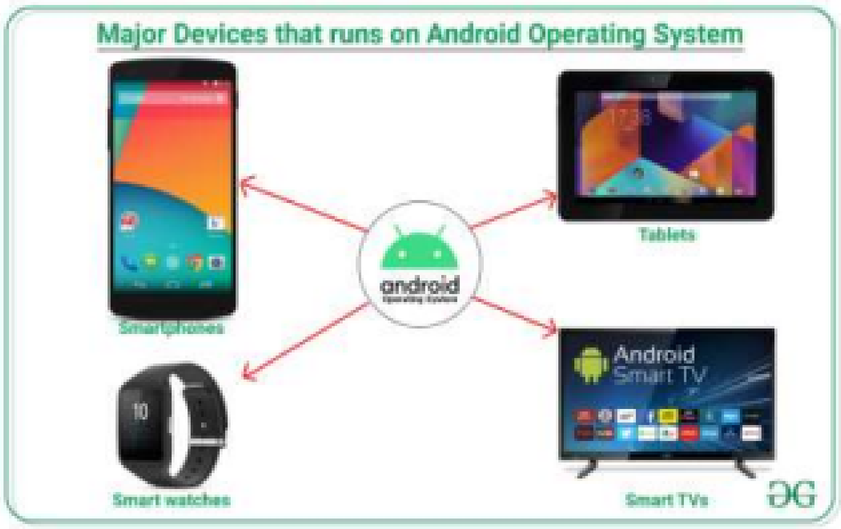
Unit-1: Project structure of Mobile Application:

1.1 Internal details of Android Application:

 Android operating system is the largest installed base among various mobile platforms across the globe. Hundreds of millions of mobile devices are powered by Android in more than 190 countries of the world.

 It conquered around 71% of the global market share by the end of 2021, and this trend is growing bigger every other day.

 The company named Open Handset Alliance developed Android for the first time that is based on the modified version of the Linux kernel and other open-source software.

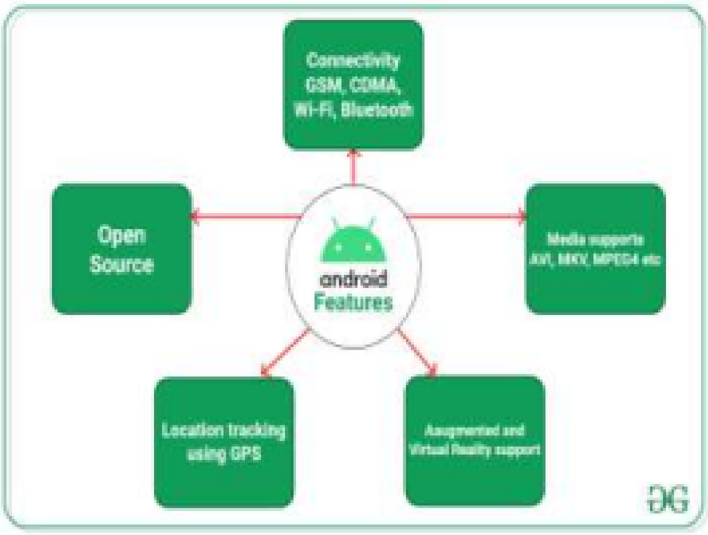


 Google sponsored the project at initial stages and in the year 2005, it acquired the whole company. In September 2008, the first Android-powered device was launched in the market.   Android dominates the mobile OS industry because of the long list of features it provides. It's user friendly, has huge community support, provides a greater extent of customization, and a large number of companies build Android-compatible smartphones.

 As a result, the market observes a sharp increase in the demand for developing Android mobile applications, and with that companies need smart developers with the right skill set.   At first, the purpose of Android was thought of as a mobile operating system. However, with the advancement of code libraries and its popularity among developers of the divergent domain, Android becomes an absolute set of software for all devices like tablets, wearables, set-top boxes, smart TVs, notebooks, etc

History of Android

1) Initially, **Andy Rubin** founded Android Incorporation in Palo Alto, California, United States in October, 2003.
2) In *17th August 2005*, **Google** acquired android Incorporation. Since then, it is in the subsidiary of Google Incorporation.
3) The key employees of Android Incorporation are Andy Rubin, Rich Miner, Chris White and Nick Sears.
4) Originally intended for camera but shifted to smart phones later because of low market for camera only.
5) Android is the nick name of Andy Rubin given by coworkers because of his love to robots.
6) In 2007, Google announces the development of android OS.
7) In 2008, HTC launched the first android mobile.


Connectivity GSM, CDMA, Wi-Fi, Bluetooth
Open Source
android Features
Media supports AVI, MKV, MPEG4 etc
Location tracking using GPS
Augmented and Virtual Reality support

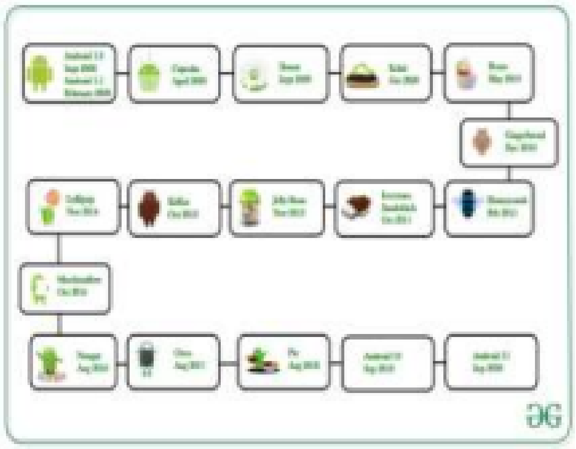
Features of Android

Android is a powerful open-source operating system that open-source provides immense features and some of these are listed below.

 Android Open Source Project so we can customize the OS based on our requirements. Android supports different types of connectivity for GSM, CDMA, Wi-Fi, Bluetooth, etc. for telephonic conversation or data transfer.
 Using wi-fi technology we can pair with other devices while playing games or using other applications.
 It contains multiple APIs to support location-tracking services such as GPS. We can manage all data storage-related activities by using the file manager. It contains a wide range of media supports like AVI, MKV, FLV, MPEG4, etc. to play or record a variety of audio/video.
 It also supports different image formats like JPEG, PNG, GIF, BMP, MP3, etc. It supports multimedia hardware control to perform playback or recording using a camera and microphone.
 Android has an integrated open-source WebKit layout-based web browser to support User Interfaces like HTML5, and CSS3.
 Android supports multi-tasking means we can run multiple applications at a time and can switch between them.
 It provides support for virtual reality or 2D/3D Graphics.

Android Versions

Google first publicly announced Android in November 2007 but was released on 23 SEPTEMBER 2008 to be exact. The first device to bring Android into the market was the HTC Dream with the version Android 1.0. Since then, Google released a lot of android versions such as Apple Pie, Banana Bread, Cupcake, Donut, Éclair, Froyo, Gingerbread, Jellybeans, Kitkat, Lollipop, marshmallow, Nougat, Oreo, etc. with extra functionalities and new features.



The following table shows the version details of android, which is released by Google from 2007 to date.

Programming Languages used in Developing Android Applications

- Java
- Kotlin

Developing the Android Application using Kotlin is preferred by Google, as Kotlin is made an official language for Android Development, which is developed and maintained by JetBrains. Previously before Java is considered the official language for Android Development. Kotlin is made official for Android Development in Google I/O 2017.

Advantages of Android Development

1. The Android is an open-source Operating system and hence possesses a vast community for support.
2. The design of the Android Application has guidelines from Google, which becomes easier for developers to produce more intuitive user applications.

3. Fragmentation gives more power to Android Applications. This means the application can run two activities on a single screen.
4. Releasing the Android application in the Google play store is easier when it is compared to other platforms.

Disadvantages of Android Development

1. Fragmentation provides a very intuitive approach to user experience but it has some drawbacks, where the development team needs time to adjust to the various screen sizes of mobile smartphones that are now available in the market and invoke the particular features in the application.
2. The Android devices might vary broadly. So the testing of the application becomes more difficult. 3. As the development and testing consume more time, the cost of the application may increase, depending on the application's complexity and features.

