-v7:23.1.0'
compile 'com.android.support:design:23.1.0' // design support library
}
```

Step 3: Open res -> layout ->activity_main.xml (or) main.xml and add following code:

In this step we add the code for displaying TabLayout and ViewPager in our xml file.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
tools:context=".MainActivity">

<android.support.design.widget.TabLayout
android:id="@+id/simpleTabLayout"
android:layout_width="match_parent"
android:layout_height="wrap_content"
app:tabBackground="@android:color/darker_gray"
app:tabIndicatorColor="#f00"
app:tabSelectedTextColor="#f00"
app:tabTextColor="#000" />

<FrameLayout
android:id="@+id/simpleFrameLayout"
android:layout_width="match_parent"
android:layout_height="match_parent" />

</LinearLayout>
```

Step 4: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code for initiate the FrameLayout and TabLayout. After that we create and add 3 tabs in the TabLayout. Finally we implement setOnTabSelectedListener event and replace the FrameLayout with the fragment's according to selected tab.

```
package com.abhiandroid.tablayoutexample;

import android.os.Bundle;
import android.support.design.widget.TabLayout;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentTransaction;
import android.support.v7.app.AppCompatActivity;
import android.widget.FrameLayout;

public class MainActivity extends AppCompatActivity {

FrameLayout simpleFrameLayout;
TabLayout tabLayout;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
// get the reference of FrameLayout and TabLayout
simpleFrameLayout = (FrameLayout) findViewById(R.id.simpleFrameLayout);
tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout);
// Create a new Tab named "First"
TabLayout.Tab firstTab = tabLayout.newTab();
firstTab.setText("First"); // set the Text for the first Tab
firstTab.setIcon(R.drawable.ic_launcher); // set an icon for the
// first tab
tabLayout.addTab(firstTab); // add  the tab at in the TabLayout
// Create a new Tab named "Second"
TabLayout.Tab secondTab = tabLayout.newTab();
secondTab.setText("Second"); // set the Text for the second Tab
secondTab.setIcon(R.drawable.ic_launcher); // set an icon for the second tab
tabLayout.addTab(secondTab); // add  the tab  in the TabLayout
// Create a new Tab named "Third"
TabLayout.Tab thirdTab = tabLayout.newTab();
thirdTab.setText("Third"); // set the Text for the first Tab
thirdTab.setIcon(R.drawable.ic_launcher); // set an icon for the first tab
tabLayout.addTab(thirdTab); // add  the tab at in the TabLayout


// perform setOnTabSelectedListener event on TabLayout
tabLayout.setOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
@Override
public void onTabSelected(TabLayout.Tab tab) {
// get the current selected tab's position and replace the fragment accordingly
Fragment fragment = null;
```

```
switch (tab.getPosition()) {
case 0:
fragment = new FirstFragment();
break;
case 1:
fragment = new SecondFragment();
break;
case 2:
fragment = new ThirdFragment();
break;
}
FragmentManager fm = getSupportFragmentManager();
FragmentTransaction ft = fm.beginTransaction();
ft.replace(R.id.simpleFrameLayout, fragment);
ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN);
ft.commit();
}

@Override
public void onTabUnselected(TabLayout.Tab tab) {

}

@Override
public void onTabReselected(TabLayout.Tab tab) {

}
});
}
}
```

Step 5: Now we need 3 fragments and 3 xml layouts for three tabs. So create three fragments by right click on your package folder and create classes and name them as FirstFragment, SecondFragment and ThirdFragment and add the following code respectively.

FirstFragment.class

```
package com.abhiandroid.tablayoutexample;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class FirstFragment extends Fragment {

public FirstFragment() {
// Required empty public constructor
}
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate(R.layout.fragment_first, container, false);
}

}
```

SecondFragment.class

```
package com.abhiandroid.tablayoutexample;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class SecondFragment extends Fragment {

public SecondFragment() {
// Required empty public constructor
}

@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate(R.layout.fragment_second, container, false);
}

}
```

ThirdFragment.class

```
package com.abhiandroid.tablayoutexample;

import android.os.Bundle;
```

```
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class ThirdFragment extends Fragment {

public ThirdFragment() {
// Required empty public constructor
}

@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate(R.layout.fragment_third, container, false);
}

}
```

Step 6: Now create 3 xml layouts by right clicking on res/layout -> New -> Layout Resource File and name them as fragment_first, fragment_second and fragment_third and add the following code in respective files.

Here we will design the basic simple UI for all the three tabs.

Fragment_first.xml

```
<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"
tools:context="info.androidhive.materialtabs.fragments.OneFragment">

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:text="First Fragment"
android:textSize="40dp"
android:textStyle="bold" />

</RelativeLayout>
```

Fragment_second.xml

```
<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"
tools:context="info.androidhive.materialtabs.fragments.OneFragment">

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:text="Second Fragment"
android:textSize="40dp"
android:textStyle="bold" />

</RelativeLayout>
```

Fragment_third.xml

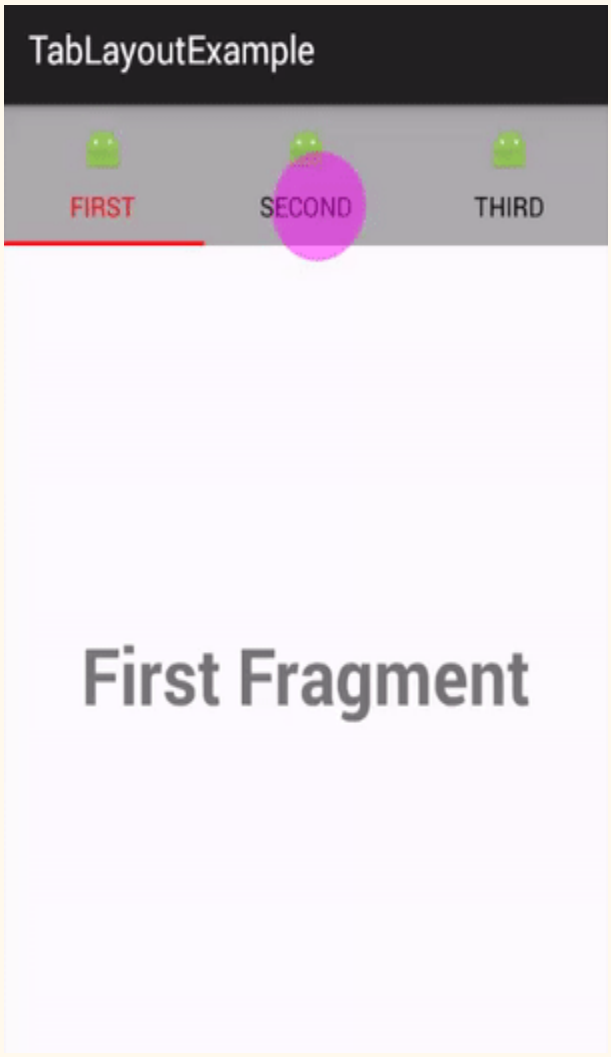
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="info.androidhive.materialtabs.fragments.OneFragment">

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:text="Third Fragment"
android:textSize="40dp"
android:textStyle="bold" />

</RelativeLayout>
```

Example 2 of TabLayout Using ViewPager:

Below is the example of TabLayout in which we display sliding tabs with the help of ViewPager. In this example we define a TabLayout and a ViewPager in our xml file. In this example we create and add 3 tabs in the TabLayout with the help of different methods of TabLayout. After that we create a PagerAdapter to set up the TabLayout with ViewPager.



Step 1: Create a new project and name it TabLayoutExample

Step 2: Open build.gradle and add Design support library dependency.

```
apply plugin: 'com.android.application'

android {
compileSdkVersion 23
buildToolsVersion "23.0.2"
```

```
defaultConfig {
applicationId "com.abhiandroid.tablayoutexample"
minSdkVersion 15
targetSdkVersion 23
versionCode 1
versionName "1.0"
}
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
}
}
}

dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
compile 'com.android.support:appcompat-v7:23.1.0'
compile 'com.android.support:design:23.1.0' // design support library
}
```

Step 3: Open res -> layout ->activity_main.xml (or) main.xml and add following code:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
tools:context=".MainActivity">

<android.support.design.widget.TabLayout
android:id="@+id/simpleTabLayout"
android:layout_width="match_parent"
android:layout_height="wrap_content"
app:tabBackground="@android:color/darker_gray"
app:tabIndicatorColor="#f00"
app:tabSelectedTextColor="#f00"
app:tabTextColor="#000" />


<android.support.v4.view.ViewPager
android:id="@+id/simpleViewPager"
android:layout_width="match_parent"
android:layout_height="match_parent"
app:layout_behavior="@string/appbar_scrolling_view_behavior" />
</LinearLayout>
```

Step 4: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code for initiate the ViewPager and TabLayout. After that we create and add 3 tabs in the TabLayout. Finally we set an Adapter named PagerAdapter to set up the TabLayout with ViewPager.

```
package com.abhiandroid.tablayoutexample;

import android.os.Bundle;
import android.support.design.widget.TabLayout;
import android.support.v4.view.ViewPager;
import android.support.v7.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

ViewPager simpleViewPager;
TabLayout tabLayout;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
// get the reference of ViewPager and TabLayout
simpleViewPager = (ViewPager) findViewById(R.id.simpleViewPager);
tabLayout = (TabLayout) findViewById(R.id.simpleTabLayout);
// Create a new Tab named "First"
TabLayout.Tab firstTab = tabLayout.newTab();
firstTab.setText("First"); // set the Text for the first Tab
firstTab.setIcon(R.drawable.ic_launcher); // set an icon for the
// first tab
tabLayout.addTab(firstTab); // add  the tab at in the TabLayout
// Create a new Tab named "Second"
TabLayout.Tab secondTab = tabLayout.newTab();
secondTab.setText("Second"); // set the Text for the second Tab
secondTab.setIcon(R.drawable.ic_launcher); // set an icon for the second tab
tabLayout.addTab(secondTab); // add  the tab  in the TabLayout
// Create a new Tab named "Third"
TabLayout.Tab thirdTab = tabLayout.newTab();
thirdTab.setText("Third"); // set the Text for the first Tab
thirdTab.setIcon(R.drawable.ic_launcher); // set an icon for the first tab
tabLayout.addTab(thirdTab); // add  the tab at in the TabLayout

PagerAdapter adapter = new PagerAdapter
(getSupportFragmentManager(), tabLayout.getTabCount());
simpleViewPager.setAdapter(adapter);
// addOnPageChangeListener event change the tab on slide
simpleViewPager.addOnPageChangeListener(new
TabLayout.TabLayoutOnPageChangeListener(tabLayout));
}
}
```

Step 5: Create a new Class named PagerAdapter and extend FragmentStatePagerAdapter in the class. In this step we create a PagerAdapter class and extends FragmentStatePagerAdapter In the class for setting the TabLayout with the ViewPager.

```
package com.abhiandroid.tablayoutexample;

import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentStatePagerAdapter;

public class PagerAdapter extends FragmentStatePagerAdapter {
int mNumOfTabs;

public PagerAdapter(FragmentManager fm, int NumOfTabs) {
super(fm);
this.mNumOfTabs = NumOfTabs;
}

@Override
public Fragment getItem(int position) {

switch (position) {
case 0:
FirstFragment tab1 = new FirstFragment();
return tab1;
case 1:
SecondFragment tab2 = new SecondFragment();
return tab2;
case 2:
ThirdFragment tab3 = new ThirdFragment();
return tab3;
default:
return null;
}
}

@Override
public int getCount() {
return mNumOfTabs;
}
}
```

Step 6: Now we need 3 fragments and 3 xml layouts for three tabs. So create three fragments by right click on your package folder and create classes and name them as FirstFragment, SecondFragment and ThirdFragment and add the following code respectively.

FirstFragment.class

```
package com.abhiandroid.tablayoutexample;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class FirstFragment extends Fragment {

public FirstFragment() {
// Required empty public constructor
}

@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate(R.layout.fragment_first, container, false);
}

}
```

SecondFragment.class

```
SecondFragment.class

package com.abhiandroid.tablayoutexample;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class SecondFragment extends Fragment {

public SecondFragment() {
// Required empty public constructor
}

@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
}
```

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate(R.layout.fragment_second, container, false);
}

}
```

ThirdFragment.class

```
package com.abhiandroid.tablayoutexample;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class ThirdFragment extends Fragment {

public ThirdFragment() {
// Required empty public constructor
}

@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate(R.layout.fragment_third, container, false);
}

}
```

Step 7: Now create 3 xml layouts by right clicking on res/layout -> New -> Layout Resource File and name them as fragment_first, fragment_second and fragment_third and add the following code in respective files.

Here we will design the basic simple UI for all the three tabs.

Fragment_first.xml

```
<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"
tools:context="info.androidhive.materialtabs.fragments.OneFragment">

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:text="First Fragment"
android:textSize="40dp"
android:textStyle="bold" />

</RelativeLayout>
```

Fragment_second.xml

```
<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"
tools:context="info.androidhive.materialtabs.fragments.OneFragment">

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:text="Second Fragment"
android:textSize="40dp"
android:textStyle="bold" />

</RelativeLayout>
```

Fragment_third.xml

```
fragment_third.xml

<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"
```

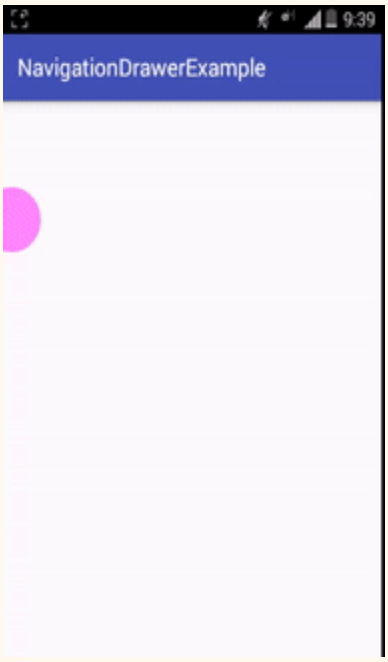
```
tools:context="info.androidhive.materialtabs.fragments.OneFragment">

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:text="Third Fragment"
android:textSize="40dp"
android:textStyle="bold" />

</RelativeLayout>
```

Navigation Drawer

In Android, Navigation Drawer is a panel that displays App's Navigation option from the left edge of the screen. It is one of the most important and useful UI pattern introduced by the Google for developing Android app.Navigation drawer is a side menu that helps us to organise the navigation inside our app. It is a uniform way to access different pages and information inside our app. It is hidden most of the time but is revealed when we swipes from left edge of the screen or whenever we click on menu/app icon in the action bar.



Need of Navigation Drawer:

We might have noticed that lot of our Android Applications introduced a sliding panel menu for navigating between major modules of our application. This kind of UI was done by using third party libraries where a ListView and swiping gestures used to achieve this type of UI. But now Android itself officially introduced sliding panel menu by introducing a newer concept called

Navigation Drawer. In Navigation Drawer we combine NavigationView and DrawerLayout to achieve the desired output.

Drawer Layout In Android:

In Android, DrawerLayout acts as top level container for window content that allows for interactive “drawer” views to be pulled out from one or both vertical edges of the window. Drawer position and layout is controlled by using layout_gravity attribute on child views corresponding to which side of view we want the drawer to emerge from like left to right.

Basic Drawer Layout XML Code

For adding a Navigation Drawer, declare your UI(user interface) with a DrawerLayout object as the root(parent) view of your layout. Inside this DrawerLayout add one view that contains the main content of the screen means primary layout that displays when the drawer is hidden and other view that contains the contents for the navigation drawer.

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!-- The main content view displays when the drawer is hidden -->
    <FrameLayout
        android:id="@+id/content_frame"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <!-- The navigation drawer -->
    <android.support.design.widget.NavigationView
  android:id="@+id/navigation"
  android:layout_width="wrap_content"
  android:layout_height="match_parent"
  android:layout_gravity="start"
  app:menu="@menu/nav_items"/>

</android.support.v4.widget.DrawerLayout>
```

In above code snippet DrawerLayout is the root view of our Layout in which we have other layouts and content.

FrameLayout is used to hold the page content shown through different fragments

The NavigationView is the “real” menu of our app. The menu items are written in nav_items file.

Important Methods Of Drawer Layout

Let’s we discuss some important methods of Navigation Drawer that may be called in order to manage the Navigation Drawer.

1. closeDrawer(int gravity): This method is used to close the drawer view by animating it into view. We can close a drawer by passing END gravity to this method.

Below we show how to close the drawer view. We can close a drawer on any click or other event of view.

```
DrawerLayout
dLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
// initiate a DrawerLayout
dLayout.closeDrawer(GravityCompat.END);
// close a Drawer
```

2. closeDrawers(): This method is used to close all the currently open drawer views by animating them out of view. We mainly use this method on click of any item of Navigation View.

Below we close all the currently open drawer views.

```
DrawerLayout
dLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
// initiate a DrawerLayout
dLayout.closeDrawers();
// close all the Drawers
```

3. isDrawerOpen(int drawerGravity): This method is used to check the drawer view is currently open or not. It returns true if the drawer view is open otherwise it returns false.

Below we check DrawerLayout is open or not. It returns true or false value.

```
DrawerLayout
dLayout = (DrawerLayout)
findViewById(R.id.drawer_layout);
// initiate a DrawerLayout
Boolean
isOpen = dLayout.isDrawerOpen(GravityCompat.END);
// check DrawerLayout view is open or not
```

4. isDrawerVisible(int drawerGravity): This method is used to check the drawer view is currently visible on screen or not. It returns true if the drawer view is visible otherwise it returns false.

Below we check DrawerLayout is visible or not. It returns true or false value.

```
DrawerLayout
dLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
// initiate a DrawerLayout
Boolean
isVisible = dLayout.isDrawerVisible(GravityCompat.END);
// check DrawerLayout view is visible or
not
```

5. openDrawer(int gravity): This method is used to open the drawer view by animating it into view. We can open a Drawer by passing START gravity to this method.

Below we show how to open a Drawer Layout view. We can open a drawer on any click or other event of view.

```
DrawerLayout
dLayout = (DrawerLayout)
findViewById(R.id.drawer_layout);
// initiate a DrawerLayout
dLayout.openDrawer(GravityCompat.START);
// open a Drawer
```

Navigation View:

In Android, Navigation View represents a standard navigation menu for our application. The menu contents can be populated by a menu resource file.

Important Methods Of Navigation View:

Let's we discuss some important methods of Navigation View that may be called in order to manage the Navigation View.

1.setNavigationItemSelectedListener(NavigationView.OnNavigationItemSelectedListener listener): This method is used to set a listener that will be notified when a menu item is selected.

Below we shows the code how to use this listener.

```
NavigationView navView = (NavigationView) findViewById(R.id.navigation); // initiate a
Navigation View
// implement setNavigationSelectedListener event
navView.setNavigationItemSelectedListener(new
NavigationView.OnNavigationItemSelectedListener() {
@Override
public boolean onNavigationItemSelected(MenuItem menuItem) {

// add code here what you need on click of items.
return false;
}
});
```

2. setItemBackground(Drawable itemBackground): This method is used to set the resource in the background of the menu item. In this method we pass the drawable object for setting background of menu item.

Below we set the resource in the background of the menu items.

```
NavigationView navView = (NavigationView) findViewById(R.id.navigation); // initiate a
Navigation View
navView.setItemBackground(getResources().getDrawable(R.drawable.background)); // set a
resource in background of menu items.
```

3. setItemBackgroundResource(int resId): This method is used to set the resource in the background of the menu item. In this method we pass the id of the resource to set in the background of menu item.

Below we set the resource in the background of the menu items.

```
NavigationView navView = (NavigationView) findViewById(R.id.navigation); // initiate a
Navigation View
navView.setItemBackgroundResource(R.drawable.background); // set a resource in background of
menu items.
```

From XML: We can also set the background of menu items from our xml file in which we define NavigationView.

Below we set the resource in the background of menu items.

```
<android.support.design.widget.NavigationView
    android:id="@+id/navigation"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    app:itemBackground="@drawable/background"
    app:menu="@menu/nav_items" /><!-- background resource for menu items -->
```

4. getItemBackground(): This method returns the background drawable for our menu items.

Below we firstly set a resource in the background of menu items and then get the same drawable.

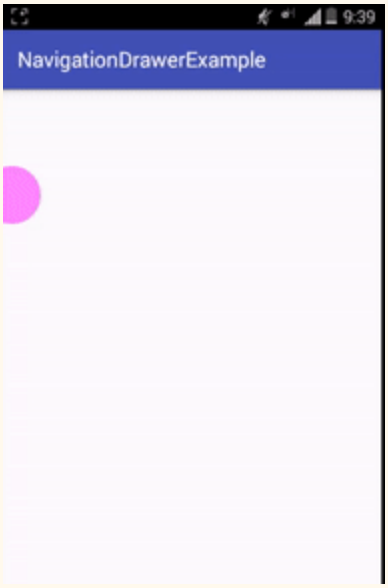
```
NavigationView navView = (NavigationView) findViewById(R.id.navigation); // initiate a
Navigation View
navView.setItemBackground(getResources().getDrawable(R.drawable.background)); // set a
resource in background of menu items.
Drawable backgroundDrawable=navView.getItemBackground(); // get the background drawable
```

Navigation Drawer Example in Android Studio:

Below is the example of Navigation Drawer in which we display App's Navigation option from left edge of the screen. In this example we use Drawer Layout and Navigation View in our XML file. Drawer Layout is the root layout in which we define a FrameLayout and a Navigation View. In Navigation View we set the items from menu file and FrameLayout is used to replace the Fragments on the click of menu items. Whenever a user click on menu item, Fragment is

replaced according to menu item's id and a toast message with menu item title is displayed on the screen. We also create three Fragments that should be replaced on click of menu items.

Below you can see final output and step by step explanation of Navigation drawer example in Android Studio.



Step 1: Create a new project and name it NavigationDrawerExample.