Android Fundamentals

1. Android Programming Languages

In Android, basically, programming is done in two languages JAVA or C++ and XML(Extension Markup Language). Nowadays KOTLIN is also preferred. The XML file deals with the design, presentation, layouts, blueprint, etc (as a front-end) while the JAVA or KOTLIN deals with the working of buttons, variables, storing, etc (as a back-end).

2. Android Components

The App components are the building blocks of Android. Each component has its own role and life cycles i.e from launching of an app till the end. Some of these components depend upon others also. Each component has a definite purpose. The four major app components are:

- Activities
- Services
- Broadcast Receivers:
- Content Provider

Activities: It deals with the UI and the user interactions to the screen. In other words, it is a User Interface that contains activities. These can be one or more depending upon the App. It starts when the application is launched. At least one activity is always present which is known as MainActivity. The activity is implemented through the following.

Syntax:

```
public class MainActivity extends Activity{
 // processes

}
```

Services: Services are the background actions performed by the app, these might be long-running operations like a user playing music while surfing the Internet. A service might need other sub-services so as to perform specific tasks. The main purpose of the Services is to provide non-stop working of the app without breaking any interaction with the user.

Syntax:

```
public class MyServices extends Services{
 // code for the services
```

}

Broadcast Receivers: A Broadcast is used to respond to messages from other applications or from the System. For example, when the battery of the phone is low, then the Android OS fires a Broadcasting message to launch the Battery Saver function or app, after receiving the message the appropriate action is taken by the app. Broadcast Receiver is the subclass of BroadcastReceiver class and each object is represented by Intent objects.

Syntax:

```
public class MyReceiver extends BroadcastReceiver{

 public void onReceive(context,intent){

 }
```

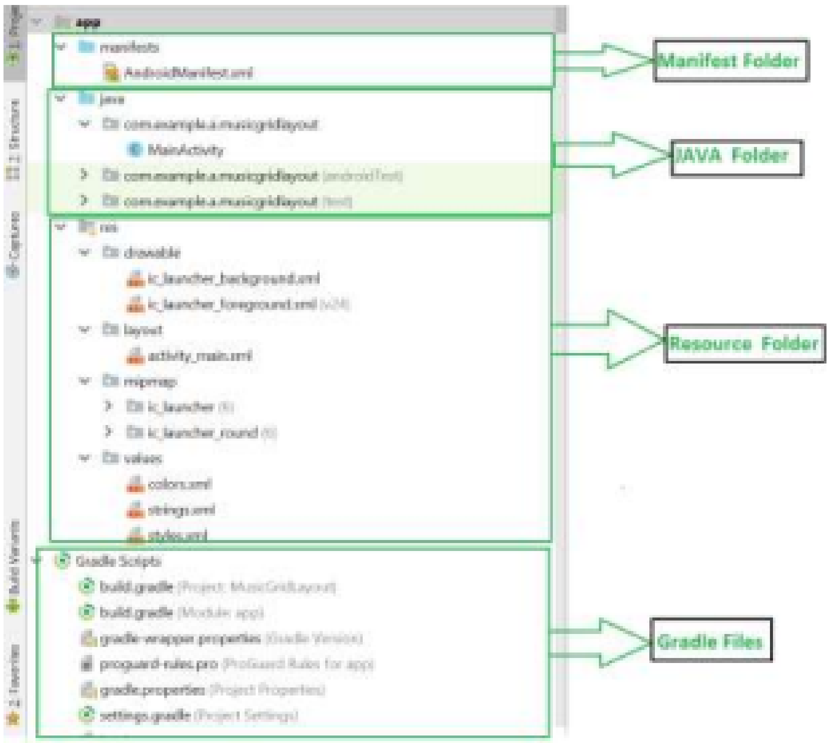
Content Provider: Content Provider is used to transferring the data from one application to the others at the request of the other application. These are handled by the class ContentResolver class. This class implements a set of APIs(Application Programming Interface) that enables the other applications to perform the transactions. Any Content Provider must implement the Parent Class of ContentProvider class.

Syntax:

```
public class MyContentProvider extends ContentProvider{
 public void onCreate()

 {}

}
```

Structural Layout Of Android Studio

The basic structural layout of Android Studio is given below:


app
manifests
AndroidManifest.xml
java
com.example.a.musicgridlayout
MainActivity
com.example.a.musicgridlayout (androidTest)
com.example.a.musicgridlayout (test)
res
drawable
ic_launcher_background.xml
ic_launcher_foreground.xml (v24)
layout
activity_main.xml
mipmap
ic_launcher (6)
ic_launcher_round (6)
values
colors.xml
strings.xml
styles.xml
Gradle Scripts
build.gradle (Project: MusicGridLayout)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
Manifest Folder
JAVA Folder
Resource Folder
Gradle Files

The above figure represents the various structure of an app.

Manifest Folder: Android Manifest is an XML file that is the root of the project source set. It describes the essential information about the app and the Android build tools, the Android Operating System, and Google Play. It contains the permission that an app might need in order to perform a specific task. It also contains the Hardware and the Software features of the app, which determines the compatibility of an app on the Play Store. It also includes special activities like services, broadcast receiver, content providers, package name, etc.

Java Folder: The JAVA folder consists of the java files that are required to perform the background task of the app. It consists of the functionality of the buttons, calculation, storing, variables, toast (small popup message), programming function, etc. The number of these files depends upon the type of activities created.

Resource Folder: The res or Resource folder consists of the various resources that are used in the app. This consists of sub-folders like drawable, layout, mipmap, raw, and values. The drawable consists of the images. The layout consists of the XML files that define the user interface layout. These are stored in res.layout and are accessed as R.layout class. The raw consists of the Resources files like audio files or music files, etc. These are accessed through R.raw.filename. values are used to store the hardcoded strings(considered safe to store string values) values, integers, and colors. It consists of various other directories like:

 R.array :arrays.xml for resource arrays
 R.integer : integers.xml for resource integers
 R.bool : bools.xml for resource boolean
 R.color :colors.xml for color values
 R.string : strings.xml for string values
 R.dimen : dimens.xml for dimension values
 R.style : styles.xml for styles

Gradle Files: Gradle is an advanced toolkit, which is used to manage the build process that allows defining the flexible custom build configurations. Each build configuration can define its own set of code and resources while reusing the parts common to all versions of your app. The Android plugin for Gradle works with the build toolkit to provide processes and configurable settings that are specific to building and testing Android applications. Gradle and the Android plugin run independently of Android Studio. This means that you can build your Android apps from within Android Studio. The flexibility of the Android build system enables you to perform custom build configurations without modifying your app’s core source files.

Basic Layout can be defined in a tree structure as:

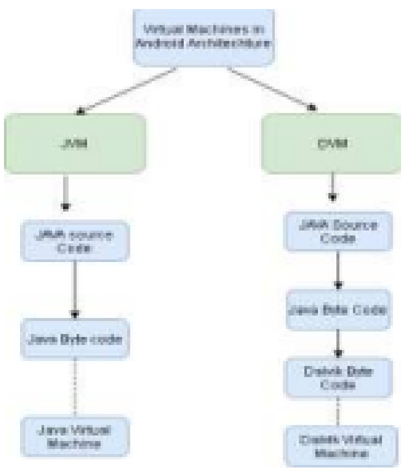
```
Project/
 app/
 manifest/
 AndroidManifest.xml
 java/
 MyActivity.java
 res/
 drawable/
 icon.png
 background.png
 drawable-hdpi/
 icon.png
 background.png
 layout/
 activity_main.xml
 info.xml
 values/
```

strings.xml

1.1.1 Dalvik VM

Dalvik Virtual Machine is a Register-Based virtual machine. Dalvik is a name of a town in Iceland. It was designed and written by ***Dan Bornstein*** with contributions of other Google engineers as part of the Android mobile phone platform. The Dalvik virtual machine was named after Bornstein after the fishing village "Dalvík" in Eyjafjörður, Iceland, where some of his ancestors used to live.

Working of DVM



The Java Compiler (javac) converts the Java Source Code into Java Byte-Code(.class). Then DEX Compiler converts this (.class) file into in Dalvik Byte Code i.e. ".dex" file.

Application: For Android, a new Virtual machine was developed by Google as stated above. It uses registers of the CPU to store the operands. So that no requirement of any pushing and popping of instructions. Hence it makes execution faster. The instructions operate on virtual registers, being those virtual registers memory positions in the host device. Register-based models are good at optimizing and running on low memory. They can store common sub expression results which can be used again in the future. This is not possible in a Stack-based model at all. Dalvik Virtual Machine uses its own byte-code and runs ".dex" (Dalvik Executable File) file.

Advantages

 DVM supports the Android operating system only.
 In DVM executable is APK.
 Execution is faster.
 From Android 2.2 SDK Dalvik has its own JIT (Just In Time) compiler.
 DVM has been designed so that a device can run multiple instances of the Virtual Machine effectively.
 Applications are given their own instances.

Disadvantages

 DVM supports only Android Operating System.
 For DVM very few Re-Tools are available.
 Requires more instructions than register machines to implement the same high-level code.
App Installation takes more time due to dex.
 More internal storage is required.

Android Screen Orientation Example

The *screenOrientation* is the attribute of activity element. The orientation of android activity can be portrait, landscape, sensor, unspecified etc. You need to define it in the AndroidManifest.xml file.

Syntax:

```
<activity android:name="package_name.Your_ActivityName"
 android:screenOrientation="orirntation_type">
```

```
  </activity>
```

Example:

```
 <activity android:name="
example.javatpoint.com.screenorientation.MainActivity"
android:screenOrientation="portrait">
 </activity>
```

OR

```
 <activity android:name=".SecondActivity"
  android:screenOrientation="landscape">
 </activity>
```

The common values for screenOrientation attribute are as follows:

Value	**Description**
unspecified	It is the default value. In such case, system chooses the orientation.
portrait	taller not wider
landscape	wider not taller
sensor	Orientation is determined by the device orientation sensor.

1.1.2 AndroidMenifest, R.java

AndroidManifest.xml file in android

The **AndroidManifest.xml file** *contains information of your package*, including components of the application such as activities, services, broadcast receivers, content providers etc. It performs some other tasks also:

- It is **responsible to protect the application** to access any protected parts by providing the permissions.
- It also **declares the android api** that the application is going to use.
- It **lists the instrumentation classes**. The instrumentation classes provide profiling and other information. This information is removed just before the application is published etc. This is the required xml file for all the android application and located inside the root directory. **A simple AndroidManifest.xml file looks like this:**

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools">

 <application
 android:allowBackup="true"
 android:dataExtractionRules="@xml/data_extraction_rules"
 android:fullBackupContent="@xml/backup_rules"
 android:icon="@mipmap/ic_launcher"
 android:label="@string/app_name"
 android:roundIcon="@mipmap/ic_launcher_round"
 android:supportsRtl="true"
 android:theme="@style/Theme.CustomCheckBox"
 tools:targetApi="31">
 <activity
 android:name=".MainActivity"
 android:exported="true">
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />

 <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
 </activity>
 </application>

</manifest>
```

Elements of the AndroidManifest.xml file

The elements used in the above xml file are described below.

<manifest>

manifest is the root element of the AndroidManifest.xml file. It has package attribute that describes the package name of the activity class.

<application>

application is the subelement of the manifest. It includes the namespace declaration. This element contains several subelements that declares the application component such as activity etc.

The commonly used attributes are of this element are icon, label, theme etc.

android:icon represents the icon for all the android application components.
android:label works as the default label for all the application components.
android:theme represents a common theme for all the android activities.

<activity>

activity is the subelement of application and represents an activity that must be defined in the AndroidManifest.xml file. It has many attributes such as label, name, theme, launchMode etc.
android:label represents a label i.e. displayed on the screen.
android:name represents a name for the activity class. It is required attribute.

<intent-filter>

intent-filter is the sub-element of activity that describes the type of intent to which activity, service or broadcast receiver can respond to.

<action>
It adds an action for the intent-filter. The intent-filter must have at least one action element.

<category>
It adds a category name to an intent-filter.

Android R.java file
Being an auto-generated file that is generated by AAPT (Android Asset Packaging Tool), Android R.java contains resource IDs for all the resources of res/ directory. The id for the created component is automatically generated in the R.java whenever a component is created in the android activity_main.xml file. The life cycle methods for an activity such as onCreate, onStop, OnResume, etc is provided by the Activity java class. The created ID can later be used in the Java Source file. To act on a component, the corresponding id can be used in the activity source file. The android creates the R.jar file automatically in case the R.jar file is deleted. The android R.java file contains many static nested classes such as menu, id, layout, attr, drawable, string, etc. Now we will see the code of the android R.java file. [Note: If you delete R.jar file, android creates it automatically.]

1.2 Android Widgets (UI)

There are given a lot of android widgets with simplified examples such as Button, EditText, AutoCompleteTextView, ToggleButton, DatePicker, TimePicker, ProgressBar etc. Android widgets are easy to learn. The widely used android widgets with examples are given below:

Android Button: Android Button represents a push-button. The android.widget.Button is subclass of TextView class and CompoundButton is the subclass of Button class.
There are different types of buttons in android such as RadioButton, ToggleButton, CompoundButton etc.

Android Toast: Displays information for the short duration of time.

Custom Toast: We are able to customize the toast, such as we can display image on the toast

ToggleButton: It has two states ON/OFF.

CheckBox: Android CheckBox is a type of two state button either checked or unchecked. There can be a lot of usage of checkboxes. For example, it can be used to know the hobby of the user, activate/deactivate the specific action etc.
Android CheckBox class is the subclass of CompoundButton class.

AlertDialog: AlertDialog displays a alert dialog containing the message with OK and Cancel buttons.

Spinner: Spinner displays the multiple options, but only one can be selected at a time.

AutoCompleteTextView : Android AutoCompleteTextView completes the word based on the reserved words, so no need to write all the characters of the word. Android AutoCompleteTextView is a editable text field, it displays a list of suggestions in a drop down menu from which user can select only one suggestion or value. Android AutoCompleteTextView is the subclass of EditText class. The MultiAutoCompleteTextView is the subclass of AutoCompleteTextView class.

RatingBar: RatingBar displays the rating bar.

DatePicker: Datepicker displays the datepicker dialog that can be used to pick the date.

TimePicker: TimePicker displays the timepicker dialog that can be used to pick the time.

ProgressBar: ProgressBar displays progress task.

1.2.1 Default and Custom Checkbox

Android CheckBox is a type of two state button either checked or unchecked.

There can be a lot of usage of checkboxes. For example, it can be used to know the hobby of the user, activate/deactivate the specific action etc.

Android CheckBox class is the subclass of CompoundButton class.

Android CheckBox class

The android.widget.CheckBox class provides the facility of creating the CheckBoxes.

Methods of CheckBox class

There are many inherited methods of View, TextView, and Button classes in the CheckBox class. Some of them are as follows:

Method	**Description**
public boolean isChecked()	Returns true if it is checked otherwise false.
public void setChecked(boolean status)	Changes the state of the CheckBox.

Android CheckBox Example

activity_main.xml

Drag the three checkboxes and one button for the layout. Now the activity_main.xml file will look like this:

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
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"
 tools:context="example.javatpoint.com.checkbox.MainActivity">


 <CheckBox
 android:id="@+id/checkBox"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginLeft="144dp"
 android:layout_marginTop="68dp"
 android:text="Pizza"
 app:layout_constraintStart_toStartOf="parent"
 app:layout_constraintTop_toTopOf="parent" />

 <CheckBox
 android:id="@+id/checkBox2"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginLeft="144dp"
 android:layout_marginTop="28dp"
 android:text="Coffee"
 app:layout_constraintStart_toStartOf="parent"
 app:layout_constraintTop_toBottomOf="@+id/checkBox" />

 <CheckBox
 android:id="@+id/checkBox3"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginLeft="144dp"
 android:layout_marginTop="28dp"
 android:text="Burger"
 app:layout_constraintStart_toStartOf="parent"
 app:layout_constraintTop_toBottomOf="@+id/checkBox2" />

 <Button
 android:id="@+id/button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginLeft="144dp"
 android:layout_marginTop="184dp"
 android:text="Order"
 app:layout_constraintStart_toStartOf="parent"
 app:layout_constraintTop_toBottomOf="@+id/checkBox3" />

</android.support.constraint.ConstraintLayout>
```