Step 2: Open Gradle Scripts > build.gradle and add Design support library dependency.

apply plugin: 'com.android.application'

```
android {
compileSdkVersion 24
buildToolsVersion "24.0.1"
defaultConfig {
applicationId "abhiandroid.navigationdrawerexample"
minSdkVersion 15
targetSdkVersion 24
versionCode 1
versionName "1.0"
}
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
}
}
```

```
}
dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
testCompile 'junit:junit:4.12'
compile 'com.android.support:appcompat-v7:24.1.1'
compile 'com.android.support:design:24.1.1' // design support Library
}
```

Step 3: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we define a DrawerLayout that is the parent layout in which we define a FrameLayout and a Navigation View. In Navigation View we set the items from menu file named "nav_items" and FrameLayout is used to replace the Fragments on the click of menu items.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context=".MainActivity">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <!-- Let's add fragment -->
        <FrameLayout
            android:id="@+id/frame"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
    </LinearLayout>
    <!--
         Navigation view to show the menu items
    -->
    <android.support.design.widget.NavigationView
        android:id="@+id/navigation"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:menu="@menu/nav_items" />
</android.support.v4.widget.DrawerLayout>
```

Step 4: Open res -> menu -> nav_items.xml and add following code:

In this step we create the three menu items that should be displayed in Drawer.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <!--
                Add menu items here
                Menu items with icon and Title
        -->
        <item
            android:id="@+id/first"
            android:icon="@drawable/ic_launcher"
            android:title="First Menu" />
        <item
            android:id="@+id/second"
            android:icon="@drawable/ic_launcher"
            android:title="Second Menu" />
        <item
            android:id="@+id/third"
            android:icon="@drawable/ic_launcher"
            android:title="Third Menu" />
    </group>
</menu>
```

Step 5: Open src -> package -> MainActivity.java

In this step we open the MainActivity and add the code for initiates the views(DrawerLayout, NavigationView and other views). After that we implement setNavigationItemSelectedListener event on NavigationView so that we can replace the Fragments according to menu item's id and a toast message with menu item's title is displayed on the screen. I have added comments in code to help you to understand the code easily so make you read the comments.

```
package abhiandroid.navigationdrawerexample;
import android.os.Bundle;
import android.support.design.widget.NavigationView;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentTransaction;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.AppCompatActivity;
import android.view.MenuItem;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
DrawerLayout dLayout;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.activity_main);
setNavigationDrawer(); // call method
}
private void setNavigationDrawer() {
dLayout = (DrawerLayout) findViewById(R.id.drawer_layout); // initiate a DrawerLayout
NavigationView navView = (NavigationView) findViewById(R.id.navigation); // initiate a
Navigation View
// implement setNavigationItemSelectedListener event on NavigationView
navView.setNavigationItemSelectedListener(new
NavigationView.OnNavigationItemSelectedListener() {
@Override
public boolean onNavigationItemSelected(MenuItem menuItem) {
Fragment frag = null; // create a Fragment Object
int itemId = menuItem.getItemId(); // get selected menu item's id
// check selected menu item's id and replace a Fragment Accordingly
if (itemId == R.id.first) {
frag = new FirstFragment();
} else if (itemId == R.id.second) {
frag = new SecondFragment();
} else if (itemId == R.id.third) {
frag = new ThirdFragment();
}
// display a toast message with menu item's title
Toast.makeText(getApplicationContext(), menuItem.getTitle(), Toast.LENGTH_SHORT).show();
if (frag != null) {
FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
transaction.replace(R.id.frame, frag); // replace a Fragment with Frame Layout
transaction.commit(); // commit the changes
dLayout.closeDrawers(); // close the all open Drawer Views
return true;
}
return false;
}
});
}
}
```

Step 6: Now we need 3 fragments and 3 xml layouts. So create three fragments by right click on your package folder and create classes and name them as FirstFragment, SecondFragment and ThirdFragment and add the following code respectively.

FirstFragment.class

```
package abhiandroid.navigationdrawerexample;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class FirstFragment extends Fragment {
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
```

```
// Inflate the layout for this fragment
return inflater.inflate(R.layout.fragment_first, container, false);
}
}
```

SecondFragment.class

```
package abhiandroid.navigationdrawerexample;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class SecondFragment extends Fragment {
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate(R.layout.fragment_second, container, false);
}
}
```

ThirdFrament.class

```
package abhiandroid.navigationdrawerexample;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class ThirdFragment extends Fragment {
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate(R.layout.fragment_third, container, false);
}
}
```

Step 7: Now create 3 xml layouts by right clicking on res/layout -> New -> Layout Resource File and name them as fragment_first, fragment_ second and fragment_third and add the following code in respective files.

Here we will design the basic simple UI by using TextView in all xml's.

Fragment_first.xml

```
<FrameLayout 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"
    tools:context="abhiandroid.navigationdrawerexample.FirstFragment">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:text="First Fragment"
        android:textSize="25sp" />
</FrameLayout>
```

fragment_second.xml

```
<FrameLayout 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"
    tools:context="abhiandroid.navigationdrawerexample.SecondFragment">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:text="Second Fragment"
        android:textSize="25sp" />
</FrameLayout>
```

Fragment_third.xml

```
fragment_third.xml
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
```

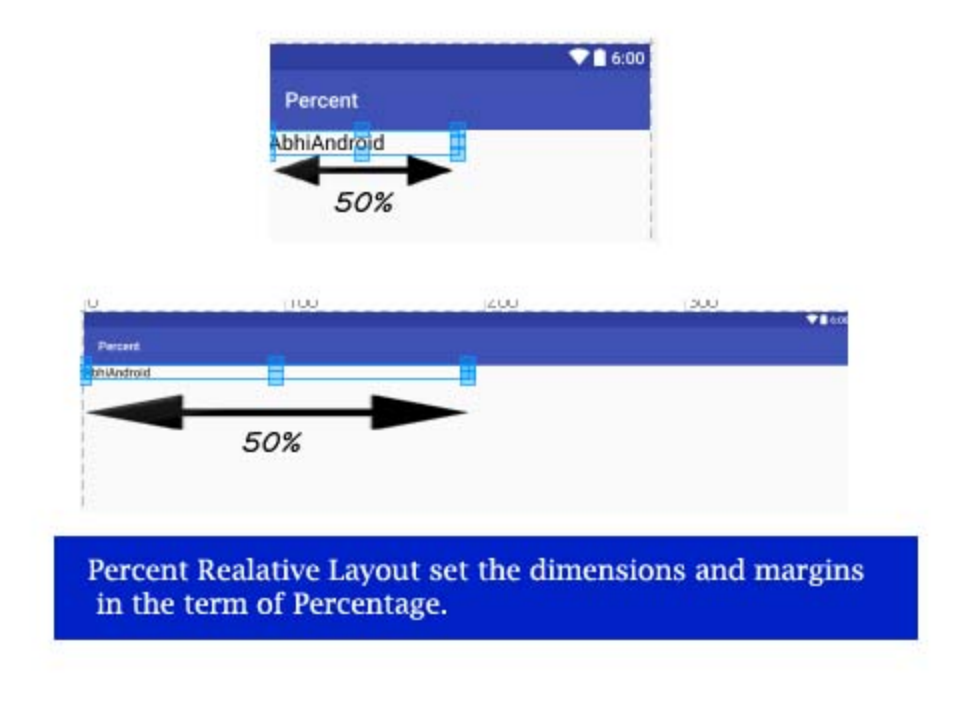
```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="abhiandroid.navigationdrawerexample.ThirdFragment">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:text="Third Fragment"
        android:textSize="25sp" />
</FrameLayout>
```

Output:

Now run the App and slide screen from left to right and you will see Menu option. On sliding right to left it will close back.

Percent Relative Layout

PercentRelativeLayout in Android is a subclass of RelativeLayout that supports percentage based margin and dimensions for Views(Button, TextView or any other view).



Percent Support Library: Percent Support Library provides a feature to set the dimensions and margins in the term of Percentage. It has two pre built Layouts- the PercentRelativeLayout and PercentFrameLayout. In this article we focused on PercentRelativeLayout. This library is pretty easy to use because it has same Relative Layout and FrameLayout that we are familiar with, just with some additional new functionalities.

Important Note – To use percentrelativelayout you need to add Percent Support Library dependency in build.gradle file.

Gradle Scripts > build.gradle (Module:App) -> inside dependencies

```
compile 'com.android.support:percent:23.3.0' // Percent Support Library
```

PercentRelativeLayout Vs Relative Layout In Android:

PercentRelativeLayout class extends from Relative Layout class so it supports all the features, attributes of RelativeLayout and it has percentage dimensions so it also supports some features of LinearLayout. In Simple words we can say that PercentRelativeLayout has features of both Layouts with reduced view complexity.

Need of PercentRelativeLayout In Android

In Android there are lot of Layout's that can be used at the time of development but at last we always end up with our main three layouts Linear Layout, Relative Layout and FrameLayout. In case if we need to create complex view then we use weight property of LinearLayout to distribute view across screens. But while using weight property we must have noticed that we have to add a default container view to encapsulate our child view . We can follow this approach but it adds an additional view hierarchy in our layout's which is of no use except holding weight sum value and child view. After the introduction of PercentRelativeLayout we can put percentage dimension and margins to our view that helps us to remove the problem of existing RelativeLayout.

Short Description About RelativeLayout:

Relative Layout is very flexible layout used in android for custom layout designing. It gives us the flexibility to position our component's based on the relative or sibling component's position. Just because it allows us to position the component anywhere we want so it is considered as most flexible layout. For the same reason Relative layout is the most used layout after the Linear Layout in Android. It allow its child view to position relative to each other or relative to the container or another container. You can read full tutorial about relative layout.

Important Note: In PercentRelativeLayout, it is not necessary to specify layout_width and layout_height attributes if we specify layout_widthPercent attribute. If in case we want the view

to be able to take up more space than what percentage value permits then we can add layout_width and layout_height attributes with wrap_content value. In that case if the size of content is larger than percentage size then it will be automatically resized using wrap_content rule. If we don't use layout_height and layout_width attribute then android studio shows us an error marker on element in the editor but it will be automatically ignored at run-time.

Basic PercentRelativeLayout XML Code:

```
-<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentRelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!-- Add Child View's Here... -->
</android.support.percent.PercentRelativeLayout>
```

Attributes of PercentRelativeLayout

Now let's we discuss some common attributes of a PercentRelativeLayout that helps us to configure it in our layout (xml). It supports all the attributes of RelativeLayout but here we are only discuss some additional attributes.

Very Important Note: Before you use below code make sure you have added Percent Support Library dependency in build.gradle file in depencies.

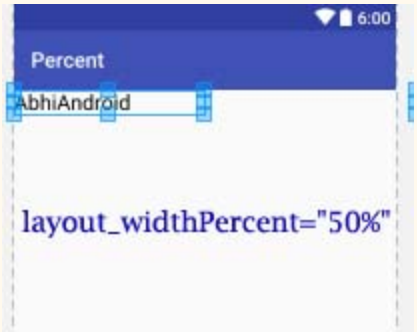
Gradle Scripts > build.gradle (Module:App) -> inside dependencies

```
compile 'com.android.support:percent:23.3.0' // Percent Support Library
```

- layout_widthPercent: This attribute is used to set the width of the view in percentage. Below we set 50% value for the widthPercent and wrap_content value for the height of the TextView.
-

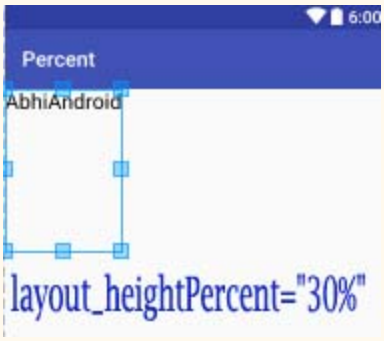
```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentRelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        app:layout_widthPercent="50%"
```

```
        android:layout_height="wrap_content"
        android:textColor="#000"
        android:textSize="20sp"
        android:text="AbhiAndroid" />
</android.support.percent.PercentRelativeLayout>
```



- Layout_heightPercent: This attribute is used to set the height of the view in percentage. Below we set 30% value for the heightPercent and wrap_content value for the width of the TextView.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentRelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:text="AbhiAndroid"
        android:textColor="#000"
        android:textSize="20sp"
        app:layout_heightPercent="30%" />
</android.support.percent.PercentRelativeLayout>
```



- **layout_marginPercent:** This attribute is used to set the margin for view in term of percentage. This attribute set the same margin from all the sides. Below we set wrap_content vlaue for height and width and 10% margin from all the sides of TextView.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentRelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="AbhiAndroid"
        android:textColor="#000"
        android:textSize="20sp"
        app:layout_marginPercent="10%" />
</android.support.percent.PercentRelativeLayout>
```

layout_marginLeftPercent: This attribute is used to set the percentage margin to the left side of the view. Below we set wrap_content value for height and width and 10% margin to the left side of the TextView.

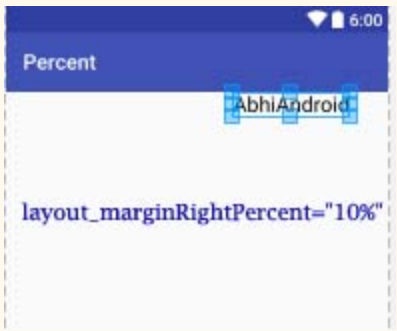
```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentRelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="AbhiAndroid"
        android:textColor="#000"
        android:textSize="20sp"
        app:layout_marginLeftPercent="10%" />
</android.support.percent.PercentRelativeLayout>
```

layout_marginTopPercent: This attribute is used to set the percentage margin to the top side of the view.Below we set wrap_content value for height and width and 10% margin to the top side of the TextView.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentRelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="AbhiAndroid"
        android:textColor="#000"
        android:textSize="20sp"
        app:layout_marginTopPercent="10%" />
</android.support.percent.PercentRelativeLayout>
```

- **layout_marginRightPercent:** This attribute is used to set the percentage margin to the right side of the view. Below we set wrap_content value for height and width and 10% margin to the right side of the TextView. In this example we set right alignment of TextView so that we can easily check the use of this attribute.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentRelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="AbhiAndroid"
        android:textColor="#000"
        android:textSize="20sp"
        android:layout_alignParentRight="true"
        app:layout_marginRightPercent="10%" />
</android.support.percent.PercentRelativeLayout>
```



- **layout_marginBottomPercent:** This attribute is used to set the percentage margin from bottom side of the view. Below we set wrap_content value for height and width and 10% margin from the bottom side of the TextView. In this example we set bottom alignment of TextView so that we can easily check the use of this attribute.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentRelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:text="AbhiAndroid"
        android:textColor="#000"
        android:textSize="20sp"
        app:layout_marginBottomPercent="10%" />
</android.support.percent.PercentRelativeLayout>
```

PercentRelativeLayout Example In Android Studio:

Below is the example of PercentRelativeLayout in which we create a Login Form with 2 fields user name and Password. In this example we take 2 EditText, 2 TextView and 1 Login Button. For these views we set the dimensions and margin in form of percentage. If a user click on the

login Button then a “Thank You” message with user name is displayed on the screen with the help of Toast.

Step 1: Create A New Project And Name It PercentRelativeLayoutExample.

Step 2: Open Gradle Scripts > build.gradle and add Percent Support Library dependency.

```
apply plugin: 'com.android.application'
android {
compileSdkVersion 24
buildToolsVersion "24.0.1"
defaultConfig {
applicationId "abhiandroid.percentrelativelayoutexample"
minSdkVersion 16
targetSdkVersion 24
versionCode 1
versionName "1.0"
}
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
}
}
}
dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
testCompile 'junit:junit:4.12'
```

```
compile 'com.android.support:appcompat-v7:24.1.1'
compile 'com.android.support:percent:23.3.0' // Percent Support Library
}
```

Step 3: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this Step we take 2 EditText‘s, 2 TextView’s and 1 Login Button and for these views we set the dimensions and margin in form of percentage. Note that in all the view’s i didn’t take layout_widht attribute and it still works fine. Android Studio shows a error line on element but it will be automatically ignored at runtime.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentRelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!--
    TextView's for labels and EditText for getting value from user
    -->
    <TextView
        android:id="@+id/txtUserName"
        android:layout_height="wrap_content"
        android:text="UserName :"
        android:textColor="#000"
        android:textSize="20sp"
        app:layout_marginLeftPercent="10%"
        app:layout_marginTopPercent="10%"
        app:layout_widthPercent="30%" />
    <TextView
        android:id="@+id/txtPassword"
        android:layout_height="wrap_content"
        android:layout_below="@+id/txtUserName"
        android:text="Password :"
        android:textColor="#000"
        android:textSize="20sp"
        app:layout_marginLeftPercent="10%"
        app:layout_marginTopPercent="4%"
        app:layout_widthPercent="30%" />
    <EditText
        android:id="@+id/userName"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@+id/txtUserName"
        android:hint="User Name"
        android:textColor="#000"
        android:textSize="20sp"
        app:layout_marginLeftPercent="3%"
        app:layout_marginTopPercent="8%"
        app:layout_widthPercent="50%" />
    <EditText
        android:id="@+id/password"
```

```
        android:layout_height="wrap_content"
        android:layout_toRightOf="@+id/txtPassword"
        android:hint="Password"
        android:inputType="textPassword"
        android:textColor="#000"
        android:textSize="20sp"
        app:layout_marginLeftPercent="3%"
        app:layout_marginTopPercent="16%"
        app:layout_widthPercent="50%" />
    <!-- Login Button -->
    <Button
        android:id="@+id/login"
        android:layout_height="wrap_content"
        android:layout_below="@+id/password"
        android:layout_centerHorizontal="true"
        android:background="#F0F"
        android:text="Login"
        android:textColor="#fff"
        android:textSize="20sp"
        app:layout_marginPercent="5%"
        app:layout_widthPercent="60%" />
</android.support.percent.PercentRelativeLayout>
```

Step 4 : Open src -> package -> MainActivity.java

In this step we open the MainActivity and add the code for initiates the views(EditText and Button). After that we implement setOnClickListener event on login Button so that whenever a user click on Button a thank you message if a user already fill both fields or a error message that enter your username and password is displayed on the screen with the help of Toast.

```
package abhiandroid.percentrelativelayoutexample;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
Button login;
EditText userName, password;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
// init the View's
userName = (EditText) findViewById(R.id.userName);
password = (EditText) findViewById(R.id.password);
login = (Button) findViewById(R.id.login);
// perform setOnClickListener Event on Login Button
login.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
// validate user name and password
```

```
if (userName.getText().toString().trim().length() > 0 &&
password.getText().toString().trim().length() > 0) {
// display a thanks you message with user name
Toast.makeText(getApplicationContext(), "Thank You " + userName.getText().toString(),
Toast.LENGTH_SHORT).show();
} else {
// display an error message that fill user name and password
Toast.makeText(getApplicationContext(), "Please Enter Your User Name And Password.!!",
Toast.LENGTH_SHORT).show();
}
}
});
}
}
```

Output:

Now run the App and you will login form designed using PercentRelativeLayout.

ToolBar

In Android Toolbar is similar to an ActionBar(now called as App Bars). Toolbar is a Viewgroup that can be placed at anywhere in the Layout. We can easily replace an ActionBar with Toolbar.

Toolbar was introduced in Material Design in API level 21 (Android 5.0 i.e Lollipop). Material Design brings lot of new features in Android that changed a lot the visual design patterns regarding the designing of modern Android applications.

An Action bar is traditionally a part of an Activity opaque window decor controlled by the framework but a Toolbar may be placed at any level of nesting within a view hierarchy. Toolbar provides more feature than ActionBar. A Toolbar may contain a combination of elements from start to end.

Important Note: Toolbar's are more flexible than ActionBar. We can easily modify its color, size and position. We can also add labels, logos, navigation icons and other views in it. In Material Design Android has updated the AppCompat support libraries so that we can use Toolbar's in our devices running API Level 7 and up. In AppCompat, Toolbar is implemented in the android.support.v7.widget.Toolbar class.

Define Design Support Library:

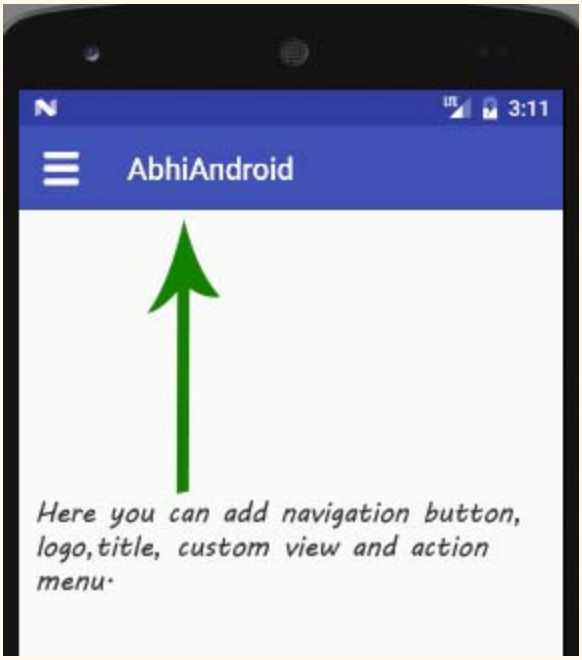
To use Toolbar you need to add design support library in build.gradle file.

Gradle Scripts > build.gradle (Module:App) -> inside dependencies

```
compile 'com.android.support:design:24.1.1' // design support Library
```

Elements of Toolbar In Android:

In Android Toolbar has more features than ActionBar and we can easily replace a ActionBar with Toolbar. In Toolbar from start to end it may contain a combination of Elements. Below we describe each and every element of Toolbar.



Navigation Button: It may be a Navigation menu toggle, up arrow, close, done, collapse or any other glyph of the app's choosing.

Brand Logo Image: It may extend to the height of the toolbar and can be arbitrarily wide.

Title and SubTitle: A title should be a signpost for the current position of Toolbar's navigation hierarchy and the content contained there. Subtitle represents any extended information about the current content. If an app uses a logo then it should strongly consider omitting a title and subtitle.

One or More Custom Views: An Application may add arbitrary child views to the Toolbar. If child view's Toolbar.LayoutParams indicates CENTER_HORIZONTAL Gravity then view will attempt to center within the available space remaining in the Toolbar after all other element's have been measured.

Action Menu: The menu of Actions will pin to the end of the Toolbar offering a few important, typical or frequent actions along with an optional overflow menu for additional actions. Action buttons are aligned vertically within the Toolbar's minimum height if we set.

Basic Toolbar XML code:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark">

</android.support.v7.widget.Toolbar>
```

Setting Toolbar as An ActionBar:

We can easily replace ActionBar with Toolbar by using setSupportActionBar() method. Here is the code of replacing ActionBar with Toolbar.

```
 Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
```

Attributes of Toolbar In Android:

Now let's we discuss some common attributes of a Toolbar that helps us to configure it in our layout (xml).

- **id:** This attribute is used to set the id for the Toolbar. Id is used to uniquely identify a Toolbar.

Below we set the id of the Toolbar.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/toolbar"
    android:background="@color/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark">
</android.support.v7.widget.Toolbar>
```

- **Logo:** This attribute is used to set as the drawable logo that appears at the starting side of the Toolbar means just after the navigation button. We can also do this programmatically by using setLogo() method.Below we set the logo for the Toolbar.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark"
    app:logo="@drawable/logo"><!-- logo for the Toolbar-->
</android.support.v7.widget.Toolbar>
```

- **logoDescription:** This attribute is used to set the content description string to describe the appearance of the associated logo image. We can also do this programmatically by using setLogoDescription() method.

Below we set the description for the displayed logo.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark"
    app:logo="@drawable/logo"
    app:logoDescription="LOGO"><!-- logo and logo description for the Toolbar-->
</android.support.v7.widget.Toolbar>
```

- **navigationIcon:** This attribute is used to set the Icon drawable for the navigation button that located at the start of the toolbar. We can also do this programmatically by using setNavigationIcon() method.Below we set the icon for the navigation button.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark"
    app:navigationIcon="@drawable/logo"><!-- navigation icon for the Toolbar-->
</android.support.v7.widget.Toolbar>
```

- **navigationContentDescription:** This attribute is used to set the text for the description of navigation button. We can also do this programmatically by using setNavigationContentDescription() method.

Below we set the content description for the displayed icon of navigation button.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark"
    app:navigationContentDescription="Navigation Description"
    app:navigationIcon="@drawable/logo">
    <!-- navigation icon and description for the Toolbar-->
</android.support.v7.widget.Toolbar>
```

- **title:** This attribute is used to set title for the Toolbar. We can also do this programmatically by using setTitle() method.