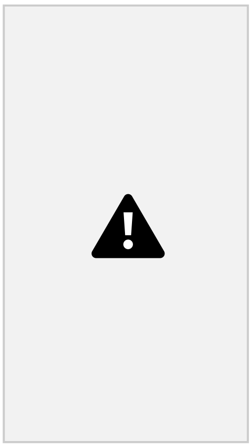
Activity class

Let's write the code to check which toggle button is ON/OFF.

File: MainActivity.java

```
package example.javatpoint.com.checkbox;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
 CheckBox pizza,coffe,burger;
 Button buttonOrder;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
addListenerOnButtonClick();
}
public void addListenerOnButtonClick(){
//Getting instance of CheckBoxes and Button from the activty_main.xml file
pizza=(CheckBox)findViewById(R.id.checkBox);
coffe=(CheckBox)findViewById(R.id.checkBox2);
burger=(CheckBox)findViewById(R.id.checkBox3);
buttonOrder=(Button)findViewById(R.id.button);

//Applying the Listener on the Button click
buttonOrder.setOnClickListener(new View.OnClickListener(){

@Override
public void onClick(View view) {
int totalamount=0;
StringBuilder result=new StringBuilder();
result.append("Selected Items:");
if(pizza.isChecked()){
result.append("\nPizza 100Rs");
totalamount+=100;
}
if(coffe.isChecked()){
result.append("\nCoffe 50Rs");
totalamount+=50;
}
if(burger.isChecked()){
result.append("\nBurger 120Rs");
totalamount+=120;
}
result.append("\nTotal: "+totalamount+"Rs");
//Displaying the message on the toast
Toast.makeText(getApplicationContext(), result.toString(), Toast.LENGTH_LONG).show();  }

});
}
}
```



Android Custom CheckBox

Android provides facility to customize the UI of view elements rather than default. You are able to create custom CheckBox in android. So, you can add some different images of checkbox on the layout.

Example of Custom CheckBox

In this example, we create both default as well as custom checkbox. Add the following code in activity_main.xml file.

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
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"
 tools:context="example.javatpoint.com.customcheckbox.MainActivity">


 <TextView
 android:id="@+id/textView1"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:gravity="center_horizontal"
 android:textSize="25dp"
 android:text="Default Check Box"
 android:layout_alignParentTop="true"
 android:layout_alignParentLeft="true"
 android:layout_alignParentStart="true" />

 <CheckBox
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="New CheckBox"
 android:id="@+id/checkBox"
 android:layout_below="@+id/textView1"
 android:layout_centerHorizontal="true"
 android:layout_marginTop="46dp" />

 <CheckBox
 android:layout_width="wrap_content"
  android:layout_height="wrap_content"
 android:text="New CheckBox"
 android:id="@+id/checkBox2"
 android:layout_below="@+id/checkBox"
 android:layout_alignLeft="@+id/checkBox"
 android:layout_alignStart="@+id/checkBox" />

 <View
 android:layout_width="fill_parent"
 android:layout_height="1dp"
 android:layout_marginTop="200dp"
 android:background="#B8B894"
 android:id="@+id/viewStub" />

 <CheckBox
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="CheckBox 1"
android:id="@+id/checkBox3"
android:button="@drawable/customcheckbox"
android:layout_below="@+id/viewStub"
android:layout_centerHorizontal="true"
android:layout_marginTop="58dp" />
<CheckBox
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="CheckBox 2"
android:id="@+id/checkBox4"
android:button="@drawable/customcheckbox"
android:layout_below="@+id/checkBox3"
android:layout_alignLeft="@+id/checkBox3"
android:layout_alignStart="@+id/checkBox3" />
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceSmall"
android:textSize="25dp"
android:text="Custom Check Box"
android:id="@+id/textView"
android:layout_alignTop="@+id/viewStub"
android:layout_centerHorizontal="true" />
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Show Checked"
android:id="@+id/button"
android:layout_alignParentBottom="true"
android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Now implement a selector in another file (checkbox.xml) under drawable folder which customizes the checkbox.

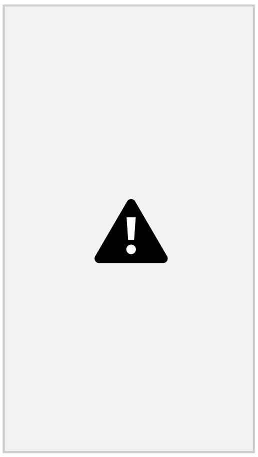
File: checkbox.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
 <item android:state_checked="true" android:drawable="@drawable/checked" />
<item android:state_checked="false" android:drawable="@drawable/unchecked"/>
</selector>
```

File: MainActivity.java

```
package example.javatpoint.com.customcheckbox;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Toast;
```

```
public class MainActivity extends AppCompatActivity {
 CheckBox cb1,cb2;
 Button button;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 cb1=(CheckBox)findViewById(R.id.checkBox3);
 cb2=(CheckBox)findViewById(R.id.checkBox4);
 button=(Button)findViewById(R.id.button);

 button.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View v) {
 StringBuilder sb=new StringBuilder("");

 if(cb1.isChecked()){
 String s1=cb1.getText().toString();
 sb.append(s1);
 }

 if(cb2.isChecked()){
 String s2=cb2.getText().toString();
 sb.append("\n"+s2);

 }
 if(sb!=null && !sb.toString().equals("")){
 Toast.makeText(getApplicationContext(), sb, Toast.LENGTH_LONG).show();

 }
 else{
 Toast.makeText(getApplicationContext(),"Nothing Selected",
Toast.LENGTH_LONG).show();
 }

 }

 });
 }
}
```



1.2.2 Dynamic and Custom RadioButton

RadioButton is a two states button which is either checked or unchecked. If a single radio button is unchecked, we can click it to make checked radio button. Once a radio button is checked, it cannot be marked as unchecked by user.

RadioButton is generally used with RadioGroup. RadioGroup contains several radio buttons, marking one radio button as checked makes all other radio buttons as unchecked.

Example of Radio Button
In this example, we are going to implement single radio button separately as well as radio button in RadioGroup.
activity_main.xml
File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
 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:orientation="vertical"
 tools:context="example.javatpoint.com.radiobutton.MainActivity">

 <TextView
 android:id="@+id/textView1"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:layout_marginTop="30dp"
 android:gravity="center_horizontal"
 android:textSize="22dp"
 android:text="Single Radio Buttons" />

<!-- Default RadioButtons -->
 <RadioButton
 android:id="@+id/radioButton1"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:layout_gravity="center_horizontal"
 android:text="Radio Button 1"
 android:layout_marginTop="20dp"

 android:textSize="20dp" />
 <RadioButton
 android:id="@+id/radioButton2"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Radio Button 2"
 android:layout_marginTop="10dp"
 android:textSize="20dp" />
 <View
 android:layout_width="fill_parent"
 android:layout_height="1dp"
 android:layout_marginTop="20dp"
 android:background="#B8B894" />

 <TextView
```

```
android:id="@+id/textView2"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:layout_marginTop="30dp"
 android:gravity="center_horizontal"
 android:textSize="22dp"
 android:text="Radio button inside RadioGroup" />
 <!-- Customized RadioButtons -->
 <RadioGroup
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:id="@+id/radioGroup">

 <RadioButton
 android:id="@+id/radioMale"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text=" Male"
 android:layout_marginTop="10dp"
 android:checked="false"
 android:textSize="20dp" />

 <RadioButton
 android:id="@+id/radioFemale"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text=" Female"
 android:layout_marginTop="20dp"
 android:checked="false"
 android:textSize="20dp" />
 </RadioGroup>

 <Button
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Show Selected"
 android:id="@+id/button"
 android:onClick="onclickbuttonMethod"
 android:layout_gravity="center_horizontal" />
</LinearLayout>
```

Activity class
File: MainActivity.java

```
package example.javatpoint.com.radiobutton;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
 Button button;
 RadioButton genderradioButton;
 RadioGroup radioGroup;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 radioGroup=(RadioGroup)findViewById(R.id.radioGroup);
 }
 public void onclickbuttonMethod(View v){
 int selectedId = radioGroup.getCheckedRadioButtonId();
 genderradioButton = (RadioButton) findViewById(selectedId);
 if(selectedId==-1){
 Toast.makeText(MainActivity.this,"Nothing selected", Toast.LENGTH_SHORT).show();  }
 else{
 Toast.makeText(MainActivity.this,genderradioButton.getText(),
Toast.LENGTH_SHORT).show();
 }

 }
}
```

Output

Android Custom RadioButton

Rather than default user interface of android RadioButton, we can also implement a custom radio button. Custom RadioButton makes user interface more attractive.

Example of Custom RadioButton

Let's see an example of custom RadioButton.

activity_main.xml

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
 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:orientation="vertical"
 tools:context="com.example.test.customradiobutton.MainActivity">
```

```
<TextView
android:id="@+id/tv"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_marginTop="30dp"
android:gravity="center_horizontal"
android:textSize="25dp"
android:text="Customized Radio Buttons" />
 <!-- Customized RadioButtons -->
<RadioGroup
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/radioGroup">
<RadioButton
android:id="@+id/radioMale"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text=" Male"
android:layout_marginTop="10dp"
android:checked="false"
android:button="@drawable/custom_radio_button"
android:textSize="20dp" />

<RadioButton
android:id="@+id/radioFemale"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text=" Female"
android:layout_marginTop="20dp"
android:checked="false"
android:button="@drawable/custom_radio_button"
android:textSize="20dp" />
</RadioGroup>
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Show Selected"
android:id="@+id/button"
android:onClick="onclickbuttonMethod"
android:layout_gravity="center_horizontal" />
</LinearLayout>
```

custom_radio_button.xml

Now implement a selector in another file (custom_radio_button.xml) in drawable and place two different checked and unchecked button images.

File: checkbox.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
 <item android:state_checked="true" android:drawable="@drawable/checkedradiobutton" />
<item android:state_checked="false" android:drawable="@drawable/unchekedradiobutton" />
</selector>
```

Activity class

File: MainActivity.java

```
package com.example.test.customradiobutton;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
 Button button;
 RadioButton genderradioButton;
 RadioGroup radioGroup;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);

 radioGroup=(RadioGroup)findViewById(R.id.radioGroup);
 }
 public void onclickbuttonMethod(View v){
 int selectedId = radioGroup.getCheckedRadioButtonId();
 genderradioButton = (RadioButton) findViewById(selectedId);
 if(selectedId==-1){
 Toast.makeText(MainActivity.this,"Nothing selected", Toast.LENGTH_SHORT).show();  }
 else{
 Toast.makeText(MainActivity.this,genderradioButton.getText(), Toast.LENGTH_SHORT).show();  }
 }
}
```

OUTPUT:

1.2.3 Spinner, AlterDialog

❖ **Android AlertDialog Example**

 Android AlertDialog can be used to display the dialog message with OK and Cancel buttons. It can be used to interrupt and ask the user about his/her choice to continue or discontinue. Android AlertDialog is composed of three regions: title, content area and action buttons.
 Android AlertDialog is the subclass of Dialog class.

Methods of AlertDialog class

Method	**Description**
public AlertDialog.Builder setTitle(CharSequence)	This method is used to set the title of AlertDialog.
public AlertDialog.Builder setMessage(CharSequence)	This method is used to set the message for AlertDialog.
public AlertDialog.Builder setIcon(int)	This method is used to set the icon over AlertDialog.

Android AlertDialog Example
Let's see a simple example of android alert dialog.
You can have multiple components, here we are having only a textview.
File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
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"
 tools:context="example.javatpoint.com.alertdialog.MainActivity">
 <Button
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:id="@+id/button"
 android:text="Close app"
 app:layout_constraintBottom_toBottomOf="parent"
 app:layout_constraintLeft_toLeftOf="parent"
 app:layout_constraintRight_toRightOf="parent"
 app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

strings.xml
Optionally, you can store the dialog message and title in the strings.xml file.
File: strings.xml

```
<resources>
 <string name="app_name">AlertDialog</string>
 <string name="dialog_message">Welcome to Alert Dialog</string>
 <string name="dialog_title">Javatpoint Alert Dialog</string>
</resources>
```

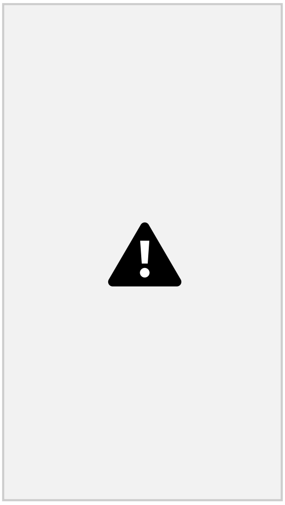
Activity class

Let's write the code to create and show the AlertDialog.

File: MainActivity.java

```
package example.javatpoint.com.alertdialog;
import android.content.DialogInterface;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.app.AlertDialog;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
 Button closeButton;
 AlertDialog.Builder builder;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
```

```
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 closeButton = (Button) findViewById(R.id.button);
 builder = new AlertDialog.Builder(this);
 closeButton.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View v) {
 //Uncomment the below code to Set the message and title from the strings.xml file
builder.setMessage(R.string.dialog_message) .setTitle(R.string.dialog_title);

 //Setting message manually and performing action on button click
 builder.setMessage("Do you want to close this application ?")
 .setCancelable(false)

 .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
 public void onClick(DialogInterface dialog, int id) {
 finish();
                         Toast.makeText(getApplicationContext(),"you choose yes action for alertbox",
                           Toast.LENGTH_SHORT).show();
 }
 })

 .setNegativeButton("No", new DialogInterface.OnClickListener() {
 public void onClick(DialogInterface dialog, int id) {
 // Action for 'NO' Button
                           dialog.cancel();
  Toast.makeText(getApplicationContext(),"you choose no action for alertbox",
Toast.LENGTH_SHORT).show();
 }
 });
 //Creating dialog box
 AlertDialog alert = builder.create();
 //Setting the title manually
 alert.setTitle("AlertDialogExample");
 alert.show();
 }
 });
 }
}
```



❖ Android Spinner Example

- Android Spinner is like the combox box of AWT or Swing. It can be used to display the multiple options to the user in which only one item can be selected by the user.
- Android spinner is like the drop down menu with multiple values from which the end user can select only one value.
- Android spinner is associated with AdapterView. So you need to use one of the adapter classes with spinner.
- Android Spinner class is the subclass of AsbSpinner class.

Android Spinner Example

In this example, we are going to display the country list. You need to use ArrayAdapter class to store the country list.