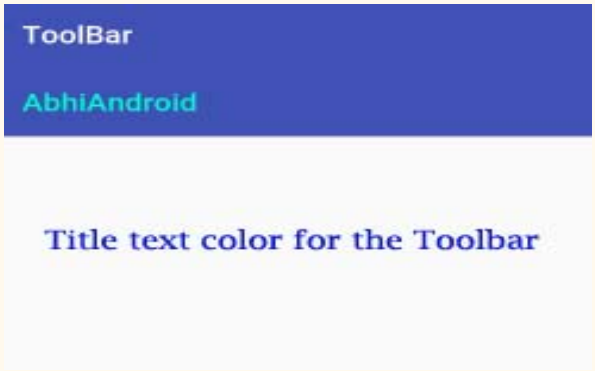
Below we set the title for the Toolbar.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark"
    app:title="AbhiAndroid">
    <!-- title for the Toolbar-->
</android.support.v7.widget.Toolbar>
```

- **titleTextColor:** This attribute is used to set the color for the title text. We can also do this programmatically by using setTitleTextColor() method.

Below we set the red color for the dispalyed title text.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark"
    app:titleTextColor="#F00"
    app:title="AbhiAndroid">
    <!-- title text color for the Toolbar-->
</android.support.v7.widget.Toolbar>
```



- **subtitle:** This attribute is used to set the sub title for the Toolbar. We can also do this programmatically by using setSubtitle() method.

Below we set the sub title for the Toolbar.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark"
    app:subtitle="AbhiAndroid">
    <!-- sub title for the Toolbar-->
</android.support.v7.widget.Toolbar>
```

- **subtitleTextColor:** This method is used to set the color for the sub title. We can also do this programmatically by using setSubtitleTextColor() method.

Below we set the red color for displayed subtitle text.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark"
    app:subtitle="AbhiAndroid"
    app:subtitleTextColor="#F00">
    <!-- sub title text  color for the Toolbar-->
</android.support.v7.widget.Toolbar>
```

Important Methods of Toolbar:

Let's we discuss some important methods of Toolbar that may be called in order to add Action icons manage the Toolbar.

Important Note: To understand below functions of Toolbar properly please do read 2nd example in this article.

- **setLogo(int resId):** This method is used to add a logo drawable from a resource id. Below we set a logo drawable in our Toolbar.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
toolbar.setLogo(R.drawable.logo); // setting a logo in toolbar
```

- **setLogo(Drawable drawable):** This method is also used to add a logo drawable in our Toolbar. In this method we set a drawable object. Below we set a drawable logo in our Toolbar.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
toolbar.setLogo(getResources().getDrawable(R.drawable.logo)); // setting a logo in toolbar
```

- **getLogo():** This method is used to get the logo icon from Toolbar. This method returns the Drawable logo which we set using setLogo() method.

Below we firstly set the drawable logo and then get the same from Toolbar.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
toolbar.setLogo(getResources().getDrawable(R.drawable.logo)); // setting a logo in toolbar
Drawable drawableLogo=toolbar.getLogo(); // get the drawable logo from Toolbar
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
```

```
Toolbar
```

- **setLogoDescription(CharSequence description):** This method is used to set a description for the drawable logo of Toolbar.

Below we set the description for Toolbar's logo.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
toolbar.setLogo(getResources().getDrawable(R.drawable.logo)); // setting a logo in toolbar
toolbar.setLogoDescription("LOGO"); // set description for the logo
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
```

setLogoDescription(int resId): This method is also used to set the description for drawable logo of the Toolbar. This method set the description from string file.

Below we set the description for Toolbar's logo.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
toolbar.setLogo(getResources().getDrawable(R.drawable.logo)); // setting a logo in toolbar
toolbar.setLogoDescription(getResources().getString(R.string.descrition)); // set
description for the logo
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
```

- **getLogoDescription():** This method is used to get the description of toolbar's logo. This method returns description in CharSequence object.

Below we firstly set the description for logo of Toolbar and then get the same from Toolbar.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
toolbar.setLogo(getResources().getDrawable(R.drawable.logo)); // setting a logo in Toolbar
toolbar.setLogoDescription("LOGO"); // set description for the logo
CharSequence logoDescription=toolbar.getLogoDescription(); // get the logo description from
```

```
Toolbar
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
```

- **setNavigationIcon(int resId):** This method is used to set the icon from resource id to the Toolbar's navigation button.

Below we set a navigation icon in the Toolbar from resource id.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
toolbar.setNavigationIcon(R.mipmap.ic_launcher); // set icon for navigation button
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
```

- **setNavigationIcon(Drawable icon):** This method is also used to set the icon for Navigation Buttton. In this method we set the icon from drawable object.

Below we set drawable icon for Navigation Button.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
toolbar.setNavigationIcon(getResources().getDrawable(R.drawable.logo)); // setting a
navigation icon in Toolbar
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
```

- **getNavigationIcon():** This method is used to get the navigation icon from Toolbar which we set using setNavigationIcon() method. This method returns a drawable object.

Below we firstly set the navigation icon and then get the same from Toolbar.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
toolbar.setNavigationIcon(getResources().getDrawable(R.drawable.logo)); // setting a
navigation icon in Toolbar
Drawable navigationIcon = toolbar.getNavigationIcon(); // get navigation icon from Toolbar
```

```
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
```

- **setTitle(int resId):** This method is used to set the Title for the Toolbar. In this method we set the title form string file.

Below we set the title for the Toolbar.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar4
toolbar.setTitle(getResources().getString(R.string.title)); // setting a title for this
Toolbar
```

- **setTitle(CharSequence title):** This method is also used to set the title for the Toolbar.

Below we set the title for the Toolbar.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar4
toolbar.setTitle("AbhiAndroid"); // setting a title for this Toolbar
```

- **setSubtitle(int resId):** This method is used to set the sub Title for the Toolbar. In this method we set the sub title form string file.

Below we set the sub title for the Toolbar.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar4
toolbar.setSubtitle(getResources().getString(R.string.subTitle)); // setting a sub title for
this Toolbar
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
```

- **setSubtitle(CharSequence subtitle):** This method is also used to set the sub title for the Toolbar.

Below we set the sub title for the Toolbar.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar4
toolbar.setSubtitle("AbhiAndroid"); // setting a sub title for this Toolbar
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
```

- **getTitle():** This method is used to get the title of the Toolbar. This method returns CharSequence object for the title displayed on Toolbar.

Below we firstly set the title and then get the same from Toolbar.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar4
toolbar.setTitle("AbhiAndroid"); // setting a title for this Toolbar
CharSequence title=toolbar.getTitle(); // get the displayed title from Toolbar.
```

- **getSubtitle():** This method is used to get the sub title of the Toolbar. This method returns CharSequence object for the sub title displayed on Toolbar.

Below we firstly set the sub title and then get the same from Toolbar.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar4
toolbar.setSubtitle("AbhiAndroid"); // setting a sub title for this Toolbar
CharSequence title=toolbar.getSubtitle(); // get the displayed sub title from Toolbar.
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
```

- **setNavigationContentDescription(CharSequence description):** This method is used to set the description for the Navigation Button.

Below we set the description for Navigation Button.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
toolbar.setNavigationIcon(R.drawable.logo); // set icon for navigation button
toolbar.setNavigationContentDescription("Navigation"); // setting description for navigation
button
```

- **setNavigationContentDescription(int resId):** This method is used to set the description for the Navigation Button. This method sets the description from string file.

Below we set the description od the navigation button.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
toolbar.setNavigationIcon(R.drawable.logo); // set icon for navigation button
toolbar.setNavigationContentDescription(getResources().getString(R.string.description)); //
setting description for navigation button.
```

- **getNavigationContentDescription():** This method is used to get the currently configured content description for the navigation button view. This method returns the description in CharSequence object which we set using setNavigationContentDescription() method.

Below we firstly set the description of navigation button and then get the same from Toolbar.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
toolbar.setNavigationIcon(R.drawable.logo); // set icon for navigation button
toolbar.setNavigationContentDescription("Navigation"); // setting description for navigation
button
CharSequence description=toolbar.getNavigationContentDescription(); // get navigation button
```

```
description
```

- **setTitleTextColor(int color):** This method is used to set the text color for the displayed title.

Below we set the red color for the displayed text of title.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
toolbar.setTitle("AbhiAndroid"); // setting a title for this Toolbar
toolbar.setTitleTextColor(Color.RED); // set title text color for Toolbar.
```

- **setSubtitleTextColor(int color):** This method is used to set the text color for the displayed sub title.

Below we set the red color for the displayed sub text of title.

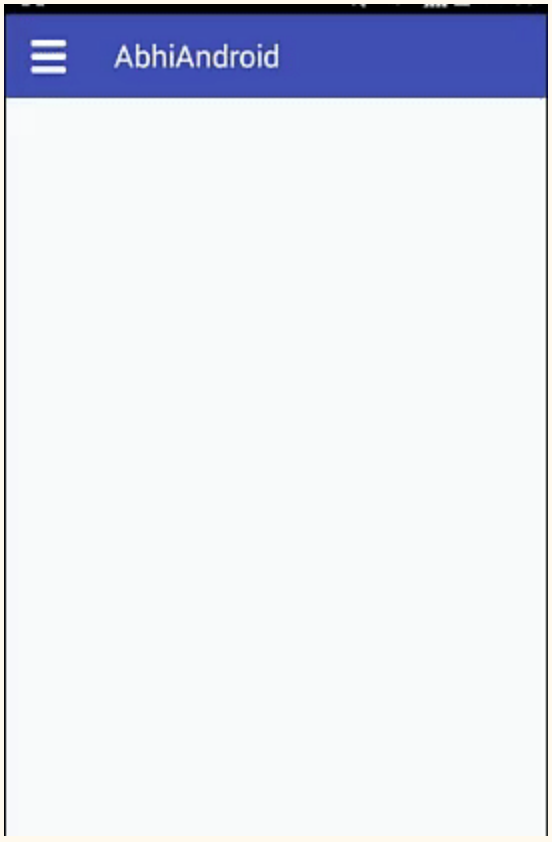
```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
toolbar.setSubtitle("AbhiAndroid"); // setting a sub title for this Toolbar
toolbar.setSubtitleTextColor (Color.RED); // set sub title text color for Toolbar
```

- **setNavigationOnClickListener(View.OnClickListener listener):** It sets a listener to respond to navigation events. This listener is used to handle the click event on navigation button. Below we implement setNavigationOnClickListener event on Toolbar that handles on click event of navigation button.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
getSupportActionBar().setDisplayShowTitleEnabled(false); // hide the current title from the
Toolbar
// implement setNavigationOnClickListener event
toolbar.setNavigationOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
// add code here that execute on click of navigation button
}
});
```

Toolbar Example 1 In Android Studio:

Below is the first example of Toolbar in which we create a Toolbar and replace it with ActionBar. In this examle we add action icons in Toobar and on click of navigation Button of Toolbar we open a Navigation Drawer. In our main layout we use Drawer Layout and Navigation View. Drawer Layout is the root layout in which we include a Toolbar and also define a FrameLayout and a Navigation View. In Navigation View we set the items from menu file and FrameLayout is used to replace the Fragments on the click of menu items. Whenever a user click on menu item, Fragment is replaced according to menu item's id and a toast message with menu item title is displayed on the screen. We also create three Fragments that should be replaced on click of menu items.



Step 1: Create a new project and name it ToolbarExample.

Step 2: Open Gradle Scripts > build.gradle and add Design support library dependency. If you are on the latest version of Android Studio you don't need to add a compiled dependency for Appcombat v7 21 library if not then please make sure you add the line below in your gradel build dependencies.

apply plugin: ‘com.android.application’

```
android {
compileSdkVersion 24
buildToolsVersion "24.0.1"
defaultConfig {
applicationId "abhiandroid.toolbarexample"
minSdkVersion 16
targetSdkVersion 24
versionCode 1
versionName "1.0"
}
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
}
}
}
dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
testCompile 'junit:junit:4.12'
compile 'com.android.support:appcompat-v7:24.1.1'
compile 'com.android.support:design:24.1.1' // design support Library
}
```

Step 3: ActionBars are replaced with Toolbar so we can still use ActionBar but in our example we are going to make a Toolbar so go to your style.xml file and change the theme to “Theme.AppCompat.Light.NoActionBar” This theme helps us get rid of the ActionBar so we can make a Toolbar.

```
<!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

Step 4: Now let’s talk about the color scheme for our application. Open your color.xml file from values folder. In this XML file colorPrimary is the primary color for the app and

colorPrimaryDark color is the color of the status bar. We can easily change our colors from this file

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

Step 5: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we define a DrawerLayout that is the parent layout in which we include a toolbar.xml file and also define a FrameLayout and a Navigation View. In Navigation View we set the items from menu file named “nav_items” and FrameLayout is used to replace the Fragments on the click of menu items.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context=".MainActivity">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <!-- include your toolbar layout-->
        <include layout="@layout/toolbar" />
        <!-- Let's add fragment -->
        <FrameLayout
            android:id="@+id/frame"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
    </LinearLayout>
    <!--
         Navigation view to show the menu items
    -->
    <android.support.design.widget.NavigationView
        android:id="@+id/navigation"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:menu="@menu/nav_items" />
</android.support.v4.widget.DrawerLayout>
```

Step 6: Open res -> menu -> nav_items.xml and add following code. Here we define the menu items.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <!--
                Add menu items here
                Menu items with icon and Title
        -->
        <item
            android:id="@+id/first"
            android:icon="@mipmap/ic_launcher"
            android:title="First Menu" />
        <item
            android:id="@+id/second"
            android:icon="@mipmap/ic_launcher"
            android:title="Second Menu" />
        <item
            android:id="@+id/third"
            android:icon="@mipmap/ic_launcher"
            android:title="Third Menu" />
    </group>
</menu>
```

Step 7: Now create a xml layouts by right clicking on res/layout -> New -> Layout Resource File and name it toolbar.xml

In this file we set the background color, navigation button and tilte for the Toolbar.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark"
    app:navigationIcon="@drawable/menu_icon"
    app:title="AbhiAndroid"
    app:titleTextColor="#FFF">
    <!-- navigation button and title for the Toolbar-->
</android.support.v7.widget.Toolbar>
```

Step 8: Open src -> package -> MainActivity.java

In this step we open the MainActivity and add the code for initiates the views(DrawerLayout, Toolbar,NavigationView and other views). In this we replace the ActionBar with our ToolBar by using setSupportActionBar() method and then implment setNavigationOnClickListener event and add the code to open the drawer on click of navigation button. After that we implement setNavigationItemSelectedListener event on NavigationView so that we can replace the Fragments according to menu item's id and a toast message with menu item's title is displayed on the screen. In this I have added comments in code to help you to understand the code easily so make you read the comments.

```
package abhiandroid.toolbarexample;
import android.os.Bundle;
import android.support.design.widget.NavigationView;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentTransaction;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Gravity;
import android.view.MenuItem;
import android.view.View;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
DrawerLayout dLayout;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar

// implement setNavigationOnClickListener event
toolbar.setNavigationOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
dLayout.openDrawer(Gravity.LEFT);
}
});
setNavigationDrawer(); // call method
}
private void setNavigationDrawer() {
dLayout = (DrawerLayout) findViewById(R.id.drawer_layout); // initiate a DrawerLayout
NavigationView navView = (NavigationView) findViewById(R.id.navigation); // initiate a
Navigation View
// implement setNavigationItemSelectedListener event on NavigationView
navView.setNavigationItemSelectedListener(new
NavigationView.OnNavigationItemSelectedListener() {
```

```
@Override
public boolean onNavigationItemSelected(MenuItem menuItem) {
Fragment frag = null; // create a Fragment Object
int itemId = menuItem.getItemId(); // get selected menu item's id
// check selected menu item's id and replace a Fragment Accordingly
if (itemId == R.id.first) {
frag = new FirstFragment();
} else if (itemId == R.id.second) {
frag = new SecondFragment();
} else if (itemId == R.id.third) {
frag = new ThirdFragment();
}
// display a toast message with menu item's title
Toast.makeText(getApplicationContext(), menuItem.getTitle(), Toast.LENGTH_SHORT).show();
if (frag != null) {
FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
transaction.replace(R.id.frame, frag); // replace a Fragment with Frame Layout
transaction.commit(); // commit the changes
dLayout.closeDrawers(); // close the all open Drawer Views
return true;
}
return false;
}
});
}
}
```

Step 9: Now we need 3 fragments and 3 xml layouts. So create three fragments by right click on your package folder and create classes and name them as FirstFragment, SecondFragment and ThirdFragment and add the following code respectively.

FirstFragment.class

```
package abhiandroid.toolbarexample;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class FirstFragment extends Fragment {
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate(R.layout.fragment_first, container, false);
}
}
```

SecondFragment.class

```
package abhiandroid.toolbarexample;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class SecondFragment extends Fragment {
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate(R.layout.fragment_second, container, false);
}
}
```

ThirdFrament.class

```
package abhiandroid.toolbarexample;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class ThirdFragment extends Fragment {
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate(R.layout.fragment_third, container, false);
}
}
```

Step 10: Now create 3 xml layouts by right clicking on res/layout -> New -> Layout Resource File and name them as fragment_first, fragment_ second and fragment_third and add the following code in respective files.

Here we will design the basic simple UI by using TextView in all xml's.

Fragment_first.xml

```
<FrameLayout 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"
    tools:context="abhiandroid.toolbarexample.FirstFragment">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:text="First Fragment"
        android:textSize="25sp" />
</FrameLayout>
```

Fragment_second.xml

```
<FrameLayout 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"
    tools:context="abhiandroid.toolbarexample.SecondFragment">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:text="Second Fragment"
        android:textSize="25sp" />
</FrameLayout>
```

Fragment_third.xml

```
<FrameLayout 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"
    tools:context="abhiandroid.toolbarexample.ThirdFragment">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:text="Third Fragment"
        android:textSize="25sp" />
</FrameLayout>
```

Output:

Now run the App and you will see Menu icon and “AbhiAndroid” text written at the top which is created using Toolbar. Now click on menu and three items inside it will be shown. Now click on any of them and corresponding to it layout will open up.

Toolbar Example 2 In Android Studio:

Below is the simple example of Toolbar in which we create a Toolbar and replace it with our ActionBar. In this example we set Logo, title and menu for Toolbar. In our main layout we create a Toolbar and in Activity get the reference of Toolbar and set the title and logo on it. In this we use Activity’s overrided method onCreateOptionsMenu to set the menu items from menu file and onOptionsItemSelected to set click listeners on menu item’s. On click of menu item the title of menu is displayed on the screen with the help of Toast.

Step 1: Create a new project and name it SimpleToolbarExample.

Step 2: Open Gradle Scripts > If you are on the latest version of Android Studio you don’t need to add a compiled dependency for Appcombat v7 21 library if not then please make sure you add the line below in your gradel build dependencies.

```
apply plugin: 'com.android.application'
android {
compileSdkVersion 24
buildToolsVersion "24.0.1"
defaultConfig {
applicationId "abhiandroid.com.simpletoolbarexample"
minSdkVersion 16
targetSdkVersion 24
versionCode 1
versionName "1.0"
}
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
}
}
}
dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
testCompile 'junit:junit:4.12'
compile 'com.android.support:appcompat-v7:24.1.1'
}
```

Step 3: ActionBars are replaced with Toolbar so we can still use ActionBar but in our example we are going to make a Toolbar so go to your style.xml file and change the theme to “Theme.AppCompat.Light.NoActionBar” This theme helps us get rid of the ActionBar so we can make a Toolbar.

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

Step 4: Now let’s talk about the color scheme for our application. Open your color.xml file from values folder. In this XML file colorPrimary is the primary color for the app and colorPrimaryDark color is the color of the status bar. We can easily change our colors from this file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

Step 5: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we define a Toolbar in our main XML file and set it’s background color and theme.

```
<?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"
    tools:context="abhiandroid.com.simpletoolbarexample.MainActivity">
    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/colorPrimary"
        android:theme="@style/ThemeOverlay.AppCompat.Dark" />
</RelativeLayout>
```

Step 6: Open res -> menu -> menu_main.xml and add following code:

In this step we create 3 menu items that will be displayed on Toolbar.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity">
    <!--
        Menu item's that displayed on Toolbar.
    -->
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never" />
    <item
        android:id="@+id/action_search"
        android:icon="@drawable/ic_search"
        android:orderInCategory="200"
        android:title="Search"
        app:showAsAction="ifRoom" />
    <item
        android:id="@+id/action_user"
        android:icon="@drawable/user"
        android:orderInCategory="300"
        android:title="User"
        app:showAsAction="ifRoom" />
</menu>
```

Step 7: Open src -> package -> MainActivity.java

In this step we open the MainActivity and add the code for initiates the Toolbar.. In this we replace the ActionBar with our ToolBar by using setSupportActionBar() method and also set the title and logo for the Toolbar. After that we implement Activity's overrided method onCreateOptionsMenu to set the menu items from menu file and onOptionsItemSelected to set click listeners on menu item's. On click of menu item the title of menu is displayed on the screen with the help of Toast. In this I have added comments in code to help you to understand the code easily so make sure you read the comments.