Let's see the simple example of spinner in android.

activity_main.xml

Drag the Spinner from the pallete, now the activity_main.xml file will like this:

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
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"
 tools:context="example.javatpoint.com.spinner.MainActivity">

 <Spinner
 android:id="@+id/spinner"
 android:layout_width="149dp"
 android:layout_height="40dp"
 android:layout_marginBottom="8dp"
 android:layout_marginEnd="8dp"
 android:layout_marginStart="8dp"
 android:layout_marginTop="8dp"
 app:layout_constraintBottom_toBottomOf="parent"
 app:layout_constraintEnd_toEndOf="parent"
 app:layout_constraintHorizontal_bias="0.502"
 app:layout_constraintStart_toStartOf="parent"
 app:layout_constraintTop_toTopOf="parent"
 app:layout_constraintVertical_bias="0.498" />

</android.support.constraint.ConstraintLayout>
```

Activity class

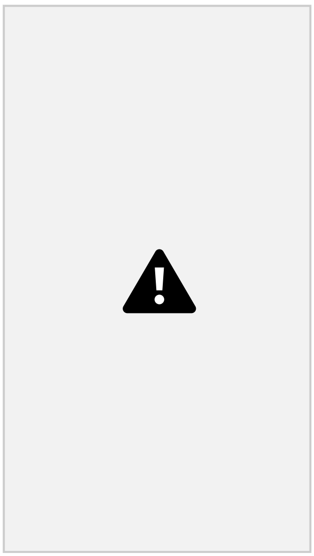
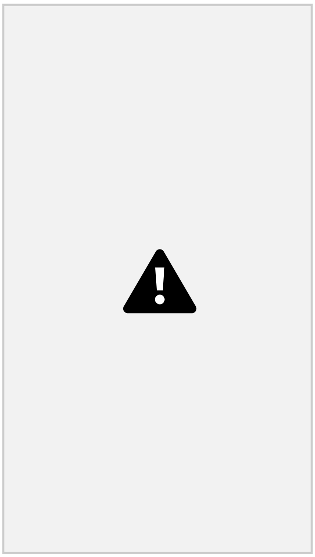
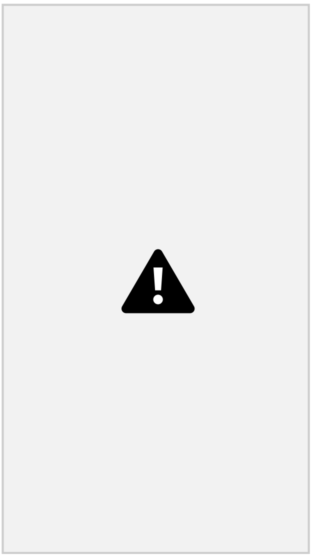
Let's write the code to display item on the spinner and perform event handling.

File: MainActivity.java

```
package example.javatpoint.com.spinner;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
```

```
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity implements
 AdapterView.OnItemSelectedListener
 {
 String[] country = { "India", "USA", "China", "Japan", "Other"};

 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 //Getting the instance of Spinner and applying OnItemSelectedListener on it
Spinner spin = (Spinner) findViewById(R.id.spinner);
 spin.setOnItemSelectedListener(this);

 //Creating the ArrayAdapter instance having the country list
 ArrayAdapter aa = new ArrayAdapter(this,android.R.layout.simple_spinner_item,country);
aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
 //Setting the ArrayAdapter data on the Spinner
 spin.setAdapter(aa);

 }

 //Performing action onItemSelected and onNothing selected
 @Override
 public void onItemSelected(AdapterView<?> arg0, View arg1, int position, long id) {
Toast.makeText(getApplicationContext(),country[position] , Toast.LENGTH_LONG).show();  }
 @Override
 public void onNothingSelected(AdapterView<?> arg0) {
 // TODO Auto-generated method stub
 }
}
```

OUTPUT:

Unit-2: Basic Attributes and Events of Important Android Widgets(UI)

2.1 ListView, Custom ListView

❖ **Listview**

 In android, ListView is a ViewGroup that is used to display the list of scrollable of items in multiple rows and the list items are automatically inserted to the list using an adapter.

 Generally, the adapter pulls data from sources such as an array or database and converts each item into a result view and that's placed into the list.

 Following is the pictorial representation of listview in android applications.

Android Adapter

In android, Adapter will act as an intermediate between the data sources and adapter views such as ListView, Gridview to fill the data into adapter views. The adapter will hold the data and iterates through an items in data set and generate the views for each item in the list. Generally, in android we have a different types of adapters available to fetch the data from different data sources to fill the data into adapter views, those are

Adapter	**Description**
ArrayAdapter	It will expects an Array or List as input.
CurosrAdapter	It will accepts an instance of cursor as an input.
SimpleAdapter	It will accepts a static data defined in the resources.
BaseAdapter	It is a generic implementation for all three adapter types and it can be used for ListView, Gridview or Spinners based on our requirements

Android ListView Example

Following is the example of creating a ListView using arrayadapter in android application. Now open an activity_main.xml file from \res\layout path and write the code like as shown below --->

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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:orientation="vertical">
 <ListView
 android:id="@+id/userlist"
 android:layout_width="match_parent"
 android:layout_height="wrap_content" >
 </ListView>
</LinearLayout>
```

Once we are done with creation of layout, now we will bind data to our ListView using ArrayAdapter, for that open main activity file MainActivity.java from \java\com.tutlane.listview path and write the code like as shown below.

MainActivity.java

```
package com.tutlane.listview;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends AppCompatActivity {
 private ListView mListView;
 private ArrayAdapter aAdapter;
```

```
 private String[] users = { "Suresh Dasari", "Rohini Alavala", "Trishika Dasari", "Praveen Alavala",
"Madav Sai", "Hamsika Yemineni"};
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 mListView = (ListView) findViewById(R.id.userlist);
 aAdapter = new ArrayAdapter(this, android.R.layout.simple_list_item_1, users);
mListView.setAdapter(aAdapter);
 }
}
```

If you observe above code, we are binding static array (users) details to ListView using ArrayAdapter and calling our layout using setContentView method in the form of R.layout.layout_file_name. Here our xml file name is activity_main.xml so we used file name activity_main.

Generally, during the launch of our activity, onCreate() callback method will be called by the android framework to get the required layout for an activity.

Output of Android ListView Example

When we run above example using android virtual device (AVD) we will get a result like as shown below.

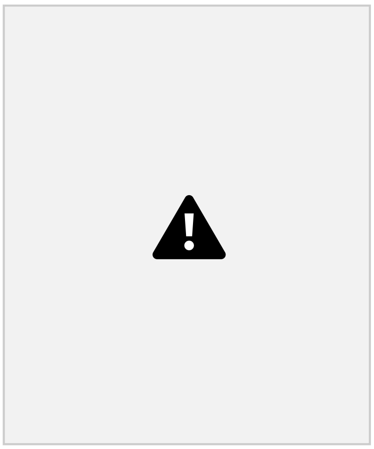
❖ **Android Custom ListView (Adding Images, sub-title)**

After creating simple ListView, android also provides facilities to customize our ListView. As the simple ListView, custom ListView also uses Adapter classes which added the content from data source (such as string array, array, database etc). Adapter bridges data between an AdapterViews and other Views

Example of Custom ListView

In this custom listview example, we are adding image, text with title and its sub-title.

Structure of custom listview project



activity_main.xml

Create an activity_main.xml file in layout folder.

File: activity_main.xml

```
 <?xml version="1.0" encoding="utf-8"?>
 <RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
```

```
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="com.example.test.listviewwithimage.MainActivity">

  <ListView
  android:id="@+id/list"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_marginBottom="50dp">
  </ListView>
 </RelativeLayout>
```

Create an additional mylist.xml file in layout folder which contains view components displayed in listview. **File: mylist.xml**

```
<?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="match_parent"
 android:orientation="horizontal" >

 <ImageView
 android:id="@+id/icon"
 android:layout_width="60dp"
 android:layout_height="60dp"
 android:padding="5dp" />

 <LinearLayout android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:orientation="vertical">

 <TextView
 android:id="@+id/title"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Medium Text"
 android:textStyle="bold"
 android:textAppearance="?android:attr/textAppearanceMedium"
 android:layout_marginLeft="10dp"
 android:layout_marginTop="5dp"
 android:padding="2dp"
 android:textColor="#4d4d4d" />
 <TextView
 android:id="@+id/subtitle"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="TextView"
 android:layout_marginLeft="10dp"/>
        </LinearLayout>
</LinearLayout>
```

Place the all required images in drawable folder.