```
package abhiandroid.com.simpletoolbarexample;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference of Toolbar
toolbar.setTitle("AbhiAndroid"); // set Title for Toolbar
toolbar.setLogo(R.drawable.android); // set logo for Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
}
// Activity's overrided method used to set the menu file
@Override
public boolean onCreateOptionsMenu(Menu menu) {
// Inflate the menu; this adds items to the action bar if it is present.
getMenuInflater().inflate(R.menu.menu_main, menu);
return true;
}
// Activity's overrided method used to perform click events on menu items
@Override
public boolean onOptionsItemSelected(MenuItem item) {
// Handle action bar item clicks here. The action bar will
// automatically handle clicks on the Home/Up button, so long
// as you specify a parent activity in AndroidManifest.xml.
int id = item.getItemId();
//noinspection SimplifiableIfStatement
// Display menu item's title by using a Toast.
if (id == R.id.action_settings) {
Toast.makeText(getApplicationContext(), "Setting Menu", Toast.LENGTH_SHORT).show();
return true;
} else if (id == R.id.action_search) {
Toast.makeText(getApplicationContext(), "Search Menu", Toast.LENGTH_SHORT).show();
return true;
} else if (id == R.id.action_user) {
Toast.makeText(getApplicationContext(), "User Menu", Toast.LENGTH_SHORT).show();
return true;
}
return super.onOptionsItemSelected(item);
}
}
```

Output:

Now run the App and you will see Toolbar at the top. Click on it and message created using Toast will be displayed.

PercentFrameLayout

In Android PercentFrameLayout is a subclass of FrameLayout that supports percentage based margin and dimensions for Views(Button, TextView or any other view).

PercentFrameLayout class extends from FrameLayout class so it supports all the features, attributes of FrameLayout and it has percentage dimensions so it also supports some features of Linear Layout. In Simple words we can say that PercentFrameLayout has features of both Layouts with reduced view complexity.

Percent Support Library: Percent Support Library provides feature to set the dimensions and margins in the term of Percentage. It has two pre built Layouts- the PercentFrameLayout and PercentRelativeLayout. This library is pretty easy to use because it has same Relative Layout and Frame Layout that we are familiar with, just with some additional new features. In this article we only discuss PercentFrameLayout.

Important Note – To use PercentFrameLayout first include Percent Support Library dependency file in build.grade(Module: app).

Gradle Scripts > build.gradle (Module:App) -> inside dependencies

```
dependencies {
compile 'com.android.support:percent:23.3.0'
}
```

Need of PercentFrameLayout:

In Android there are lot of Layout's that can be used at the time of development but at last we ends up with our main three layouts LinearLayout, RelativeLayout and FrameLayout. In many situations when we need to create complex view then we use weight property of LinearLayout to distribute view across screens. But while using weight property we must have noticed that we have to add a default container view to encapsulate our child view. We can follow this approach but it adds an additional view hierarchy in our layout's which is of no use except holding weight sum value and child view. After the introduction of PercentFrameLayout we can put percentage dimension and margins to our view that helps us to remove the problem of existing FrameLayout.

Short Description About FrameLayout In Android:

In Android Frame Layout is one of the simplest layout used to organize view controls. It is designed to block an area on the screen. Frame Layout should be used to hold child view, because it can be difficult to display single views at a specific area on the screen without overlapping each other. You can read full FrameLayout tutorial here.

Important Note:

In PercentFrameLayout, it is not necessary to specify layout_width and layout_height attributes if we specify layout_widthPercent attribute. If in case we want the view to be able to take up more space than what percentage value permits then we can add layout_width and layout_height attributes with wrap_content value. In that case if the size of content is larger than percentage size then it will be automatically resized using wrap_content rule. If we don't use layout_height and layout_width attribute then Android Studio shows us an error marker on element in the editor but it will be automatically ignored at runtime.

Basic PercentFrameLayout XML Code:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentFrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!-- Add Child View's Here... -->
</android.support.percent.PercentFrameLayout>
```

Attributes of PercentFrameLayout:

Now let's we discuss some common attributes of a PercentFrameLayout that helps us to configure it in our layout (xml). It supports all the attributes of FrameLayout but here we are only discuss some additional attributes.

Important Note – Before using below code please include Percent Support Library dependency file in build.grade(Module: app).

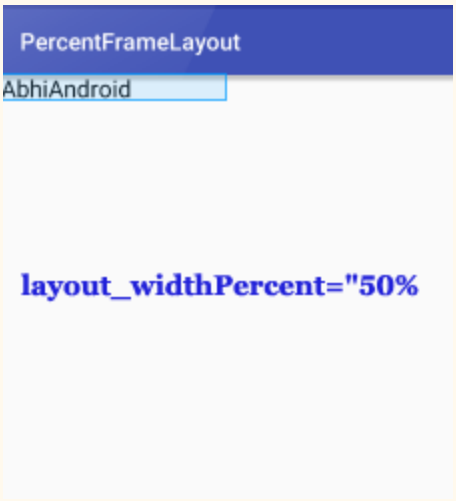
Gradle Scripts > build.gradle (Module:App) -> inside dependencies

```
dependencies {
compile 'com.android.support:percent:23.3.0'
}
```

1. layout_widthPercent: This attribute is used to set the width of the view in percentage.

Below we set 50% value for the widthPercent and wrap_content value for the height of the TextView.

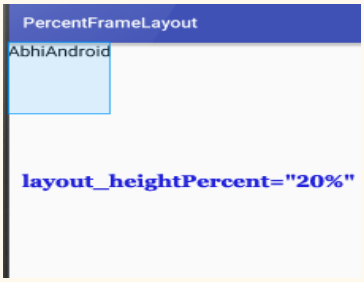
```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentFrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        app:layout_widthPercent="50%"
        android:layout_height="wrap_content"
        android:textColor="#000"
        android:textSize="20sp"
        android:text="AbhiAndroid" />
</android.support.percent.PercentFrameLayout>
```



2. layout_heightPercent: This attribute is used to set the height of the view in percentage.

Below we set 20% value for the heightPercent and wrap_content value for the width of the TextView.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentFrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:text="AbhiAndroid"
        android:textColor="#000"
        android:textSize="20sp"
        app:layout_heightPercent="20%" />
</android.support.percent.PercentFrameLayout>
```



3. layout_marginPercent: This attribute is used to set the margin for view in term of percentage. This attribute set the same margin from all the sides.

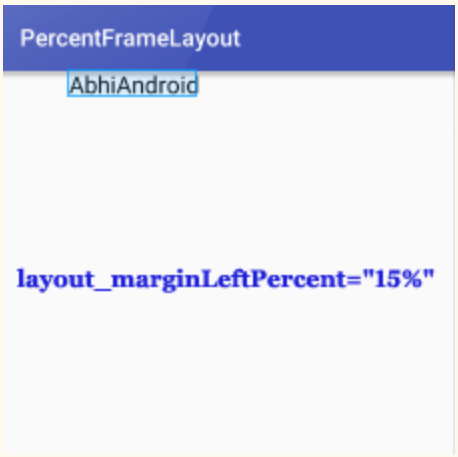
Below we set wrap_content vlaue for height and width and 20% margin from all the sides of TextView.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentFrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="AbhiAndroid"
        android:textColor="#000"
        android:textSize="20sp"
        app:layout_marginPercent="20%" />
</android.support.percent.PercentFrameLayout>
```

4. layout_marginLeftPercent: This attribute is used to set the percentage margin to the left side of the view.

Below we set wrap_content value for height and width and 15% margin to the left side of the TextView.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentFrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="AbhiAndroid"
        android:textColor="#000"
        android:textSize="20sp"
        app:layout_marginLeftPercent="15%" />
</android.support.percent.PercentFrameLayout>
```

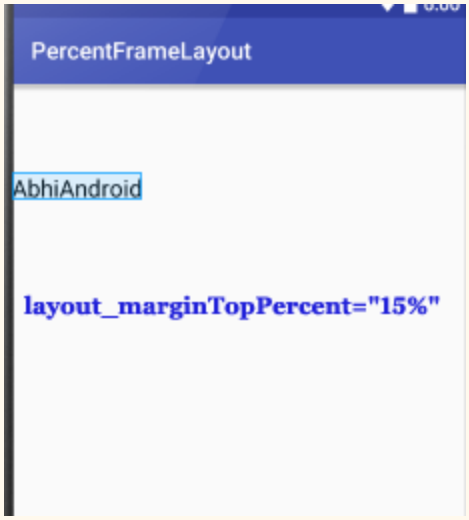


5. layout_marginTopPercent: This attribute is used to set the percentage margin to the top side of the view.

Below we set wrap_content value for height and width and 15% margin to the top side of the TextView.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentFrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

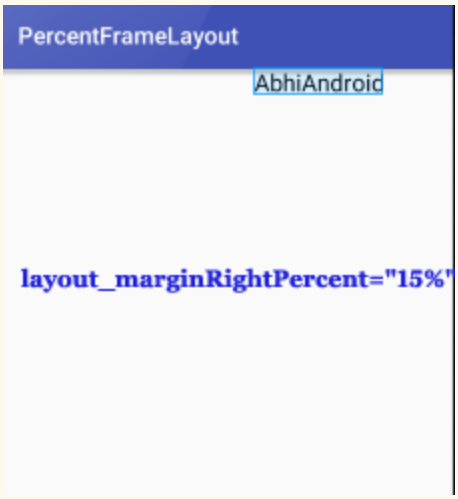
```
        android:text="AbhiAndroid"
        android:textColor="#000"
        android:textSize="20sp"
        app:layout_marginTopPercent="15%" />
</android.support.percent.PercentFrameLayout>
```



6. layout_marginRightPercent: This attribute is used to set the percentage margin to the right side of the view.

Below we set wrap_content value for height and width and 15% margin to the right side of the TextView. In this example we set right gravity of TextView so that we can easily check the use of this attribute.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentFrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="AbhiAndroid"
        android:textColor="#000"
        android:textSize="20sp"
        app:layout_marginRightPercent="15%" />
</android.support.percent.PercentFrameLayout>
```



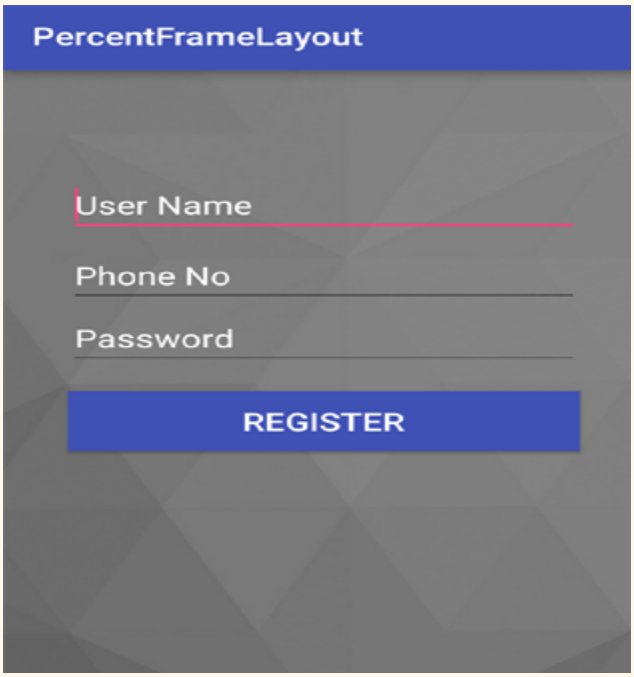
7. layout_marginBottomPercent: This attribute is used to set the percentage margin from bottom side of the view.

Below we set wrap_content value for height and width and 15% margin from the bottom side of the TextView. In this example we set bottom gravity of TextView so that we can easily check the use of this attribute.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentFrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        android:text="AbhiAndroid"
        android:textColor="#000"
        android:textSize="20sp"
        app:layout_marginBottomPercent="10%" />
</android.support.percent.PercentFrameLayout>
```

PercentFrameLayout Example In Android Studio:

Below is the example of PercentFrameLayout in which we create a Registration Form with 3 fields User Name, Phone No and Password. In this example we take 3 EditText and 1 Register Button. And for these views we set the dimensions and margin in form of percentage. If a user click on the Register Button then a “Thank You” message with user name is displayed on the screen with the help of Toast.



Step 1: Create A New Project And Name It PercentFrameLayout

Step 2: Open Gradle Scripts > build.gradle and add Percent Support Library dependency.

```
apply plugin: 'com.android.application'
android {
compileSdkVersion 24
buildToolsVersion "24.0.1"
defaultConfig {
applicationId "abhiandroid.percentframelayout"
minSdkVersion 16
targetSdkVersion 24
versionCode 1
versionName "1.0"
}
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
}
}
}
dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
testCompile 'junit:junit:4.12'
compile 'com.android.support:appcompat-v7:24.1.1'
compile 'com.android.support:percent:23.3.0' // Percent Support Library
}
```

Step 3: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this Step we take 3 EditText's and 1 Register Button and for these views we set the dimensions and margin in form of percentage. Note that in all the view's i didn't take layout_width attribute and it still works fine. Android Studio shows a error line on element but it will be automatically ignored at runtime.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.percent.PercentFrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/background">
<!--
    3 EditText's for user Name, Phone No and Password
-->
    <EditText
        android:id="@+id/userName"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:hint="User Name"
        android:imeOptions="actionNext"
        android:singleLine="true"
        android:textColor="#fff"
        android:textColorHint="#fff"
        android:textSize="20sp"
        app:layout_marginTopPercent="15%"
        app:layout_widthPercent="80%" />
    <EditText
        android:id="@+id/phoneNo"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:hint="Phone No"
        android:imeOptions="actionNext"
        android:inputType="number"
        android:singleLine="true"
        android:textColor="#fff"
        android:textColorHint="#fff"
        android:textSize="20sp"
        app:layout_marginTopPercent="25%"
        app:layout_widthPercent="80%" />
    <EditText
        android:id="@+id/password"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:hint="Password"
        android:imeOptions="actionDone"
        android:singleLine="true"
        android:textColor="#fff"
        android:textColorHint="#fff"
        android:textSize="20sp"
        app:layout_marginTopPercent="34%"
```

```
        app:layout_widthPercent="80%" />
    <Button
        android:id="@+id/register"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:background="@color/colorPrimary"
        android:text="Register"
        android:textColor="#fff"
        android:textSize="20sp"
        app:layout_widthPercent="80%" />
</android.support.percent.PercentFrameLayout>
```

Step 4 : Open src -> package -> MainActivity.java

In this step we open the MainActivity and add the code for initiates the views(EditText and Button). After that we implement setOnClickListener event on Register Button so that whenever a user click on Button a thank you message appear if a user fill all the fields or an error message appear if fields are empty telling user 'Please Fill All The Fields' is displayed on the screen by using a Toast.

```
package abhiandroid.percentframelayout;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
Button register;
EditText userName, phoneNo, password;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
// init the View's
userName = (EditText) findViewById(R.id.userName);
phoneNo = (EditText) findViewById(R.id.phoneNo);
password = (EditText) findViewById(R.id.password);
register = (Button) findViewById(R.id.register);
// perform setOnClickListener Event on Register Button
register.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
if (userName.getText().toString().trim().length() > 0 &&
phoneNo.getText().toString().trim().length() > 0 &&
password.getText().toString().trim().length() > 0) {
// display a thanks you message with user name
Toast.makeText(getApplicationContext(), "Thank You " + userName.getText().toString(),
Toast.LENGTH_SHORT).show();
} else {
```

```
// display an error message that fill user name and password
Toast.makeText(getApplicationContext(), "Please Fill All The Fields.!!",
Toast.LENGTH_SHORT).show();
}
}
});
}
}
```

Output:

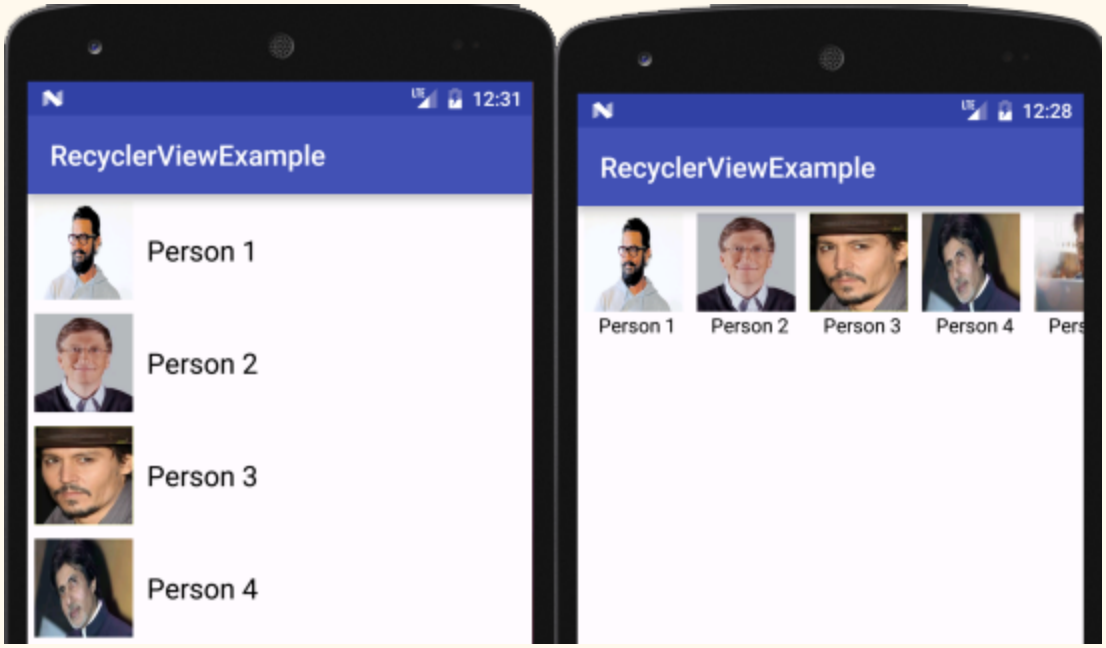
Now run the App and you will see login form which is designed using PercentFrameLayout.

RecyclerView As ListView

In Android, RecyclerView is an advanced and flexible version of ListView and GridView. It is a container used for displaying large amount of data sets that can be scrolled very efficiently by maintaining a limited number of views.

RecyclerView was introduced in Material Design in API level 21 (Android 5.0 i.e Lollipop). Material Design brings lot of new features in Android that changed a lot the visual design patterns regarding the designing of modern Android applications.

This new widget is a big step for displaying data because the ListView is one of the most commonly used UI widget. In RecyclerView android provides a lots of new features which are not present in existing ListView or GridView.



Basic RecyclerView XML code:

```
<?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"
    tools:context="abhiandroid.com.recyclerviewexample.MainActivity">
    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

Gradle Dependency to use in RecyclerView:

The RecyclerView widget is a part of separate library valid for API 7 level or higher. You will need to add the following dependency in your Gradle build file to use recyclerview.

Gradle Scripts > build.gradle

```
dependencies {
 ...
 compile "com.android.support:recyclerview-v7:23.0.1"
 }
```

RecyclerView As ListView In Android:

In this article we will discuss how to use a RecyclerView as ListView. For that we need to understand LayoutManager component of RecyclerView.

Layout Manager is basically a very new concept introduced in RecyclerView for defining the type of Layout which RecyclerView should use.

It contains the references for all the views that are filled by the data of the entry. We can create a Custom Layout Manager by extending RecyclerView.

LayoutManager Class of RecyclerView provides three types of in-built Layout Managers.

- **LinearLayoutManager:** It is used for displaying vertical or horizontal list.
- **GridLayoutManager:** It is used for displaying the items in the form of Grids.
- **StaggeredGridLayoutManager:** It is used to show the items in staggered Grid (uneven size image in Grid).

As this article only focus on RecyclerView as ListView so in this article article I am going to discuss only about LinearLayoutManager because it is used to display the data in the form of List.

LinearLayoutManager : LinearLayoutManager is used for displaying the data items in a horizontal or vertical scrolling List.

If we need a list(vertical or horizontal) then we need to use LinearLayoutManager setting with require orientation. In Simple words we can say that we use the LinearLayoutManager for displaying RecyclerView as a ListView.

Public constructor for LinearLayoutManager: Below we define the public constructor for LinearLayoutManager that should be used for defining orientation(vertical or horizontal) of RecyclerView.

1) LinearLayoutManager(Context context): It is used to create a vertical LinearLayoutManager. In this we need to set only one parameter i.e used to set the context of the current Activity.

Example:

In below code snippet we show how to use this constructor in Android.

With Default Vertical Orientation:

```
// get the reference of RecyclerView
 RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
 // set a LinearLayoutManager with default orientation
 LinearLayoutManager linearLayoutManager = new LinearLayoutManager(getApplicationContext());
 recyclerView.setLayoutManager(linearLayoutManager); // set LayoutManager to RecyclerView
```

With Horizontal Orientation:

```
// get the reference of RecyclerView
 RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
 // set a LinearLayoutManager
 LinearLayoutManager linearLayoutManager = new LinearLayoutManager(getApplicationContext());
```

```
 linearLayoutManager.setOrientation(LinearLayoutManager.HORIZONTAL); // set Horizontal
Orientation
 recyclerView.setLayoutManager(linearLayoutManager); // set LayoutManager to RecyclerView
```

2) LinearLayoutManager(Context context, int orientation, boolean reverseLayout): In this first parameter is used to set the current context, second is used to set the Layout Orientation should be vertical or horizontal. By using this constructor we can easily create a horizontal or vertical List. Third parameter is a boolean value which set to true layouts from end to start, it means items are arranged from end to start.

Example: .

In below code snippet we show how to use this constructor in Android.

```
// get the reference of RecyclerView
 RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
 // set a LinearLayoutManager with default horizontal orientation and false value for
reverseLayout to show the items from start to end
 LinearLayoutManager linearLayoutManager = new
LinearLayoutManager(getApplicationContext(),LinearLayoutManager.HORIZONTAL,false);
 recyclerView.setLayoutManager(linearLayoutManager); // set LayoutManager to RecyclerView
```

Comparison between RecyclerView and ListView:

There are a lots of new features in RecyclerView that are not present in existing ListView. The RecyclerView is more flexible, powerful and a major enhancement over ListView. Here I will try to give you a detailed insight into it.

Below we discuss some important features of RecyclerView that should clear the reason why RecyclerView is better than ListView.

1. Custom Item Layouts: ListView can only layout the items in Vertical Arrangement and that arrangement cannot be customized according to our requirements. Suppose we need to create a horizontal List then that thing is not feasible with default ListView. But with introduction of Recyclerview we can easily create a horizontal or Vertical List. By using Layout Manager component of RecyclerView we can easily define the orientation of Items.

2. Use Of ViewHolder Pattern: ListView Adapters do not require the use of ViewHolder but RecyclerView require the use of ViewHolder that is used to store the reference of View's.

In ListView it is recommended to use the ViewHolder but it is not compulsion but in RecyclerView it is mandatory to use ViewHolder which is the main difference between RecyclerView and ListView.

ViewHolder is a static inner class in our Adapter which holds references to the relevant view's. By using these references our code can avoid time consuming findViewById() method to update the widgets with new data.

3. Adapters: In ListView we use many Adapter's like ArrayAdapter for displaying simple array data, BaseAdapter and SimpleAdapters for custom Lists. In RecyclerView we only use RecyclerView.Adapter to set the data in List. In Below code sinppet we show how our CustomAdapter looks when we extends RecyclerView.Adapter class and use ViewHolder in it.