Activity class

File: MainActivity.java

```
package com.example.test.listviewwithimage;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity
{
 ListView list;
 String[] maintitle ={
 "Title 1","Title 2",
 "Title 3","Title 4",
 "Title 5",
  };
 String[] subtitle ={
 "Sub Title 1","Sub Title 2",
 "Sub Title 3","Sub Title 4",
 "Sub Title 5",
 };
 Integer[] imgid={
 R.drawable.download_1,R.drawable.download_2,
 R.drawable.download_3,R.drawable.download_4,
 R.drawable.download_5,
 };
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);

 MyListAdapter adapter=new MyListAdapter(this, maintitle, subtitle,imgid);
list=(ListView)findViewById(R.id.list);
 list.setAdapter(adapter);

 list.setOnItemClickListener(new AdapterView.OnItemClickListener() {

 @Override
 public void onItemClick(AdapterView<?> parent, View view,int position, long id) {  //
TODO Auto-generated method stub
 if(position == 0) {
 //code specific to first list item
 Toast.makeText(getApplicationContext(),"Place Your First Option
Code",Toast.LENGTH_SHORT).show();
 }
 else if(position == 1) {
 //code specific to 2nd list item
 Toast.makeText(getApplicationContext(),"Place Your Second Option
Code",Toast.LENGTH_SHORT).show();
 }
 else if(position == 2) {

 Toast.makeText(getApplicationContext(),"Place Your Third Option
Code",Toast.LENGTH_SHORT).show();
 }
 else if(position == 3) {
```

```
 Toast.makeText(getApplicationContext(),"Place Your Forth Option
Code",Toast.LENGTH_SHORT).show();
 }
 else if(position == 4) {

 Toast.makeText(getApplicationContext(),"Place Your Fifth Option
Code",Toast.LENGTH_SHORT).show();
 }

 }
 });
 }
 }
```

Customize Our ListView

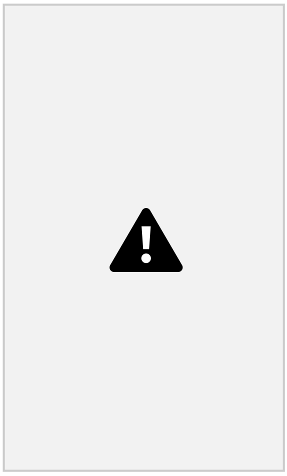
Create another java class MyListView.java which extends ArrayAdapter class. This class customizes our listview.

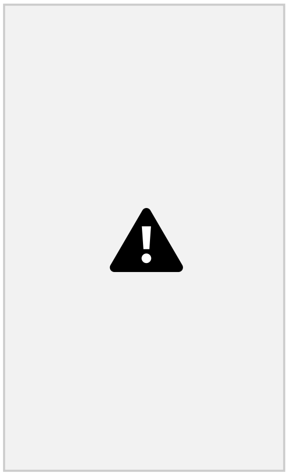
MyListView.java

```
package com.example.test.listviewwithimage;

import android.app.Activity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

public class MyListAdapter extends ArrayAdapter<String> {

 private final Activity context;
 private final String[] maintitle;
 private final String[] subtitle;
 private final Integer[] imgid;

 public MyListAdapter(Activity context, String[] maintitle,String[] subtitle, Integer[] imgid) {
super(context, R.layout.mylist, maintitle);
 // TODO Auto-generated constructor stub

 this.context=context;
 this.maintitle=maintitle;
 this.subtitle=subtitle;
 this.imgid=imgid;

 }
 public View getView(int position,View view,ViewGroup parent) {
 LayoutInflater inflater=context.getLayoutInflater();
 View rowView=inflater.inflate(R.layout.mylist, null,true);

 TextView titleText = (TextView) rowView.findViewById(R.id.title);
 ImageView imageView = (ImageView) rowView.findViewById(R.id.icon);
 TextView subtitleText = (TextView) rowView.findViewById(R.id.subtitle);

 titleText.setText(maintitle[position]);
 imageView.setImageResource(imgid[position]);
```

```
 subtitleText.setText(subtitle[position]);
 return rowView;
 };
}
```

OUTPUT:

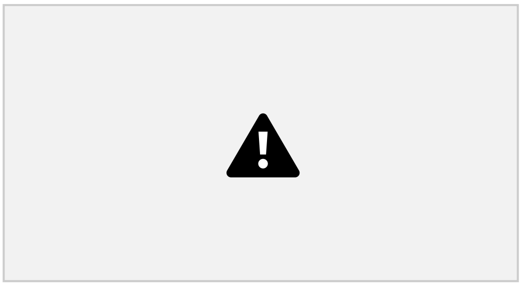




2.2 DatePicker, TimePicker, ProgressBar

 In android, DatePicker is a control that will allow users to select the date by a day, month and year in our application user interface.

 If we use DatePicker in our application, it will ensure that the users will select a valid date.
Following is the pictorial representation of using a datepicker control in android applications.



Generally, in android DatePicker available in two modes, one is to show the complete calendar and another one is to show the dates in spinner view.

Create Android DatePicker in XML Layout File

In android, we can create a DatePicker in XML layout file using <DatePicker> element with different attributes like as shown below

```
<DatePicker android:id="@+id/datePicker1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content" />
```

In anroid, the DatePicker supports a two types of modes, those are Calendar and Spinner to show the date details in our application.

Android DatePicker with Calendar Mode

We can define android DatePicker to show only a calendar view by using DatePicker android:datePickerMode attribute.

Following is the example of showing the DatePicker in Calendar mode.

```
<DatePicker
 android:id="@+id/datePicker1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:datePickerMode="calendar"/>
```

Android DatePicker with Spinner Mode
If we want to show the DatePicker in spinner format like showing day, month and year separately to select the date, then by using DatePicker android:datePickerMode attribute we can achieve this.
Following is the example of showing the DatePicker in Spinner mode.

```
 <DatePicker
android:id="@+id/datePicker1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:datePickerMode="spinner"/>
```

If you observe the above result we got the DatePicker in both Spinner and Calendar modes to select the date.
To get only spinner mode date selection, then we need to set android:calendarViewShown="false" attribute in DatePicker control like as shown below.

```
<DatePicker
 android:id="@+id/datePicker1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:datePickerMode="spinner"
 android:calendarViewShown="false"/>
```

The above code will return the DatePicker like as shown below:

If you observe the above result we got the DatePicker in spinner mode to select the date separately by day, month and year.
This is how we can use DatePicker in different modes based on our requirements in android applications.

Android DatePicker Control Attributes

The following are some of the commonly used attributes related to DatePicker control in android applications.

Attribute	**Description**
android:id	It is used to uniquely identify the control
android:datePickerMode	It is used to specify datepicker mode either spinner or calendar
android:background	It is used to set the background color for the date picker.
android:padding	It is used to set the padding for left, right, top or bottom of the date picker.

Android DatePicker Example

Following is the example of defining one DatePicker control, one TextView control and one Button control in RelativeLayout to show the selected date on Button click in the android application.