```
package abhiandroid.com.recyclerviewexample;
 import android.support.v7.widget.RecyclerView;
 import android.view.LayoutInflater;
 import android.view.View;
 import android.view.ViewGroup;
 import android.widget.TextView;
 public class CustomAdapter extends RecyclerView.Adapter {
 @Override
 public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
 // infalte the item Layout
 View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.rowlayout, parent,
false);
 // set the view's size, margins, paddings and layout parameters
 MyViewHolder vh = new MyViewHolder(v); // pass the view to View Holder
 return vh;
 }
 @Override
 public void onBindViewHolder(MyViewHolder holder, int position) {
 }
 @Override
 public int getItemCount() {
 return 0;
 }
 public class MyViewHolder extends RecyclerView.ViewHolder {
 TextView textView;// init the item view's
 public MyViewHolder(View itemView) {
 super(itemView);
```

```
// get the reference of item view's
 textView = (TextView) itemView.findViewById(R.id.textView);
 }
 }
 }
```

4. Item Animator: ListView are lacking in support of good animation. RecyclerView brings a new dimensions in it. By using RecyclerView. ItemAnimator class we can easily animate the view.

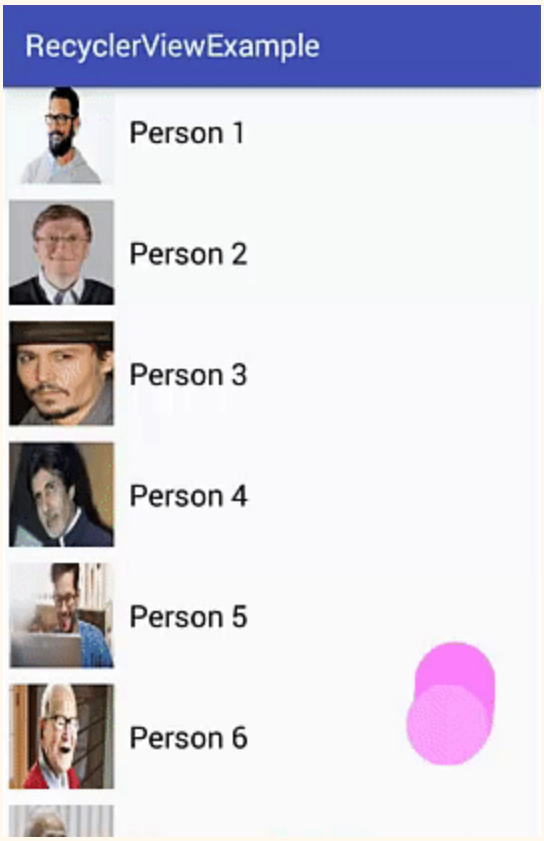
5. Item Decoration: In ListView dynamically decorating items like adding divider or border was not easy but in RecyclerView by using RecyclerView. ItemDecorator class we have a huge control on it.

Conclusion: At the end we can say that RecyclerView is much more customizable than the existing ListView and gives a lots of control and power to developers.

Example Of RecyclerView As Vertical ListView In Android Studio:

Below is the example of RecyclerView As ListView in which we display the vertical list of Person Names with their images by using RecyclerView. In this example we are using LinearLayoutManger with vertical orientation to display the items.

Firstly we declare a RecyclerView in our XML file and then get the reference of it in our Activity. After that we creates two ArrayList for Person Names and Images. After that we set a LayoutManager and finally we set the Adapter to show the items in RecyclerView. Whenever a user clicks on an item the name of the Person is displayed on the screen with the help of Toast.



Step 1: Create A New Project And Name It RecyclerViewExample.

Step 2: Open Gradle Scripts > build.gradle and add RecyclerView Library dependency in it.

```
apply plugin: 'com.android.application'
android {
compileSdkVersion 24
buildToolsVersion "24.0.1"
defaultConfig {
applicationId "abhiandroid.com.recyclerviewexample"
minSdkVersion 16
targetSdkVersion 24
versionCode 1
versionName "1.0"
}
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
}
}
}
dependencies {
```

```
compile fileTree(dir: 'libs', include: ['*.jar'])
testCompile 'junit:junit:4.12'
compile 'com.android.support:appcompat-v7:24.1.1'
compile "com.android.support:recyclerview-v7:23.0.1" // dependency file for RecyclerView
}
```

Step 3: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we create a RecyclerView in our XML file.

```
<?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"
    tools:context="abhiandroid.com.recyclerviewexample.MainActivity">
    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

Step 4: Create a new XML file rowlayout.xml for item of RecyclerView and paste the following code in it.

In this step we create a new xml file for item row in which we creates a TextView and ImageView to show the data.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="5dp">
    <!--
    items for a single row of RecyclerView
    -->
    <ImageView
        android:id="@+id/image"
        android:layout_width="70dp"
        android:layout_height="70dp"
        android:scaleType="fitXY"
        android:src="@mipmap/ic_launcher" />
    <TextView
```

```
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:layout_marginLeft="10dp"
        android:text="ABCD"
        android:textColor="#000"
        android:textSize="20sp" />
</LinearLayout>
```

Step 5 : Now open app -> java -> package -> MainActivity.java and add the below code.

In this step firstly we get the reference of RecyclerView. After that we creates two ArrayList‘s for Person Names and Images. After that we set a LayoutManager and finally we set the Adapter to show the items in RecyclerView.

```
package abhiandroid.com.recyclerviewexample;
 import android.os.Bundle;
 import android.support.v7.app.AppCompatActivity;
 import android.support.v7.widget.LinearLayoutManager;
 import android.support.v7.widget.RecyclerView;
 import java.util.ArrayList;
 import java.util.Arrays;
 public class MainActivity extends AppCompatActivity {
 // ArrayList for person names
 ArrayList personNames = new ArrayList<>(Arrays.asList("Person 1", "Person 2", "Person 3",
"Person 4", "Person 5", "Person 6", "Person 7","Person 8", "Person 9", "Person 10", "Person
11", "Person 12", "Person 13", "Person 14"));
 ArrayList personImages = new ArrayList<>(Arrays.asList(R.drawable.person1,
R.drawable.person2, R.drawable.person3, R.drawable.person4, R.drawable.person5,
R.drawable.person6, R.drawable.person7,R.drawable.person1, R.drawable.person2,
R.drawable.person3, R.drawable.person4, R.drawable.person5, R.drawable.person6,
R.drawable.person7));
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 // get the reference of RecyclerView
 RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
 // set a LinearLayoutManager with default vertical orientation
 LinearLayoutManager linearLayoutManager = new LinearLayoutManager(getApplicationContext());
 recyclerView.setLayoutManager(linearLayoutManager);
 // call the constructor of CustomAdapter to send the reference and data to Adapter
 CustomAdapter customAdapter = new CustomAdapter(MainActivity.this,
personNames,personImages);
 recyclerView.setAdapter(customAdapter); // set the Adapter to RecyclerView
 }
 }
```

Step 6: Create a new class CustomAdapter.java inside package and add the following code.

In this step we create a CustomAdapter class and extends RecyclerView.Adapter class with View Holder in it. After that we implement the overrided methods and create a constructor for getting the data from Activity. In this custom Adapter two methods are more important first is onCreateViewHolder in which we inflate the layout item xml and pass it to View Holder and other is onBindViewHolder in which we set the data in the view's with the help of View Holder. Finally we implement the setOnClickListener event on itemview and on click of item we display the name of the person with the help of Toast.

```
package abhiandroid.com.recyclerviewexample;
 import android.content.Context;
 import android.support.v7.widget.RecyclerView;
 import android.view.LayoutInflater;
 import android.view.View;
 import android.view.ViewGroup;
 import android.widget.ImageView;
 import android.widget.TextView;
 import android.widget.Toast;
 import java.util.ArrayList;
 public class CustomAdapter extends RecyclerView.Adapter {
 ArrayList personNames;
 ArrayList personImages;
 Context context;
 public CustomAdapter(Context context, ArrayList personNames, ArrayList personImages) {
 this.context = context;
 this.personNames = personNames;
 this.personImages = personImages;
 }
 @Override
 public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
 // infalte the item Layout
 View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.rowlayout, parent,
false);
 // set the view's size, margins, paddings and layout parameters
 MyViewHolder vh = new MyViewHolder(v); // pass the view to View Holder
 return vh;
 }
 @Override
 public void onBindViewHolder(MyViewHolder holder, final int position) {
 // set the data in items
 holder.name.setText(personNames.get(position));
 holder.image.setImageResource(personImages.get(position));
 // implement setOnClickListener event on item view.
 holder.itemView.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View view) {
 // display a toast with person name on item click
 Toast.makeText(context, personNames.get(position), Toast.LENGTH_SHORT).show();
 }
 });
```

```
}
@Override
public int getItemCount() {
return personNames.size();
}
public class MyViewHolder extends RecyclerView.ViewHolder {
// init the item view's
TextView name;
ImageView image;
public MyViewHolder(View itemView) {
super(itemView);
// get the reference of item view's
name = (TextView) itemView.findViewById(R.id.name);
image = (ImageView) itemView.findViewById(R.id.image);
}
}
}
```

Output:

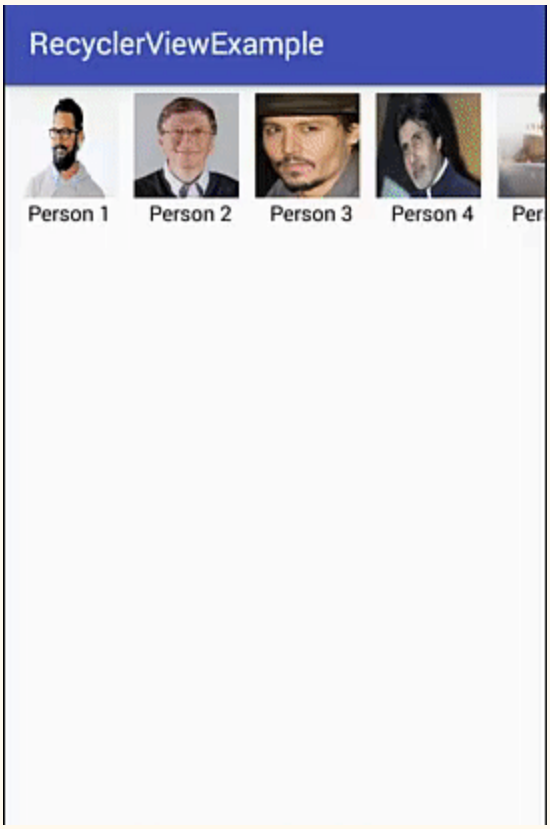
Now run the App and you will person name which can be scrolled in horizontal direction created using RecyclerView.

Example Of RecyclerView As Horizontal ListView In Android Studio

Below is the example of RecyclerView As ListView in which we display the horizontal list of Person Names with their images by using RecyclerView. In this example we are using LinearLayoutManger with horizontal orientation to display the items.

Firstly we declare a RecyclerView in our XML file and then get the reference of it in our Activity. After that we creates two ArrayList's for Person Names and Images.

After that we set a LayoutManager with horizontal orientation and finally we set the Adapter to show the items in RecyclerView. Whenever a user clicks on an item the name of the Person is displayed on the screen with the help of Toast.



Step 1: Create A New Project And Name It RecyclerViewExample.

Step 2: Open Gradle Scripts > build.gradle and add RecyclerView Library dependency in it.

```
apply plugin: 'com.android.application'
 android {
 compileSdkVersion 24
 buildToolsVersion "24.0.1"
 defaultConfig {
 applicationId "abhiandroid.com.recyclerviewexample"
 minSdkVersion 16
 targetSdkVersion 24
 versionCode 1
 versionName "1.0"
 }
 buildTypes {
 release {
 minifyEnabled false
 proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
 }
 }
```

```
}
dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
testCompile 'junit:junit:4.12'
compile 'com.android.support:appcompat-v7:24.1.1'
compile "com.android.support:recyclerview-v7:23.0.1" // dependency file for RecyclerView
}
```

Step 3: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we create a RecyclerView in our XML file.

```
<?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"
tools:context="abhiandroid.com.recyclerviewexample.MainActivity">
<android.support.v7.widget.RecyclerView
android:id="@+id/recyclerView"
android:layout_width="match_parent"
android:layout_height="match_parent" />
</RelativeLayout>
```

Step 4: Create a new XML file rowlayout.xml for item of RecyclerView and paste the following code in it.

In this step we create a new xml file for item row in which we creates a TextView and ImageView to show the data.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="vertical"
android:padding="5dp">
<!--
items for a single row of RecyclerView
-->
<ImageView
android:id="@+id/image"
android:layout_width="70dp"
android:layout_height="70dp"
android:scaleType="fitXY"
android:src="@mipmap/ic_launcher" />
<TextView
android:id="@+id/name"
```

```
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_gravity="center_vertical"
 android:layout_marginLeft="10dp"
 android:text="ABCD"
 android:textColor="#000" />
 </LinearLayout>
```

Step 5 : Now open app -> java -> package -> MainActivity.java and add the below code.

In this step firstly we get the reference of RecyclerView. After that we creates two ArrayList's for Person Names and Images. After that we set a LayoutManager with horizontal orientation and finally we set the Adapter to show the items in RecyclerView.

```
package abhiandroid.com.recyclerviewexample;
 import android.os.Bundle;
 import android.support.v7.app.AppCompatActivity;
 import android.support.v7.widget.LinearLayoutManager;
 import android.support.v7.widget.RecyclerView;
 import java.util.ArrayList;
 import java.util.Arrays;
 public class MainActivity extends AppCompatActivity {
 // ArrayList for person names
 ArrayList personNames = new ArrayList<>(Arrays.asList("Person 1", "Person 2", "Person 3",
"Person 4", "Person 5", "Person 6", "Person 7","Person 8", "Person 9", "Person 10", "Person
11", "Person 12", "Person 13", "Person 14"));
 ArrayList personImages = new ArrayList<>(Arrays.asList(R.drawable.person1,
R.drawable.person2, R.drawable.person3, R.drawable.person4, R.drawable.person5,
R.drawable.person6, R.drawable.person7,R.drawable.person1, R.drawable.person2,
R.drawable.person3, R.drawable.person4, R.drawable.person5, R.drawable.person6,
R.drawable.person7));
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 // get the reference of RecyclerView
 RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
 // set a LinearLayoutManager with default horizontal orientation and false value for
reverseLayout to show the items from start to end
 LinearLayoutManager linearLayoutManager = new
LinearLayoutManager(getApplicationContext(),LinearLayoutManager.HORIZONTAL,false);
 recyclerView.setLayoutManager(linearLayoutManager);
 // call the constructor of CustomAdapter to send the reference and data to Adapter
 CustomAdapter customAdapter = new CustomAdapter(MainActivity.this,
personNames,personImages);
 recyclerView.setAdapter(customAdapter); // set the Adapter to RecyclerView
 }
 }
```

Step 6: Create a new class CustomAdapter.java inside package and add the following code.

In this step we create a CustomAdapter class and extends RecyclerView.Adapter class with View Holder in it. After that we implement the overrided methods and create a constructor for getting the data from Activity, In this custom Adapter two methods are more important first is onCreateViewHolder in which we inflate the layout item xml and pass it to View Holder and other is onBindViewHolder in which we set the data in the view's with the help of View Holder. Finally we implement the setOnClickListener event on itemview and on click of item we display the name of the person with the help of Toast.

```
package abhiandroid.com.recyclerviewexample;
 import android.content.Context;
 import android.support.v7.widget.RecyclerView;
 import android.view.LayoutInflater;
 import android.view.View;
 import android.view.ViewGroup;
 import android.widget.ImageView;
 import android.widget.TextView;
 import android.widget.Toast;
 import java.util.ArrayList;
 public class CustomAdapter extends RecyclerView.Adapter {
 ArrayList personNames;
 ArrayList personImages;
 Context context;
 public CustomAdapter(Context context, ArrayList personNames, ArrayList personImages) {
 this.context = context;
 this.personNames = personNames;
 this.personImages = personImages;
 }
 @Override
 public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
 // infalte the item Layout
 View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.rowlayout, parent,
false);
 // set the view's size, margins, paddings and layout parameters
 MyViewHolder vh = new MyViewHolder(v); // pass the view to View Holder
 return vh;
 }
 @Override
 public void onBindViewHolder(MyViewHolder holder, final int position) {
 // set the data in items
 holder.name.setText(personNames.get(position));
 holder.image.setImageResource(personImages.get(position));
 // implement setOnClickListener event on item view.
 holder.itemView.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View view) {
 // display a toast with person name on item click
 Toast.makeText(context, personNames.get(position), Toast.LENGTH_SHORT).show();
 }
 });
```

```
}
@Override
public int getItemCount() {
return personNames.size();
}
public class MyViewHolder extends RecyclerView.ViewHolder {
// init the item view's
TextView name;
ImageView image;
public MyViewHolder(View itemView) {
super(itemView);
// get the reference of item view's
name = (TextView) itemView.findViewById(R.id.name);
image = (ImageView) itemView.findViewById(R.id.image);
}
}
}
```

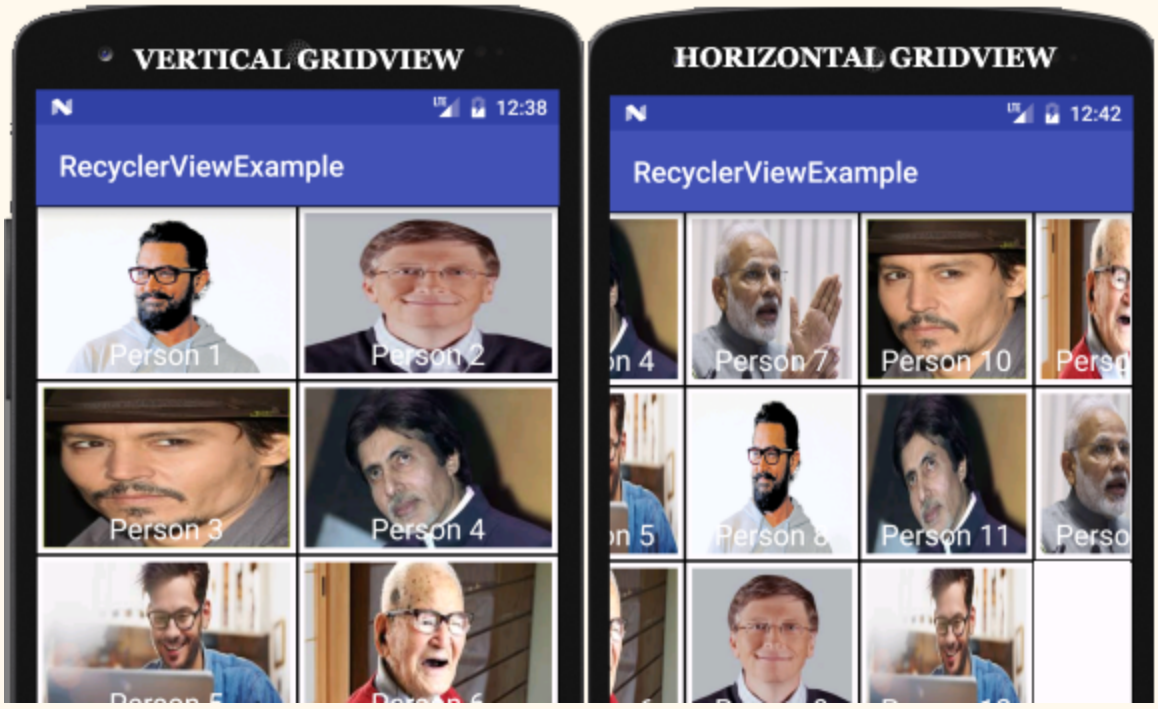
Output:

Now run the App and you will person name which can be scrolled in horizontal direction created using RecyclerView.

RecyclerView Using GridLayoutManager

In Android, RecyclerView is an advance and flexible version of ListView and GridView. It is a container used to display large amount of data sets that can be scrolled very efficiently by maintaining a limited number of views. RecyclerView was introduced in Material Design in API level 21 (Android 5.0 i.e Lollipop).

Material Design brings lot of new features in Android that changed a lot the visual design patterns regarding the designing of modern Android applications. This new widget is a big step for displaying data because the GridView is one of the most commonly used UI widget. In RecyclerView android provides a lot of new features which are not present in existing ListView or GridView.



Basic RecyclerView XML code:

```
<?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"
    tools:context="abhiandroid.com.recyclerviewexample.MainActivity">
    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

Gradle Dependency to use RecyclerView:

The RecyclerView widget is a part of separate library valid for API 7 level or higher. Add the following dependency in your Gradle build file to use recyclerview.

```
dependencies {
 ...
 compile "com.android.support:recyclerview-v7:23.23.4.0"
 }
```

RecyclerView As GridView In Android:

In this article we will discuss how to use a RecyclerView As GridView. For that we need to understand LayoutManager component of RecyclerView. Layout Manager is a very new concept introduced in RecyclerView for defining the type of Layout which RecyclerView should use. It Contains the references for all the views that are filled by the data of the entry. We can create a Custom Layout Manager by extending RecyclerView.LayoutManager Class but RecyclerView Provides three types of in-built Layout Managers.

1. LinearLayoutManager: It is used for displaying Vertical or Horizontal List. To understand it please read RecyclerView as Listview

2. GridLayoutManager: It is used for displaying the items in the form of Grids.

3. StaggeredGridLayoutManager: It is used to show the items in staggered Grid.

In this article our main focus is on GridLayoutManager because it is used to display the data in the form of grids. By using this Layout Manager we can easily create grid items. A common example of grid items is our phone's gallery in which all the images are displayed in the form of grids.

GridLayoutManager is used for displaying the data items in grid format and we can easily define the orientation for the items. In Simple words we can say that we use the GridLayoutManager for displaying RecyclerView as a GridView.

Public constructors for GridLayoutManager: Below we define the public constructor for GridLayoutManager that should be used for defining orientation(vertical or horizontal) of RecyclerView.

1- GridLayoutManager (Context context, int spanCount): It is used to create a Vertical GridLayoutManager. In this constructor first parameter is used to set the current context and second parameter is used to set the span Count means the number of columns in the grid.

Example: In below code snippet we show how to use this constructor in Android.

With Default Vertical Orientation:

```
// get the reference of RecyclerView
RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
// set a GridLayoutManager with default vertical orientation and 3 number of columns
GridLayoutManager gridLayoutManager = new GridLayoutManager(getApplicationContext(),3);
recyclerView.setLayoutManager(gridLayoutManager); // set LayoutManager to RecyclerView
```

With Horizontal Orientation:

```
// get the reference of RecyclerView
RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
// set a GridLayoutManager with 3 number of columns
GridLayoutManager gridLayoutManager = new GridLayoutManager(getApplicationContext(),3);
gridLayoutManager.setOrientation(LinearLayoutManager.HORIZONTAL); // set Horizontal
Orientation
recyclerView.setLayoutManager(gridLayoutManager); // set LayoutManager to RecyclerView
```

2- GridLayoutManager (Context context, int spanCount, int orientation, boolean reverseLayout): In this constructor first parameter is used to set the current context, second parameter is used to set the span Count means the number of columns in the grid, third parameter is used to set the Layout Orientation should be vertical or horizontal and last param is a boolean value when sets to true layout from end to start means grids are arranged from end to start.

Example: In below code snippet we show how to use this constructor in Android.

```
// get the reference of RecyclerView
RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
// set a GridLayoutManager with 3 number of columns , horizontal gravity and false value for
reverseLayout to show the items from start to end
GridLayoutManager gridLayoutManager = new
GridLayoutManager(getApplicationContext(),3,LinearLayoutManager.HORIZONTAL,false);
recyclerView.setLayoutManager(gridLayoutManager); // set LayoutManager to RecyclerView
```

Comparison between RecyclerView and GridView

There are a lot of new features in RecyclerView that are not present in existing GridView. The RecyclerView is more flexible, powerful and a major enhancement over GridView. I will try to give you a detailed insight into it. Below we discuss some important features of RecyclerView that should clear the reason why RecyclerView is better than GridView.

1. Custom Item Layouts: GridView can only layout the grid items in Vertical Arrangement in which we set the number of columns and rows are automatically creates according to number of items. GridView cannot be customized according to our requirements. Suppose we need to show

items in horizontal arrangement in which we want to fix number of rows and columns are automatically creates according to number of items but that thing is not feasible with default GridView. But with introduction of Recyclerview we can easily create a horizontal or Vertical arrangement for grid items. By using GridLayoutManager component of RecyclerView we can easily define the orientation of grid items and spanCount used for number of columns if orientation is vertical or number of rows if orientation is horizontal.

2. Use Of ViewHolder Pattern: GridView Adapters do not require the use of ViewHolder but RecyclerView require the use of ViewHolder that is used to store the reference of View's. In GridView it is recommended to use the ViewHolder but it is not compulsion but in RecyclerView it is mandatory to use ViewHolder that is the main difference between RecyclerView and GridView. ViewHolder is a static inner class in our Adapter which holds references to the relevant view's. By using these references our code can avoid time consuming findViewById() method to update the widgets with new data.

3. GridView Adapters: In GridView we use many Adapter's like ArrayAdapter for displaying simple array data, Base and SimpleAdapters for custom grids with images and text. In RecyclerView we only use RecyclerView.Adapter to setting the grid items. In Below code snippet we show how our CustomAdapter looks when we extends RecyclerView.Adapter class and use ViewHolder in it.

```
package abhiandroid.com.recyclerviewexample;
 import android.support.v7.widget.RecyclerView;
 import android.view.LayoutInflater;
 import android.view.View;
 import android.view.ViewGroup;
 import android.widget.TextView;
 public class CustomAdapter extends RecyclerView.Adapter {
 @Override
 public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
 // infalte the item Layout
 View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.rowlayout, parent,
false);
 // set the view's size, margins, paddings and layout parameters
 MyViewHolder vh = new MyViewHolder(v); // pass the view to View Holder
 return vh;
 }
 @Override
```

```
 public void onBindViewHolder(MyViewHolder holder, int position) {
 }
 @Override
 public int getItemCount() {
 return 0;
 }
 public class MyViewHolder extends RecyclerView.ViewHolder {
 TextView textView;// init the item view's
 public MyViewHolder(View itemView) {
 super(itemView);

// get the reference of item view's
 textView = (TextView) itemView.findViewById(R.id.textView);
 }
 }
 }
```

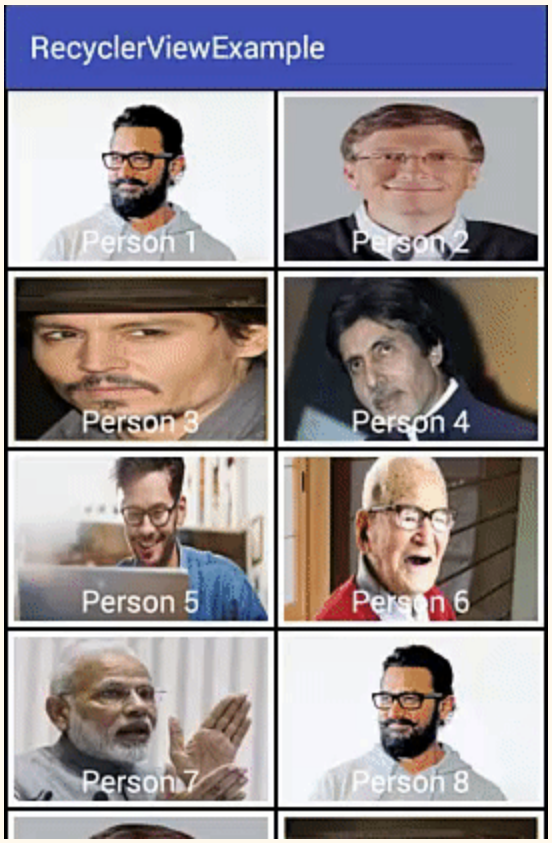
4. Item Animator: GridView are lacking in support of Good Animation's. RecyclerView brings a new dimensions in it. By using RecycleView.ItemAnimator class we can easily animate the view's.

5. Item Decoration: In GridView's Dynamically decorating items like Adding divider or border was never easy but in RecyclerView by using RecycleView.ItemDecorator class we have a huge control on it.

Conclusion: At the end we can say that RecyclerView is much more customizable than the existing GridView and gives a lot of control and power to its developers.

Example Of RecyclerView As Vertical GridView In Android Studio:

Below is the example of RecyclerView As GridView in which we display grids of Person Names with their images with default vertical orientation by using RecyclerView. In this example we are using GridLayoutManager with vertical orientation and 2 span count value to display the items. Firstly we declare a RecyclerView in our XML file and then get the reference of it in our Activity. After that we creates two ArrayList‘s for Person Names and Images. After that we set a GridLayoutManager and finally we set the Adapter to show the grid items in RecyclerView. Whenever a user clicks on an item the full size image will be displayed on the next screen.



Step 1: Create a New Project And Name It RecyclerViewExample.

Step 2: Open Gradle Scripts > build.gradle and add RecyclerView Library dependency in it.

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 24
    buildToolsVersion "24.0.1"
    defaultConfig {
        applicationId "abhiandroid.com.recyclerviewexample"
        minSdkVersion 16
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
```

```
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:24.1.1'
    compile "com.android.support:recyclerview-v7:23.4.0" // dependency file for RecyclerView
}
```

Step 3: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we create a RecyclerView in our XML file.

```
<?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"
    tools:context="abhiandroid.com.recyclerviewexample.MainActivity">
    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

Step 4: Create a new drawable XML file in Drawable folder and name it custom_item_layout.xml and add the following code in it for creating a custom grid item.

In this step we create a drawable XML file in which we add the code for creating custom grid items.

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <!--
       stroke with color and width for creating outer line
    -->
    <stroke
        android:width="1dp"
        android:color="#000" />
</shape>
```

Step 5: Create a new XML file rowlayout.xml for grid item of RecyclerView and paste the following code in it.

In this step we create a new xml file for item row in which we creates a TextView and ImageView to show the data in grid format.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:background="@drawable/custom_item_layout"
    android:padding="5dp">
    <!--
    grid items for RecyclerView
    -->
    <ImageView
        android:id="@+id/image"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:scaleType="fitXY"
        android:src="@mipmap/ic_launcher" />
    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:text="ABCD"
        android:textSize="20sp"
        android:textColor="#fff" />
</RelativeLayout>
```

Step 6 : Now open app -> java -> package -> MainActivity.java and add the below code.

In this step firstly we get the reference of RecyclerView. After that we creates two ArrayList‘s for Person Names and Images. After that we set a GridLayoutManager and finally we set the Adapter to show the grid items in RecyclerView.

```
package abhiandroid.com.recyclerviewexample;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.GridLayoutManager;
import android.support.v7.widget.RecyclerView;
import java.util.ArrayList;
import java.util.Arrays;
public class MainActivity extends AppCompatActivity {
```

```
    // ArrayList for person names
    ArrayList personNames = new ArrayList<>(Arrays.asList("Person 1", "Person 2", "Person
3", "Person 4", "Person 5", "Person 6", "Person 7","Person 8", "Person 9", "Person 10",
"Person 11", "Person 12", "Person 13", "Person 14"));
    ArrayList personImages = new ArrayList<>(Arrays.asList(R.drawable.person1,
R.drawable.person2, R.drawable.person3, R.drawable.person4, R.drawable.person5,
R.drawable.person6, R.drawable.person7,R.drawable.person1, R.drawable.person2,
R.drawable.person3, R.drawable.person4, R.drawable.person5, R.drawable.person6,
R.drawable.person7));
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // get the reference of RecyclerView
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
        // set a GridLayoutManager with default vertical orientation and 2 number of columns
        GridLayoutManager gridLayoutManager = new
GridLayoutManager(getApplicationContext(),2);
        recyclerView.setLayoutManager(gridLayoutManager); // set LayoutManager to
RecyclerView
        //  call the constructor of CustomAdapter to send the reference and data to Adapter
        CustomAdapter customAdapter = new CustomAdapter(MainActivity.this,
personNames,personImages);
        recyclerView.setAdapter(customAdapter); // set the Adapter to RecyclerView
    }
}
```

Step 7: Create a new class CustomAdapter.java inside package and add the following code.

In this step we create a CustomAdapter class and extends RecyclerView.Adapter class with Viewholder in it. After that we implement the overrided methods and create a constructor for getting the data from Activity. In this custom Adapter two methods are more important first is onCreateViewHolder in which we inflate the layout item xml and pass it to View Holder and other is onBindViewHolder in which we set the data in the view's with the help of ViewHolder. Finally we implement the setOnClickListener event on itemview and on click of item we display the selected image in full size in the next Activity.

```
package abhiandroid.com.recyclerviewexample;
import android.content.Context;
import android.content.Intent;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;
import java.util.ArrayList;
public class CustomAdapter extends RecyclerView.Adapter {
```

```
    ArrayList personNames;
    ArrayList personImages;
    Context context;
    public CustomAdapter(Context context, ArrayList personNames, ArrayList personImages) {
        this.context = context;
        this.personNames = personNames;
        this.personImages = personImages;
    }
    @Override
    public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        // infalte the item Layout
        View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.rowlayout,
parent, false);
        // set the view's size, margins, paddings and layout parameters
        MyViewHolder vh = new MyViewHolder(v); // pass the view to View Holder
        return vh;
    }
    @Override
    public void onBindViewHolder(MyViewHolder holder, final int position) {
        // set the data in items
        holder.name.setText(personNames.get(position));
        holder.image.setImageResource(personImages.get(position));
        // implement setOnClickListener event on item view.
        holder.itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // open another activity on item click
                Intent intent = new Intent(context, SecondActivity.class);
                intent.putExtra("image", personImages.get(position)); // put image data in
Intent
                context.startActivity(intent); // start Intent
            }
        });
    }
    @Override
    public int getItemCount() {
        return personNames.size();
    }
    public class MyViewHolder extends RecyclerView.ViewHolder {
        // init the item view's
        TextView name;
        ImageView image;
        public MyViewHolder(View itemView) {
            super(itemView);
            // get the reference of item view's
            name = (TextView) itemView.findViewById(R.id.name);
            image = (ImageView) itemView.findViewById(R.id.image);
        }
    }
}
```

Step 8: Create a new XML file activity_second.xml and add below code in it.

In this step we create a ImageView in our XML file to show the selected image in full size.

```
<?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"
    android:background="#fff">
    <ImageView
        android:id="@+id/selectedImage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:scaleType="fitXY" />
</RelativeLayout>
```

Step 9: Create a new Activity and name it SecondActivity.class and add the below code in it.

In this step we get the reference of ImageView and then get Intent which was set from adapter of Previous Activity and then finally we set the image in ImageView.

```
package abhiandroid.com.recyclerviewexample;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.widget.ImageView;
public class SecondActivity extends AppCompatActivity {
    ImageView selectedImage;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        selectedImage = (ImageView) findViewById(R.id.selectedImage); // init a ImageView
        Intent intent = getIntent(); // get Intent which was set from adapter of Previous
Activity
        selectedImage.setImageResource(intent.getIntExtra("image", 0)); // get image from
Intent and set it in ImageView
    }
}
```

Output:

Now run the App and you will see person name which can be scrolled in vertical direction created using RecyclerView as Gridview.

Example of RecyclerView As Horizontal GridView In Android Studio

Below is the example of RecyclerView As GridView in which we display the grids of Person Names with their images with horizontal orientation by using RecyclerView. In this example we are using GridLayoutManager with horizontal orientation and 3 span count value which defines the number of rows. Firstly we declare a RecyclerView in our XML file and then get the reference of it in our Activity. After that we creates two ArrayList's for Person Names and Images. After that we set a GridLayoutManager and finally we set the Adapter to show the grid items in RecyclerView. Whenever a user clicks on an item the full size image will be displayed on the next screen..

Step 1: Create a New Project and name it RecyclerViewExample.

Step 2: Open Gradle Scripts > build.gradle and add RecyclerView Library dependency in it.

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 24
    buildToolsVersion "24.0.1"
    defaultConfig {
        applicationId "abhiandroid.com.recyclerviewexample"
        minSdkVersion 16
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:24.1.1'
    compile "com.android.support:recyclerview-v7:23.4.0" // dependency file for RecyclerView
}
```

Step 3: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we create a RecyclerView in our XML file.

```
<?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"
    tools:context="abhiandroid.com.recyclerviewexample.MainActivity">
    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

Step 4: Create a new drawable XML file in Drawable folder and name it custom_item_layout.xml and add the following code in it for creating a custom grid item.

In this step we create a drawable XML file in which we add the code for creating custom grid items.

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <!--
       stroke with color and width for creating outer line
    -->
    <stroke
        android:width="1dp"
        android:color="#000" />
</shape>
```

Step 5: Create a new XML file rowlayout.xml for grid item of RecyclerView and paste the following code in it.

In this step we create a new xml file for item row in which we creates a TextView and ImageView to show the data in grid format.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="120dp"
    android:layout_height="120dp"
    android:background="@drawable/custom_item_layout"
    android:padding="5dp">
    <!--
    grid items for RecyclerView
    -->
    <ImageView
        android:id="@+id/image"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitXY"
        android:src="@mipmap/ic_launcher" />
    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:text="ABCD"
        android:textSize="20sp"
        android:textColor="#fff" />
</RelativeLayout>
```

Step 6 : Now open app -> java -> package -> MainActivity.java and add the below code.

In this step firstly we get the reference of RecyclerView. After that we creates two ArrayList's for Person Names and Images. After that we set a GridLayoutManager with horizontal orientation

and false value for reverseLayout to show the grids to show the items from start to end and then finally we set the Adapter to show the grid items in RecyclerView.

```
package abhiandroid.com.recyclerviewexample;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.GridLayoutManager;
import android.support.v7.widget.RecyclerView;
import java.util.ArrayList;
import java.util.Arrays;
public class MainActivity extends AppCompatActivity {
    // ArrayList for person names
    ArrayList personNames = new ArrayList<>(Arrays.asList("Person 1", "Person 2", "Person
3", "Person 4", "Person 5", "Person 6", "Person 7","Person 8", "Person 9", "Person 10",
"Person 11", "Person 12", "Person 13", "Person 14"));
    ArrayList personImages = new ArrayList<>(Arrays.asList(R.drawable.person1,
R.drawable.person2, R.drawable.person3, R.drawable.person4, R.drawable.person5,
R.drawable.person6, R.drawable.person7,R.drawable.person1, R.drawable.person2,
R.drawable.person3, R.drawable.person4, R.drawable.person5, R.drawable.person6,
R.drawable.person7));
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // get the reference of RecyclerView
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
        // set a GridLayoutManager with 3 number of columns , horizontal gravity and false
value for reverseLayout to show the items from start to end
        GridLayoutManager gridLayoutManager = new
GridLayoutManager(getApplicationContext(),3, LinearLayoutManager.HORIZONTAL,false);
        recyclerView.setLayoutManager(gridLayoutManager); // set LayoutManager to
RecyclerView
        //  call the constructor of CustomAdapter to send the reference and data to Adapter
        CustomAdapter customAdapter = new CustomAdapter(MainActivity.this,
personNames,personImages);
        recyclerView.setAdapter(customAdapter); // set the Adapter to RecyclerView
    }
}
```

Step 7: Create a new class CustomAdapter.java inside package and add the following code.

In this step we create a CustomAdapter class and extends RecyclerView.Adapter class with ViewHolder in it. After that we implement the overrided methods and create a constructor for getting the data from Activity, in this custom Adapter two methods are more important first is onCreateViewHolder in which we inflate the layout item xml and pass it to View Holder and other is onBindViewHolder in which we set the data in the view's with the help of ViewHolder. Finally we implement the setOnClickListener event on itemview and on click of item we display the selected image in full size in the next Activity.

```
package abhiandroid.com.recyclerviewexample;
import android.content.Context;
import android.content.Intent;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;
import java.util.ArrayList;
public class CustomAdapter extends RecyclerView.Adapter {
    ArrayList personNames;
    ArrayList personImages;
    Context context;
    public CustomAdapter(Context context, ArrayList personNames, ArrayList personImages) {
        this.context = context;
        this.personNames = personNames;
        this.personImages = personImages;
    }
    @Override
    public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        // infalte the item Layout
        View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.rowlayout,
parent, false);
        // set the view's size, margins, paddings and layout parameters
        MyViewHolder vh = new MyViewHolder(v); // pass the view to View Holder
        return vh;
    }
    @Override
    public void onBindViewHolder(MyViewHolder holder, final int position) {
        // set the data in items
        holder.name.setText(personNames.get(position));
        holder.image.setImageResource(personImages.get(position));
        // implement setOnClickListener event on item view.
        holder.itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // open another activity on item click
                Intent intent = new Intent(context, SecondActivity.class);
                intent.putExtra("image", personImages.get(position)); // put image data in
Intent
                context.startActivity(intent); // start Intent
            }
        });
    }
    @Override
    public int getItemCount() {
        return personNames.size();
    }
    public class MyViewHolder extends RecyclerView.ViewHolder {
        // init the item view's
        TextView name;
        ImageView image;
        public MyViewHolder(View itemView) {
            super(itemView);
            // get the reference of item view's
```

```
            name = (TextView) itemView.findViewById(R.id.name);
            image = (ImageView) itemView.findViewById(R.id.image);
        }
    }
}
```

Step 8: Create a new XML file activity_second.xml and add below code in it.

In this step we create a ImageView in our XML file to show the selected image in full size.

```
<?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"
    android:background="#fff">
    <ImageView
        android:id="@+id/selectedImage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:scaleType="fitXY" />
</RelativeLayout>
```

Step 9: Create a new Activity and name it SecondActivity.class and add the below code in it.

In this step we get the reference of ImageView and then get Intent which was set from adapter of Previous Activity and then finally we set the image in ImageView.

```
package abhiandroid.com.recyclerviewexample;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.widget.ImageView;
public class SecondActivity extends AppCompatActivity {
    ImageView selectedImage;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        selectedImage = (ImageView) findViewById(R.id.selectedImage); // init a ImageView
```

```
        Intent intent = getIntent(); // get Intent which was set from adapter of Previous
Activity
        selectedImage.setImageResource(intent.getIntExtra("image", 0)); // get image from
Intent and set it in ImageView
    }
}
```

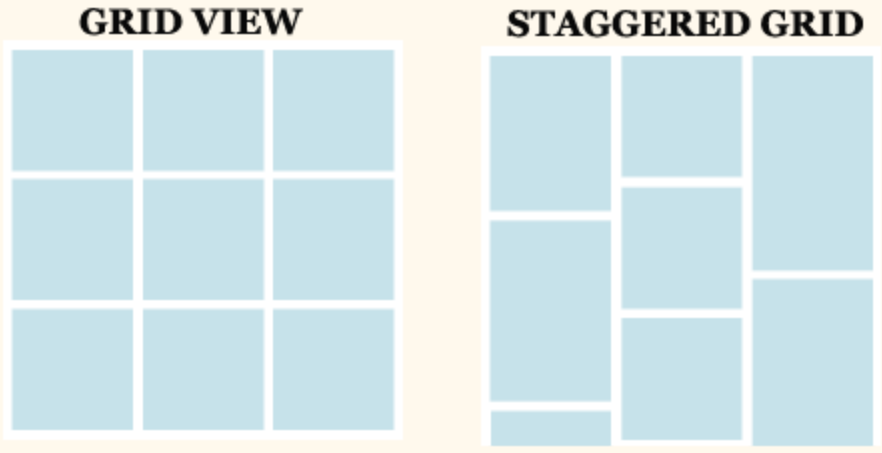
Output:

Now run the App and you will person name which can be scrolled in horizontal direction created using RecyclerView as GridView.

RecyclerView As Staggered Grid

In Android, RecyclerView is an advanced and flexible version of ListView and GridView. It is a container used to display large amount of data sets that can be scrolled very efficiently by maintaining a limited number of views. RecyclerView was first introduced in Material Design in API level 21 (Android 5.0 i.e Lollipop).

Material Design brings lot of new features in Android that changed a lot the visual design patterns regarding the designing of modern Android applications. This new widget is a big step for displaying data because the GridView is one of the most commonly used UI widget. In RecyclerView android provides a lot of new features which are not present in existing ListView or GridView.



Basic RecyclerView XML code:

```
<?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"
    tools:context="abhiandroid.com.recyclerviewexample.MainActivity">
    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

Gradle Dependency to use RecyclerView:

The RecyclerView widget is a part of separate library valid for API 7 level or higher. Add the following dependency in your Gradle build file to use recyclerview.

Gradle Scripts > build.gradle and inside dependencies

```
dependencies {
    ...
    compile "com.android.support:recyclerview-v7:23.4.0"
}
```

RecyclerView As Staggered Grid

In this article we will discuss how to use a RecyclerView for creating Staggered Grids. For that firstly we need to understand LayoutManager component of RecyclerView. Layout Manager is a very new concept introduced in RecyclerView for defining the type of Layout which RecyclerView should use. It contains the references for all the views that are filled by the data of the entry. We can create a Custom Layout Manager by extending RecyclerView.LayoutManager Class but RecyclerView Provides three types of in-built Layout Managers.

1. LinearLayoutManager: It is used for displaying Vertical or Horizontal List.

2. GridLayoutManager: It is used for displaying the items in the form of Grids.

3. StaggeredGridLayoutManager: It is used to show the items in staggered Grid.

In this article our main focus is on StaggeredGridLayoutManager because it is used to display the data in the form of staggered grids. By using this Layout Manager we can easily create staggered grid items.

In Android Pinterest and SamePinchh both apps are the best example of staggered grid. StaggeredGridLayoutManager is used for displaying the data items in staggered grid format and we can easily define the orientation for the items.

StaggeredGridLayoutManager is similar with GridLayoutManager but in this each grid have its own size(width and height). It automatically sets the items according to their height and width. The main difference between GridView and Staggered Grid is of varying size in staggered. Staggered Grid View shows asymmetric items in view.

Public constructor for StaggeredGridLayoutManager

• StaggeredGridLayoutManager(int spanCount, int orientation): It is used to create a StaggeredGridLayoutManager with given parameters. First parameter is used to set spanCount means number of columns if orientation is vertical or number of rows if orientation is horizontal, Second Parameter is used to set the Orientation, it should be vertical or horizontal.

Example: In below code snippet we show how to use this constructor in Android.

With Vertical Orientation:

```
// get the reference of RecyclerView
 RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
// set a StaggeredGridLayoutManager with 3 number of columns and vertical orientation
StaggeredGridLayoutManager staggeredGridLayoutManager = new StaggeredGridLayoutManager(3,
LinearLayoutManager.VERTICAL);
recyclerView.setLayoutManager(staggeredGridLayoutManager); // set LayoutManager to
RecyclerView
```

With Horizontal Orientation:

```
// get the reference of RecyclerView
 RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
 // set a StaggeredGridLayoutManager with 3 number of columns and horizontal orientation
 StaggeredGridLayoutManager staggeredGridLayoutManager = new StaggeredGridLayoutManager(3,
LinearLayoutManager.HORIZONTAL);
 recyclerView.setLayoutManager(staggeredGridLayoutManager); // set LayoutManager to
RecyclerView
```

Example Of Vertical Staggered GridView using RecyclerView In Android Studio

Below is the example of RecyclerView As GridView in which we display staggered grids of Person Names with their images and setting default vertical orientation using RecyclerView.

In this example we are using StaggeredGridLayoutManager with vertical orientation and 3 span count value to display the items. Firstly we declare a RecyclerView in our XML file and then get the reference of it in our Activity. After that we creates two ArrayList for Person Names and Images. After that we set a StaggeredGridLayoutManager and finally we set the Adapter to show the grid items in RecyclerView. Whenever a user clicks on an item the full size image will be displayed on the next screen.

Step 1: Create a New Project and Name it RecyclerViewExample.

Step 2: Open Gradle Scripts > build.gradle and add RecyclerView Library dependency in it.

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 24
    buildToolsVersion "24.0.1"
    defaultConfig {
        applicationId "abhiandroid.com.recyclerviewexample"
        minSdkVersion 16
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
```

```
    }
}
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:24.1.1'
    compile "com.android.support:recyclerview-v7:23.4.0" // dependency file for RecyclerView
}
```

Step 3: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we create a RecyclerView in our XML file.

```
<?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"
    tools:context="abhiandroid.com.recyclerviewexample.MainActivity">
    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

Step 4: Create a new drawable XML file in Drawable folder and name it custom_item_layout.xml and add the following code in it for creating a custom grid item.

In this step we create a drawable XML file in which we add the code for creating custom grid items.

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <!--
       stroke with color and width for creating outer line
    -->
    <stroke
        android:width="1dp"
        android:color="#000" />
</shape>
```

Step 5: Create a new XML file rowlayout.xml for grid item of RecyclerView and paste the following code in it.

In this step we create a new xml file for item row in which we creates a TextView and ImageView to show the data in grid format.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/custom_item_layout"
    android:padding="5dp">
    <!--
    grid items for RecyclerView
    -->
    <ImageView
        android:id="@+id/image"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitXY"
        android:src="@mipmap/ic_launcher" />
    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:text="ABCD"
        android:textSize="20sp"
        android:textColor="#fff" />
</RelativeLayout>
```

Step 6 : Now open app -> java -> package -> MainActivity.java and add the below code.

In this step firstly we get the reference of RecyclerView. After that we creates two ArrayList‘s for Person Names and Images. After that we set a StaggeredGridLayoutManager and finally we set the Adapter to show the grid items in RecyclerView..

```
package abhiandroid.com.recyclerviewexample;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.StaggeredGridLayoutManager;
import java.util.ArrayList;
import java.util.Arrays;
```

```
public class MainActivity extends AppCompatActivity {
    // ArrayList for person names
    ArrayList personNames = new ArrayList<>(Arrays.asList("Person 1", "Person 2", "Person
3", "Person 4", "Person 5", "Person 6", "Person 7", "Person 8", "Person 9", "Person 10",
"Person 11", "Person 12", "Person 13", "Person 14"));
    ArrayList personImages = new ArrayList<>(Arrays.asList(R.drawable.person1,
R.drawable.person2, R.drawable.person3, R.drawable.person4, R.drawable.person5,
R.drawable.person6, R.drawable.person7, R.drawable.person1, R.drawable.person2,
R.drawable.person3, R.drawable.person4, R.drawable.person5, R.drawable.person6,
R.drawable.person7));
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // get the reference of RecyclerView
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
        // set a StaggeredGridLayoutManager with 3 number of columns and vertical
orientation
        StaggeredGridLayoutManager staggeredGridLayoutManager = new
StaggeredGridLayoutManager(3, LinearLayoutManager.VERTICAL);
        recyclerView.setLayoutManager(staggeredGridLayoutManager); // set LayoutManager to
RecyclerView
        //  call the constructor of CustomAdapter to send the reference and data to Adapter
        CustomAdapter customAdapter = new CustomAdapter(MainActivity.this, personNames,
personImages);
        recyclerView.setAdapter(customAdapter); // set the Adapter to RecyclerView
    }
}
```

Step 7: Create a new class CustomAdapter.java inside package and add the following code.

In this step we create a CustomAdapter class and extends RecyclerView.Adapter class with Viewholder in it. After that we implement the overrided methods and create a constructor for getting the data from Activity. In this custom Adapter two methods are more important first is onCreateViewHolder in which we inflate the layout item xml and pass it to View Holder and other is onBindViewHolder in which we set the data in the view's with the help of ViewHolder. Finally we implement the setOnClickListener event on itemview and on click of item we display the selected image in full size in the next Activity.

```
package abhiandroid.com.recyclerviewexample;
import android.content.Context;
import android.content.Intent;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
```

```
import android.widget.ImageView;
import android.widget.TextView;
import java.util.ArrayList;
public class CustomAdapter extends RecyclerView.Adapter {
    ArrayList personNames;
    ArrayList personImages;
    Context context;
    public CustomAdapter(Context context, ArrayList personNames, ArrayList personImages) {
        this.context = context;
        this.personNames = personNames;
        this.personImages = personImages;
    }
    @Override
    public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        // infalte the item Layout
        View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.rowlayout,
parent, false);
        // set the view's size, margins, paddings and layout parameters
        MyViewHolder vh = new MyViewHolder(v); // pass the view to View Holder
        return vh;
    }
    @Override
    public void onBindViewHolder(MyViewHolder holder, final int position) {
        // set the data in items
        holder.name.setText(personNames.get(position));
        holder.image.setImageResource(personImages.get(position));
        // implement setOnClickListener event on item view.
        holder.itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // open another activity on item click
                Intent intent = new Intent(context, SecondActivity.class);
                intent.putExtra("image", personImages.get(position)); // put image data in
Intent
                context.startActivity(intent); // start Intent
            }
        });
    }
    @Override
    public int getItemCount() {
        return personNames.size();
    }
    public class MyViewHolder extends RecyclerView.ViewHolder {
        // init the item view's
        TextView name;
        ImageView image;
        public MyViewHolder(View itemView) {
            super(itemView);
            // get the reference of item view's
            name = (TextView) itemView.findViewById(R.id.name);
            image = (ImageView) itemView.findViewById(R.id.image);
        }
    }
}
```

Step 8: Create a new XML file activity_second.xml and add below code in it.

In this step we create a ImageView in our XML file to show the selected image in full size.

```
<?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"
    android:background="#fff">
    <ImageView
        android:id="@+id/selectedImage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:scaleType="fitXY" />
</RelativeLayout>
```

Step 9: Create a new Activity and name it SecondActivity.class and add the below code in it.

In this step we get the reference of ImageView and then get Intent which was set from adapter of Previous Activity and then finally we set the image in ImageView.

```
package abhiandroid.com.recyclerviewexample;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.widget.ImageView;
public class SecondActivity extends AppCompatActivity {
    ImageView selectedImage;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        selectedImage = (ImageView) findViewById(R.id.selectedImage); // init a ImageView
        Intent intent = getIntent(); // get Intent which was set from adapter of Previous
Activity
        selectedImage.setImageResource(intent.getIntExtra("image", 0)); // get image from
Intent and set it in ImageView
    }
}
```

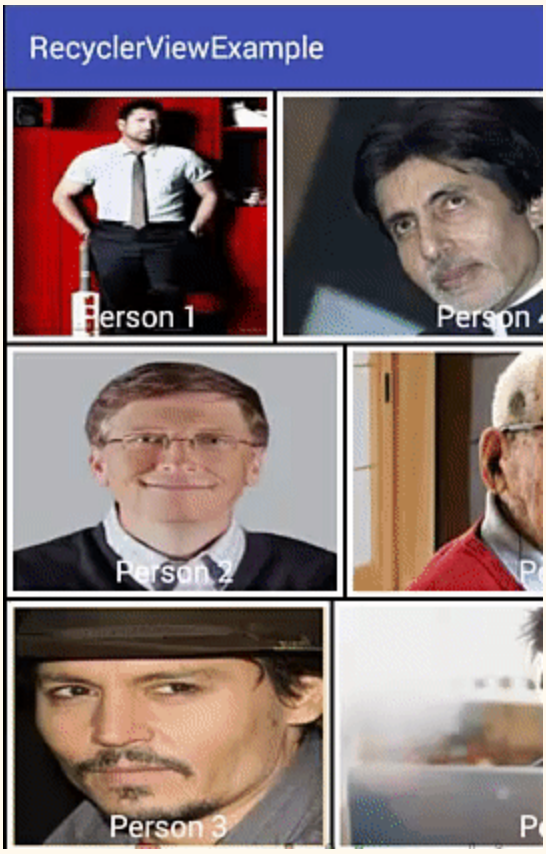
Output:

Now run the App and you will see person name in staggered grid which can be scrolled in vertical direction created using RecyclerView.

Example Of Horizontal Staggered GridView using RecyclerView In Android Studio

Below is the second example of RecyclerView As Staggered GridView in which we display the grids of Person Names with their images with horizontal orientation by using RecyclerView.

In this example we are using StaggeredGridLayoutManger with horizontal orientation and 3 span count value which defines the number of rows. Firstly we declare a RecyclerView in our XML file and then get the reference of it in our Activity. After that we creates two ArrayList for Person Names and Images. After that we set a StaggeredGridLayoutManager and finally we set the Adapter to show the grid items in RecyclerView. Whenever a user clicks on an item the full size image will be displayed on the next screen.



Step 1: Create A New Project And Name It RecyclerViewExample.

Step 2: Open Gradle Scripts > build.gradle and add RecyclerView Library dependency in it.

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 24
    buildToolsVersion "24.0.1"
    defaultConfig {
        applicationId "abhiandroid.com.recyclerviewexample"
        minSdkVersion 16
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
```

```
        }
    }
}
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:24.1.1'
    compile "com.android.support:recyclerview-v7:23.4.0" // dependency file for RecyclerView
}
```

Step 3: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we create a RecyclerView in our XML file.

```
<?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"
    tools:context="abhiandroid.com.recyclerviewexample.MainActivity">
    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

Step 4: Create a new drawable XML file in Drawable folder and name it custom_item_layout.xml and add the following code in it for creating a custom grid item.

In this step we create a drawable XML file in which we add the code for creating custom grid items.

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <!--
       stroke with color and width for creating outer line
    -->
    <stroke
        android:width="1dp"
        android:color="#000" />
</shape>
```

Step 5: Create a new XML file rowlayout.xml for grid item of RecyclerView and paste the following code in it.

In this step we create a new xml file for item row in which we creates a TextView and ImageView to show the data in grid format.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/custom_item_layout"
    android:padding="5dp">
    <!--
    grid items for RecyclerView
    -->
    <ImageView
        android:id="@+id/image"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitXY"
        android:src="@mipmap/ic_launcher" />
    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:text="ABCD"
        android:textSize="20sp"
        android:textColor="#fff" />
</RelativeLayout>
```

Step 6 : Now open app -> java -> package -> MainActivity.java and add the below code.

In this step firstly we get the reference of RecyclerView. After that we creates two ArrayList's for Person Names and Images. After that we set a StaggeredGridLayoutManager with horizontal orientation and then finally we set the Adapter to show the grid items in RecyclerView.

```
package abhiandroid.com.recyclerviewexample;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.StaggeredGridLayoutManager;
import java.util.ArrayList;
```

```
import java.util.Arrays;
public class MainActivity extends AppCompatActivity {
    // ArrayList for person names
    ArrayList personNames = new ArrayList<>(Arrays.asList("Person 1", "Person 2", "Person
3", "Person 4", "Person 5", "Person 6", "Person 7", "Person 8", "Person 9", "Person 10",
"Person 11", "Person 12", "Person 13", "Person 14"));
    ArrayList personImages = new ArrayList<>(Arrays.asList(R.drawable.person1,
R.drawable.person2, R.drawable.person3, R.drawable.person4, R.drawable.person5,
R.drawable.person6, R.drawable.person7, R.drawable.person1, R.drawable.person2,
R.drawable.person3, R.drawable.person4, R.drawable.person5, R.drawable.person6,
R.drawable.person7));
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // get the reference of RecyclerView
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
        // set a StaggeredGridLayoutManager with 3 number of columns and horizontal
orientation
        StaggeredGridLayoutManager staggeredGridLayoutManager = new
StaggeredGridLayoutManager(3, LinearLayoutManager.HORIZONTAL);
        recyclerView.setLayoutManager(staggeredGridLayoutManager); // set LayoutManager to
RecyclerView
        //  call the constructor of CustomAdapter to send the reference and data to Adapter
        CustomAdapter customAdapter = new CustomAdapter(MainActivity.this, personNames,
personImages);
        recyclerView.setAdapter(customAdapter); // set the Adapter to RecyclerView
    }
}
```

Step 7: Create a new class CustomAdapter.java inside package and add the following code.

In this step we create a CustomAdapter class and extends RecyclerView.Adapter class with ViewHolder in it. After that we implement the overrided methods and create a constructor for getting the data from Activity. In this custom Adapter two methods are more important first is onCreateViewHolder in which we inflate the layout item xml and pass it to View Holder and other is onBindViewHolder in which we set the data in the view's with the help of ViewHolder. Finally we implement the setOnClickListener event on itemview and on click of item we display the selected image in full size in the next Activity.

```
package abhiandroid.com.recyclerviewexample;
import android.content.Context;
import android.content.Intent;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
```

```
import android.widget.TextView;
import java.util.ArrayList;
public class CustomAdapter extends RecyclerView.Adapter {
    ArrayList personNames;
    ArrayList personImages;
    Context context;
    public CustomAdapter(Context context, ArrayList personNames, ArrayList personImages) {
        this.context = context;
        this.personNames = personNames;
        this.personImages = personImages;
    }
    @Override
    public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        // infalte the item Layout
        View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.rowlayout,
parent, false);
        // set the view's size, margins, paddings and layout parameters
        MyViewHolder vh = new MyViewHolder(v); // pass the view to View Holder
        return vh;
    }
    @Override
    public void onBindViewHolder(MyViewHolder holder, final int position) {
        // set the data in items
        holder.name.setText(personNames.get(position));
        holder.image.setImageResource(personImages.get(position));
        // implement setOnClickListener event on item view.
        holder.itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // open another activity on item click
                Intent intent = new Intent(context, SecondActivity.class);
                intent.putExtra("image", personImages.get(position)); // put image data in
Intent
                context.startActivity(intent); // start Intent
            }
        });
    }
    @Override
    public int getItemCount() {
        return personNames.size();
    }
    public class MyViewHolder extends RecyclerView.ViewHolder {
        // init the item view's
        TextView name;
        ImageView image;
        public MyViewHolder(View itemView) {
            super(itemView);
            // get the reference of item view's
            name = (TextView) itemView.findViewById(R.id.name);
            image = (ImageView) itemView.findViewById(R.id.image);
        }
    }
}
```

Step 8: Create a new XML file activity_second.xml and add below code in it.

In this step we create a ImageView in our XML file to show the selected image in full size.

```
<?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"
    android:background="#fff">
    <ImageView
        android:id="@+id/selectedImage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:scaleType="fitXY" />
</RelativeLayout>
```

Step 9: Create a new Activity and name it SecondActivity.class and add the below code in it.

In this step we get the reference of ImageView and then get Intent which was set from adapter of Previous Activity and then finally we set the image in ImageView.

```
package abhiandroid.com.recyclerviewexample;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.widget.ImageView;
public class SecondActivity extends AppCompatActivity {
    ImageView selectedImage;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        selectedImage = (ImageView) findViewById(R.id.selectedImage); // init a ImageView
        Intent intent = getIntent(); // get Intent which was set from adapter of Previous
Activity
        selectedImage.setImageResource(intent.getIntExtra("image", 0)); // get image from
Intent and set it in ImageView
    }
}
```

Output:

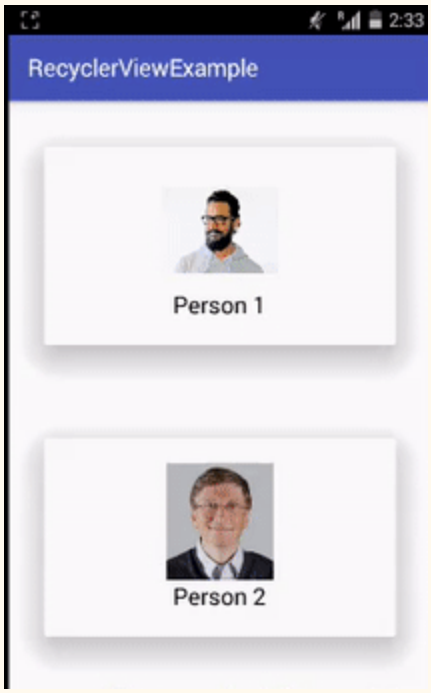
Now run the App and you will see person name in staggered grid which can be scrolled in horizontal direction created using RecyclerView.

CardView

In Android, CardView is another main element that can represent the information in a card manner with a drop shadow called elevation and corner radius which looks consistent across the platform. CardView was introduced in Material Design in API level 21 (Android 5.0 i.e Lollipop).

CardView uses elevation property on Lollipop for shadows and falls back to a custom emulated shadow implementation on older platforms.

This new widget is a big step for displaying data/information inside cards. We can easily design good looking UI when we combined CardView with RecyclerView. A CardView is a ViewGroup that can be added in our Activity or Fragment using a layout XML file.



Basic CardView XML code In Android Studio:

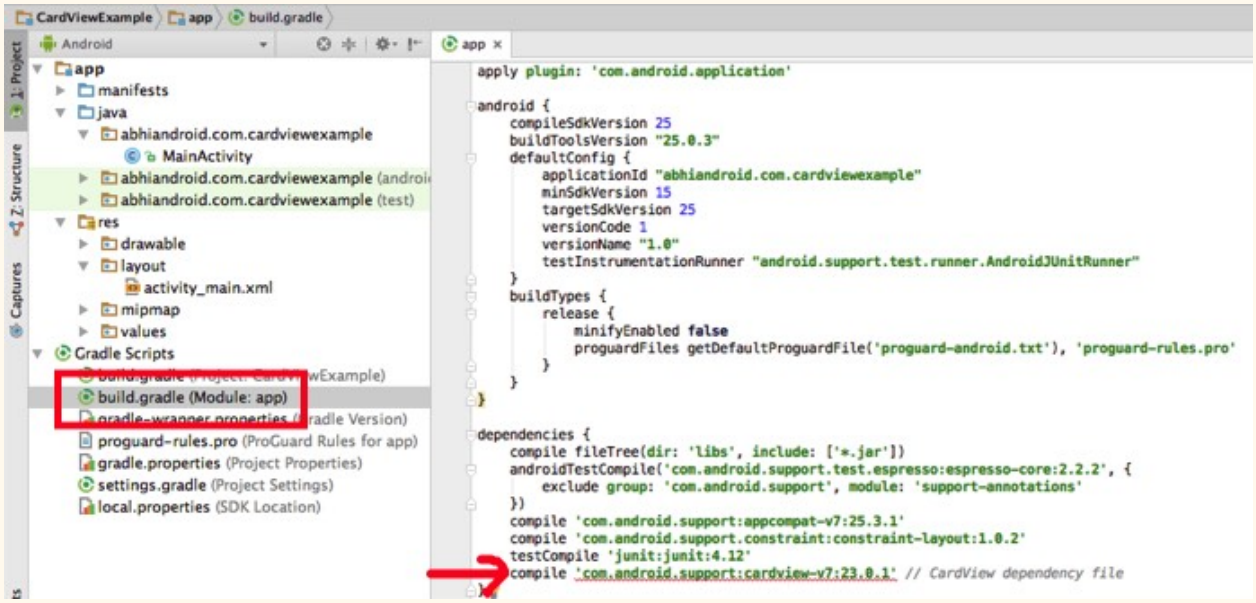
```
<android.support.v7.widget.CardView
xmlns:card_view="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="wrap_content">
</android.support.v7.widget.CardView>
```

Gradle Dependency to use CardView:

The CardView widget is a part of separate library valid for API 7 level or higher. Add the following dependency in your Gradle build file to use CardView.

Add inside Gradle Scripts > build.gradle (Module: app) and inside dependencies

```
dependencies {
compile 'com.android.support:cardview-v7:23.0.1'
}
```



Attributes of CardView In Android:

Now let's we discuss some common attributes of a CardView that helps us to configure it in our layout (xml).

1. card_view:cardBackgroundColor :

This attribute is used to set the Background color for the CardView. We can also set the background color programmatically using setCardBackgroundColor(int color) method.

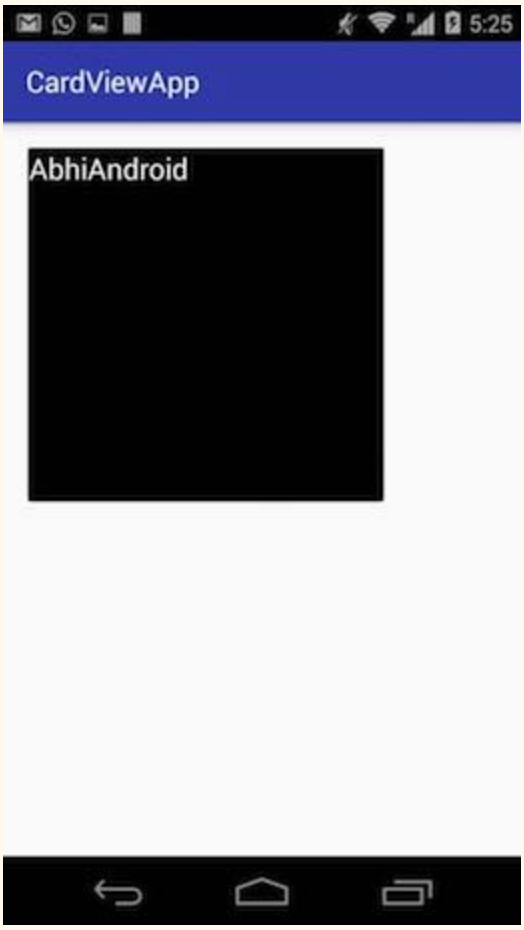
Below we set the black color in the background of CardView..

```
<android.support.v7.widget.CardView
        android:id="@+id/card_view"
        android:layout_width="250dp"
        android:layout_height="250dp"
        android:layout_gravity="center"
        card_view:cardBackgroundColor="#000">

        <TextView
            android:layout_width="match_parent"
```

```
            android:layout_height="match_parent"
            android:text="AbhiAndroid"
            android:textColor="#fff"
            android:textSize="20sp" />

    </android.support.v7.widget.CardView>
```



2. card view:cardCornerRadius:

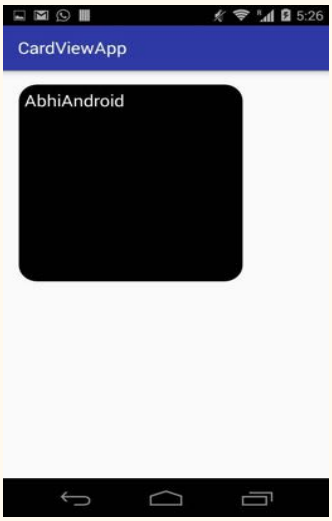
This attribute is used to set the corner radius for the CardView. We can also set the corner radius programmatically using setRadius(float radius) method.

Below we set the black background and corner radius for CardView.

```
<android.support.v7.widget.CardView
        android:id="@+id/card_view"
        android:layout_width="250dp"
        android:layout_height="250dp"
        android:layout_gravity="center"
        card_view:cardCornerRadius="20dp"
        card_view:cardBackgroundColor="#000"><!--
    corner radius value 20dp and black background
    -->

        <TextView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="AbhiAndroid"
            android:textColor="#fff"
            android:textSize="20sp" />

    </android.support.v7.widget.CardView>
```



3. card_view:cardElevation

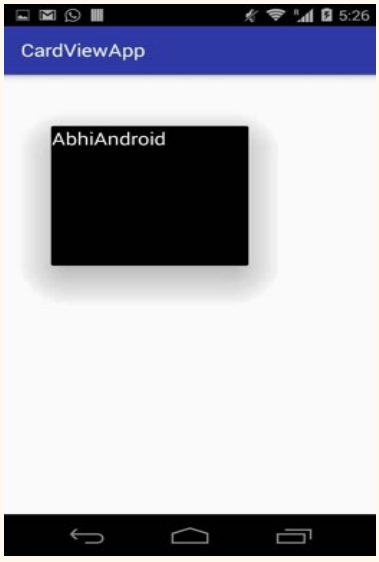
This attribute is used to set the elevation for the CardView. Elevation is used to show the shadow of the CardView. We can also set the card elevation value programmatically using setCardElevation(float elevation) method.

Below we set the black background and elevation value for the CardView.

```
<android.support.v7.widget.CardView
        android:id="@+id/card_view"
        android:layout_width="250dp"
        android:layout_height="250dp"
        android:layout_gravity="center"
        card_view:cardElevation="30dp"
        card_view:cardBackgroundColor="#000"><!--
    card elevation value 30dp and black background
    -->

        <TextView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="AbhiAndroid"
            android:textColor="#fff"
            android:textSize="20sp" />

    </android.support.v7.widget.CardView>
```



4. card_view:contentPadding :

This attribute is used to set the inner padding between the edges of the Card and children of the CardView. This attribute is used to set the equally padding between all the edges of the card and children of the CardView. We can also set the padding from bottom, top, left and right edges. We can also set padding programmatically using setContentPadding(int left, int top, int right, int bottom) method.

Below we set the black background and 20dp value for the inner padding between the edges of the Card and children of the CardView.

```
<android.support.v7.widget.CardView
        android:id="@+id/card_view"
        android:layout_width="250dp"
        android:layout_height="250dp"
        android:layout_gravity="center"
        card_view:cardBackgroundColor="#000"
        card_view:cardCornerRadius="20dp"
        card_view:cardElevation="30dp"
        card_view:contentPadding="20dp"><!--
    content padding value 20dp and black background
    -->

        <TextView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="AbhiAndroid"
            android:textColor="#fff"
            android:textSize="20sp" />

    </android.support.v7.widget.CardView>
```

5. cardview:contentPaddingBottom :

This attribute is used to set the inner padding between the bottom edge of the card and children of the CardView.

Below we set the black background and 20dp value for the inner padding between the bottom edge of the Card and children of the CardView.

```
<android.support.v7.widget.CardView
        android:id="@+id/card_view"
        android:layout_width="250dp"
        android:layout_height="250dp"
        android:layout_gravity="center"
        card_view:cardBackgroundColor="#000"
        card_view:cardElevation="10dp"
        card_view:contentPaddingBottom="20dp"><!--
    content bottom padding value 20dp and black background
    -->
        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentBottom="true"
                android:text="AbhiAndroid"
                android:textColor="#fff"
                android:textSize="20sp" />
        </RelativeLayout>

    </android.support.v7.widget.CardView>
```

6. cardview:contentPaddingLeft :

This attribute is used to set the inner padding between the left edge of the card and children of the CardView.

Below we set the black background and 20dp value for the inner padding between the left edge of the Card and children of the CardView.

```
<android.support.v7.widget.CardView
        android:id="@+id/card_view"
        android:layout_width="250dp"
        android:layout_height="250dp"
        android:layout_gravity="center"
        card_view:cardBackgroundColor="#000"
        card_view:cardElevation="10dp"
        card_view:contentPaddingLeft="20dp"><!--
    content left padding value 20dp and black background
    -->
        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
```

```
                android:text="AbhiAndroid"
                android:textColor="#fff"
                android:textSize="20sp" />
        </RelativeLayout>

    </android.support.v7.widget.CardView>
```

7. cardview:contentPaddingRight :

This attribute is used to set the inner padding between the right edge of the card and children of the CardView.

Below we set the black background and 20dp value for the inner padding between the right edge of the Card and children of the CardView.

```
<android.support.v7.widget.CardView
        android:id="@+id/card_view"
        android:layout_width="250dp"
        android:layout_height="250dp"
        android:layout_gravity="center"
        card_view:cardBackgroundColor="#000"
        card_view:cardElevation="10dp"
        card_view:contentPaddingRight="20dp"><!--
    content right padding value 20dp and black background
    -->
        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentRight="true"
                android:text="AbhiAndroid"
                android:textColor="#fff"
                android:textSize="20sp" />
        </RelativeLayout>

    </android.support.v7.widget.CardView>
```

8. cardview:contentPaddingTop :

This attribute is used to set the inner padding between the top edge of the card and children of the CardView.

Below we set the black background and 20dp value for the inner padding between the top edge of the Card and children of the CardView.

```
<android.support.v7.widget.CardView
        android:id="@+id/card_view"
        android:layout_width="250dp"
        android:layout_height="250dp"
        android:layout_gravity="center"
        card_view:cardBackgroundColor="#000"
        card_view:cardElevation="10dp"
        card_view:contentPaddingTop="20dp"><!--
    content left padding value 20dp and black background
    -->
        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="AbhiAndroid"
                android:textColor="#fff"
                android:textSize="20sp" />
        </RelativeLayout>

    </android.support.v7.widget.CardView>
```

Important Methods Of CardView:

Let's we discuss some important methods of CardView that may be called in order to manage the CardView.

1. setCardBackgroundColor(int color) : This method is used to set the background color for the CardView.

Below we set the background color for the CardView.

```
// init the CardView
        CardView cardView = (CardView) findViewById(R.id.card_view);
```

```
        // set black color for the Background of the CardView
        cardView.setCardBackgroundColor(Color.BLACK);
```

2. setCardElevation(float): This method is used to set the backward compatible elevation of the CardView. This method sets the value in float type format.

Below we set the backward compatible elevation value of the CardView.

```
// init the CardView
        CardView cardView = (CardView) findViewById(R.id.card_view);
        // set black color for the Background of the CardView
        cardView.setCardBackgroundColor(Color.BLACK);
        cardView.setCardElevation(20f); // backward compatible elevation value
```

3. getCardElevation() : This method is used to get the elevation value which we set using setCardElevation(float) method. This method return a float type value.

Below we firstly set the elevation value and then get the same of CardView.

```
// init the CardView
        CardView cardView = (CardView) findViewById(R.id.card_view);
        // set black color for the Background of the CardView
        cardView.setCardBackgroundColor(Color.BLACK);
        cardView.setCardElevation(20f); // set backward compatible elevation value
        float elevationValue=cardView.getCardElevation(); // get the card background
compatible elevation value
```

4. setRadius(float radius) : This method is used to set the corner radius value of the CardView. This method sets the value in float type format.

Below we set the corner radius value of CardView.

```
// init the CardView
        CardView cardView = (CardView) findViewById(R.id.card_view);
        // set black color for the Background of the CardView
        cardView.setCardBackgroundColor(Color.BLACK);
        cardView.setRadius(20f); // sets corner radius value
```

5. getRadius() : This method is used to get the corner radius value that we set using setRadius(float radius) method. This method return a float type value.

Below we firstly set the corner radius value and then get the same of CardView.

```
// init the CardView
        CardView cardView = (CardView) findViewById(R.id.card_view);
        // set black color for the Background of the CardView
        cardView.setCardBackgroundColor(Color.BLACK);
        cardView.setRadius(20f); // sets corner radius value
        float radiusValue=cardView.getRadius(); // get the corner radius value
```

6. setContentPadding(int left, int top, int right, int bottom) : This method is used to set the padding betweeen the card's edges and the children of the CardView. This method sets the int type value for the content padding. This method set the equally padding between all the edges of the card and children of the CardVie

Below we set the value of padding between the card's edges and the children of the CardView.

```
// init the CardView
        CardView cardView = (CardView) findViewById(R.id.card_view);
        // set black color for the Background of the CardView
        cardView.setCardBackgroundColor(Color.BLACK);
        cardView.setContentPadding(10,10,10,10); // set the padding between the card's edges
and the children of the CardView
```

7. getContentPaddingBottom() : This method is used to get the inner padding before the Card's bottom edge. This method returns an int type value.

Below we firstly set the content padding and then get the inner padding before the Card's bottom edge.

```
// init the CardView
        CardView cardView = (CardView) findViewById(R.id.card_view);
        // set black color for the Background of the CardView
```

```
        cardView.setCardBackgroundColor(Color.BLACK);
        cardView.setContentPadding(10, 10, 10, 10); // set the padding between the card's
edges and the children of the CardView
        int bottomPadding = cardView.getPaddingBottom(); // get the inner padding before the
Card's bottom edge.
```

8. getContentPaddingLeft() : This method is used to get the inner padding after the Card's left edge. This method returns an int type value.

Below we firstly set the content padding and then get the inner padding after the Card's left edge.

```
// init the CardView
        CardView cardView = (CardView) findViewById(R.id.card_view);
        // set black color for the Background of the CardView
        cardView.setCardBackgroundColor(Color.BLACK);
        cardView.setContentPadding(10, 10, 10, 10); // set the padding between the card's
edges and the children of the CardView
        int leftPadding = cardView.getPaddingLeft(); // get the inner padding after the
Card's left edge.
```

9. getContentPaddingRight() : This method is used to get the inner padding before the Card's right edge. This method returns an int type value.

Below we firstly set the content padding and then get the inner padding before the Card's right edge.

```
 // init the CardView
        CardView cardView = (CardView) findViewById(R.id.card_view);
        // set black color for the Background of the CardView
        cardView.setCardBackgroundColor(Color.BLACK);
        cardView.setContentPadding(10, 10, 10, 10); // set the padding between the card's
edges and the children of the CardView
        int rightPadding = cardView.getPaddingLeft(); // get the inner padding before the
Card's right edge.
```

10. getContentPaddingTop() : This method is used to get the inner padding after the Card's top edge. This method returns an int type value.

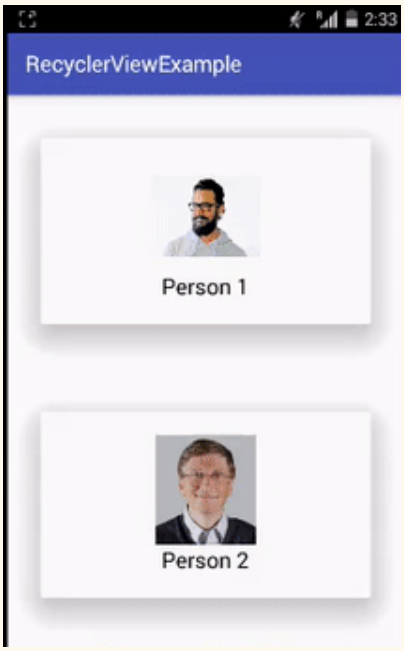
Below we firstly set the content padding and then get the inner padding after the Card's top edge.

```
// init the CardView
        CardView cardView = (CardView) findViewById(R.id.card_view);
        // set black color for the Background of the CardView
        cardView.setCardBackgroundColor(Color.BLACK);
        cardView.setContentPadding(10, 10, 10, 10); // set the padding between the card's
edges and the children of the CardView
        int topPadding = cardView.getPaddingLeft(); // get the inner padding after the
Card's top edge.
```

CardView Example Using RecyclerView as GridView in Android Studio:

Below is the example of RecyclerView As GridView in which we display list of Person Names with their images with default vertical orientation by using RecyclerView. In this example we are using LinearLayoutManager with default vertical orientation to display the items. Firstly we declare a RecyclerView in our XML file and then get the reference of it in our Activity. After that we creates two ArrayList's for Person Names and Images. After that we set a LinearLayoutManager and finally we set the Adapter to show the list items in RecyclerView. Whenever a user clicks on an item the full size image will be displayed on the next screen.

Below you can see final output and step by step explanation of the example:



Step 1: Create a New Project And Name It RecyclerViewExample.

Step 2: Open Gradle Scripts > build.gradle (Module: app) and add RecyclerView & CardView Library dependency in it.

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 24
    buildToolsVersion "24.0.1"
    defaultConfig {
        applicationId "abhiandroid.com.recyclerviewexample"
        minSdkVersion 16
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}
```

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:24.1.1'
    compile "com.android.support:recyclerview-v7:23.0.1" // dependency file for RecyclerView
    compile 'com.android.support:cardview-v7:23.0.1' // CardView dependency file

}
```

Step 3: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we create a RecyclerView in our XML file.

```
<?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"
    tools:context="abhiandroid.com.recyclerviewexample.MainActivity">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

Step 4: Create a new XML file rowlayout.xml for list item of RecyclerView and paste the following code in it.

In this step we create a new xml file for item row in which we creates a TextView and ImageView to show the data in grid format. In this we define the views inside CardView to display the items in the form of cards.

```
<android.support.v7.widget.CardView android:id="@+id/card_view"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    card_view:cardElevation="20dp"
    android:layout_margin="10dp"
    card_view:contentPadding="20dp"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:android="http://schemas.android.com/apk/res/android">

        <RelativeLayout
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center">

            <ImageView
                android:id="@+id/image"
                android:layout_width="100dp"
                android:layout_height="100dp"
                android:layout_centerHorizontal="true"
                android:src="@drawable/person1" />


            <TextView
                android:id="@+id/name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@+id/image"
                android:layout_centerHorizontal="true"
                android:linksClickable="true"
                android:text="http://abhiandroid.com"
                android:textColor="#000"
                android:textSize="20sp" />
        </RelativeLayout>

    </android.support.v7.widget.CardView>
```

Step 5 : Now open app -> java -> package -> MainActivity.java and add the below code.

In this step firstly we get the reference of RecyclerView. After that we creates two ArrayList‘s for Person Names and Images. After that we set a LinearLayoutManager and finally we set the Adapter to show the list items in RecyclerView.

```
package abhiandroid.com.recyclerviewexample;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.GridLayoutManager;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;

import java.util.ArrayList;
import java.util.Arrays;

public class MainActivity extends AppCompatActivity {

// ArrayList for person names
ArrayList personNames = new ArrayList<>(Arrays.asList("Person 1", "Person 2", "Person 3",
"Person 4", "Person 5", "Person 6", "Person 7","Person 8", "Person 9", "Person 10", "Person
11", "Person 12", "Person 13", "Person 14"));
```

```
ArrayList personImages = new ArrayList<>(Arrays.asList(R.drawable.person1,
R.drawable.person2, R.drawable.person3, R.drawable.person4, R.drawable.person5,
R.drawable.person6, R.drawable.person7,R.drawable.person1, R.drawable.person2,
R.drawable.person3, R.drawable.person4, R.drawable.person5, R.drawable.person6,
R.drawable.person7));

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
// get the reference of RecyclerView
RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
// set a LinearLayoutManager with default vertical orientaion
LinearLayoutManager linearLayoutManager = new LinearLayoutManager(getApplicationContext());
recyclerView.setLayoutManager(linearLayoutManager); // set LayoutManager to RecyclerView
// call the constructor of CustomAdapter to send the reference and data to Adapter
CustomAdapter customAdapter = new CustomAdapter(MainActivity.this,
personNames,personImages);
recyclerView.setAdapter(customAdapter); // set the Adapter to RecyclerView
}
}
```

Step 6: Create a new class CustomAdapter.java inside package and add the following code.

In this step we create a CustomAdapter class and extends RecyclerView.Adapter class with Viewholder in it. After that we implement the overrided methods and create a constructor for getting the data from Activity.

In this custom Adapter two methods are important: First is onCreateViewHolder in which we inflate the layout item xml and pass it to View Holder and other is onBindViewHolder in which we set the data in the view's with the help of ViewHolder. Finally we implement the setOnClickListener event on itemview and on click of item we display the selected image in full size in the next Activity.

```
package abhiandroid.com.recyclerviewexample;
import android.content.Context;
import android.content.Intent;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

import java.util.ArrayList;

public class CustomAdapter extends RecyclerView.Adapter {
```

```
ArrayList personNames;
ArrayList personImages;
Context context;

public CustomAdapter(Context context, ArrayList personNames, ArrayList personImages) {
this.context = context;
this.personNames = personNames;
this.personImages = personImages;
}

@Override
public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
// infalte the item Layout
View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.rowlayout, parent,
false);
// set the view's size, margins, paddings and layout parameters
MyViewHolder vh = new MyViewHolder(v); // pass the view to View Holder
return vh;
}

@Override
public void onBindViewHolder(MyViewHolder holder, final int position) {
// set the data in items
holder.name.setText(personNames.get(position));
holder.image.setImageResource(personImages.get(position));
// implement setOnClickListener event on item view.
holder.itemView.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
// open another activity on item click
Intent intent = new Intent(context, SecondActivity.class);
intent.putExtra("image", personImages.get(position)); // put image data in Intent
context.startActivity(intent); // start Intent
}
});

}

@Override
public int getItemCount() {
return personNames.size();
}

public class MyViewHolder extends RecyclerView.ViewHolder {
// init the item view's
TextView name;
ImageView image;

public MyViewHolder(View itemView) {
super(itemView);

// get the reference of item view's
name = (TextView) itemView.findViewById(R.id.name);
image = (ImageView) itemView.findViewById(R.id.image);

}
}
}
```

Step 7: Create a new XML file activity_second.xml and add below code in it.

In this step we create a ImageView in our XML file to show the selected image in full size.

```
<?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"
    android:background="#fff">

    <ImageView
        android:id="@+id/selectedImage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:scaleType="fitXY" />
</RelativeLayout>
```

Step 8: Create a new Activity and name it SecondActivity.class and add the below code in it.

In this step we get the reference of ImageView and then get Intent which was set from adapter of Previous Activity and then finally we set the image in ImageView.

```
package abhiandroid.com.recyclerviewexample;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.widget.ImageView;

public class SecondActivity extends AppCompatActivity {

ImageView selectedImage;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_second);
selectedImage = (ImageView) findViewById(R.id.selectedImage); // init a ImageView
Intent intent = getIntent(); // get Intent which we set from Previous Activity
selectedImage.setImageResource(intent.getIntExtra("image", 0)); // get image from Intent and
set it in ImageView
}
}
```

Output:

Now run the App and you can see person names with their images with default vertical orientation in Card style. Click on any image and it will open in large size.

CardView Example 2 in Android Studio:

Below is the example of CardView in which we display the data/information in a Card. In this example we use ImageView and TextView inside a card. Whenever a user click on the card a message “CardView clicked event ” is displayed on the screen with the help of Toast.

Below you can see final output and step by step explanation of the example:

Step 1: Create a new project and name it CardViewExample.

Step 2: Open Gradle Scripts > build.gradle and add CardView Library dependency in it.

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 25
    buildToolsVersion "25.0.3"
    defaultConfig {
        applicationId "abhiandroid.com.cardviewexample"
        minSdkVersion 15
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:25.3.1'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:cardview-v7:23.0.1' // CardView dependency file
}
```

Step 3: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we create a ImageView and a TextView inside our CardView. We also use elevation, background and other attributes of CardView.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <android.support.v7.widget.CardView
        android:id="@+id/card_view"
        android:layout_width="250dp"
        android:layout_height="250dp"
        android:layout_gravity="center"
        card_view:cardBackgroundColor="#000"
        card_view:cardElevation="20dp"
```

```
        card_view:contentPadding="20dp">

        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center">

            <ImageView
                android:id="@+id/imageView"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerHorizontal="true"
                android:src="@drawable/abhiandroid" />


            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@+id/imageView"
                android:layout_centerHorizontal="true"
                android:linksClickable="true"
                android:text="http://abhiandroid.com"
                android:textColor="#fff"
                android:textSize="20sp" />
        </RelativeLayout>

    </android.support.v7.widget.CardView>

</LinearLayout>
```

Step 4 : Now open app -> java -> package -> MainActivity.java and add the below code.

In this step firstly we get the reference of CardView and then set the corner radius value.Finally we implement the onClickListener event on CardView so that if a user clicks on CardView a message “CardView clicked event” is displayd on the screen by using a Toast.

```
package abhiandroid.com.cardviewexample;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.CardView;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // init the CardView
        CardView cardView = (CardView) findViewById(R.id.card_view);
        cardView.setRadius(20F); // set corner radius value
        // Implement onClickListener event on CardView
        cardView.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                Toast.makeText(MainActivity.this, "CardView clicked event ",
Toast.LENGTH_LONG).show();
            }
        });

    }
}
```

Output:

Now run the App and you will see simple CardView which will display the information.

Premium Android App Source Code

List of premium Android App source code on sale at a very affordable price:

1. **Convert Website Into Advance Android App Source Code** - Do you have app for your website? If NO, then use my Ultimate WebView App source code and convert your website into an advance Android App in just 15 minutes. The App code comes with 20+ advance features, clean code and build in Android Studio. Some of the key features are Firebase push notification, customized navigation menu, ActionBar, Admob, progressbar, offline handling, video support and lots more.
2. **Smart News Android App Source Code** - This is web-admin based Android App source code. Using this source code you can create news, blog or information type App whose content will be dynamically updated directly from the server with ease. This App code will have 20+ advance features, clean code and build in Android Studio.

If you have any query or question in mind, please email to info@abhiandroid.com and I will get back to you within 24 hours(mostly ASAP).

Thanks!

Thank you for reading. I hope you like it…

For more Android tutorials please like our Facebook page and stay in touch with AbhiAndroid.com!