Create a new android application using android studio and give names as DatePickerExample. In case if you are not aware of creating an app in android studio check this article Android Hello World App. Now open an activity_main.xml file from \res\layout path and write the code like as shown below
activity_main.xml

```
<?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="match_parent">
<DatePicker
 android:id="@+id/datePicker1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_centerHorizontal="true"
 android:layout_marginTop="20dp" />
 <Button
 android:id="@+id/button1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_below="@+id/datePicker1"
 android:layout_marginLeft="100dp"
 android:text="Get Date" />
 <TextView
 android:id="@+id/textView1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_below="@+id/button1"
 android:layout_marginLeft="100dp"
 android:layout_marginTop="10dp"
 android:textStyle="bold"
 android:textSize="18dp"/>
</RelativeLayout>
```

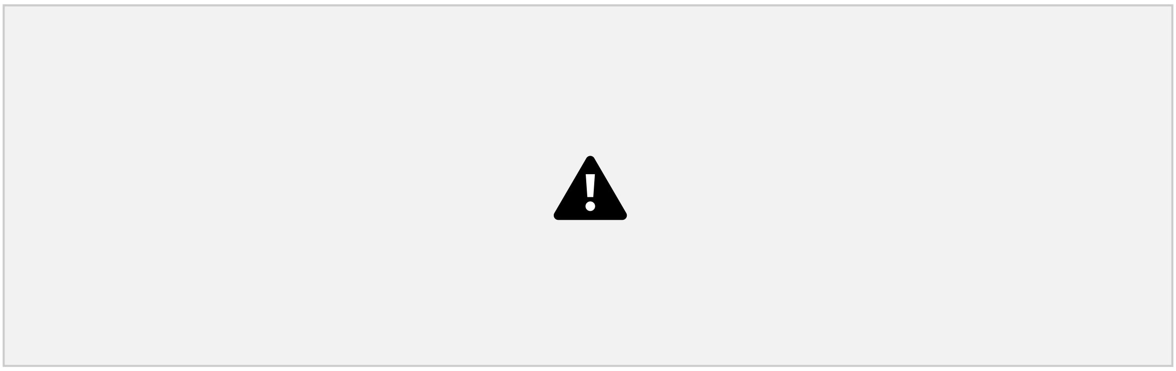
If you observe above code we created a one DatePicker control, one TextView control and one Button control in XML Layout file.
Once we are done with the creation of layout with required controls, we need to load the XML layout resource from our activity onCreate() callback method, for that open main activity file MainActivity.java from \java\com.tutlane.datepickerexample path and write the code like as shown below.
MainActivity.java

```
package com.tutlane.datepickerexample;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
 DatePicker picker;
 Button btnGet;
 TextView tvw;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
```

```
 tvw=(TextView)findViewById(R.id.textView1);
 picker=(DatePicker)findViewById(R.id.datePicker1);
 btnGet=(Button)findViewById(R.id.button1);
 btnGet.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View v) {
 tvw.setText("Selected Date: "+ picker.getDayOfMonth()+"/"+ (picker.getMonth() +
1)+"/"+picker.getYear());
 }
 });
 }
}
```

OUTPUT:

In android, TimePicker is a widget for selecting the time of day, in either 24-hour or AM/PM mode. If we use TimePicker in our application, it will ensure that the users will select a valid time for the day. Following is the pictorial representation of using a timepicker control in android applications.



Generally, in android TimePicker available in two modes, one is to show the time in **clock mode** and another one is to show the time in **spinner mode.**

Create Android DatePicker in XML Layout File

In android, we can create a TimePicker in XML layout file using <TimePicker> element with different attributes like as shown below

```
<TimePicker android:id="@+id/timePicker1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content" />
```

In anroid, the TimePicker supports two types of modes; those are Clock and Spinner to show the date details in our application.

Android TimePicker with Clock Mode

We can define android TimePicker to show time in clock format by using TimePicker android:timePickerMode attribute.

Following is the example of showing the TimePicker in Clock mode.

```
<TimePicker android:id="@+id/timePicker1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:timePickerMode="clock" />
```

The above code will return the TimePicker like as shown below.

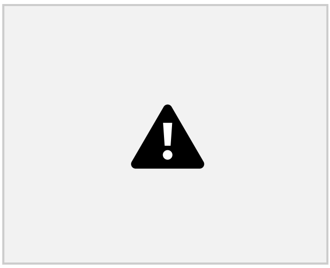
If you observe the above result we got the TimePicker in clock mode to select a time in Hours and Minutes based on our requirements.

Android TimePicker with Spinner Mode

If we want to show the TimePicker in spinner format like showing hours and minutes separately to select the time, then by using TimePicker android:timePickerMode attribute we can achieve this.

Following is the example of showing the TimePicker in spinner mode.

```
<TimePicker
 android:id="@+id/datePicker1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:timePickerMode="spinner"/>
```



If you observe the above result we got the TimePicker in spinner mode to select the time in Hours and Minutes.

We can change the TimePicker in spinner mode to **AM** / **PM** format instead of **24 Hours** format by using the **setIs24HourView(true)** method in Activity file like as shown below.

```
TimePicker picker=(TimePicker)findViewById(R.id.timePicker1);
picker.setIs24HourView(true);
```

The above code will return the TimePicker like as shown below

If you observe the above result, we got the TimePicker with AM / PM format in spinner mode to select the time separately by hours, minutes and AM/PM.

Android TimePicker Control Attributes

The following are some of the commonly used attributes related to **TimePicker** control in android applications.

Attribute	Description
android:id	It is used to uniquely identify the control
android:timePickerMode	It is used to specify timepicker mode, either spinner or clock
android:background	It is used to set the background color for the date picker.
android:padding	It is used to set the padding for left, right, top or bottom of the date picker.

Android TimePicker Example

Now open an activity_main.xml file from \res\layout path and write the code like as shown below

activity_main.xml

```
<?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="match_parent">
<TimePicker
```

```
android:id="@+id/timePicker1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:layout_marginTop="20dp" />
<Button
android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@+id/timePicker1"
android:layout_marginTop="10dp"
android:layout_marginLeft="160dp"
android:text="Get Date" />
<TextView
android:id="@+id/textView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@+id/button1"
android:layout_marginLeft="120dp"
android:layout_marginTop="10dp"
android:textStyle="bold"
android:textSize="18dp"/>
</RelativeLayout>
```

MainActivity.java

```
package com.tutlane.timepickerexample;
import android.os.Build;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.TimePicker;
public class MainActivity extends AppCompatActivity {
 TimePicker picker;
 Button btnGet;
 TextView tvw;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 tvw=(TextView)findViewById(R.id.textView1);
 picker=(TimePicker)findViewById(R.id.timePicker1);
 picker.setIs24HourView(true);
 btnGet=(Button)findViewById(R.id.button1);
 btnGet.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View v) {
 int hour, minute;
 String am_pm;
 if (Build.VERSION.SDK_INT >= 23 ){
 hour = picker.getHour();
 minute = picker.getMinute();
 }
 else{
 hour = picker.getCurrentHour();
 minute = picker.getCurrentMinute();
 }
 if(hour > 12) {
 am_pm = "PM";
 hour = hour - 12;
 }
 else
 {
 am_pm="AM";
 }
 tvw.setText("Selected Date: "+ hour +":"+ minute+" "+am_pm);
 }
 });
 }
}
```

Android ProgressBar with Examples

In android, ProgressBar is a user interface control that is used to indicate the progress of an operation. For example, downloading a file, uploading a file.

Following is the pictorial representation of using a different type of progress bars in android applications.

By default the ProgressBar will be displayed as a spinning wheel, in case if we want to show it like a horizontal bar then we need to change the style property to horizontal like style="?android:attr/progressBarStyleHorizontal".

Create Android ProgressBar in XML Layout File

In android, we can create ProgressBar in XML layout file using <ProgressBar> element with different attributes like as shown below

```
<ProgressBar
 android:id="@+id/pBar3"
 style="?android:attr/progressBarStyleHorizontal"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:minHeight="50dp"
 android:minWidth="250dp"
 android:max="100"
 android:indeterminate="true"
 android:progress="1" />
```

If you observe above code snippet, we defined a progress bar (<ProgressBar>) with different attributes, those are

Attribute	**Description**
android:id	It is used to uniquely identify the control
android:minHeight	It is used to set the height of the progress bar.
android:minWidth	It is used to set the width of the progress bar.
android:max	It is used to set the maximum value of the progress bar.
android:progress	It is used to set the default progress value between 0 and max. It must be an integer value.

In android, the ProgressBar supports two types of modes to show the progress, those are Determinate and Indeterminate.

Android ProgressBar with Determinate Mode

Generally, we use the determinate progress mode in progress bar when we want to show the quantity of progress has occurred. For example, the percentage of file downloaded, number of records inserted into a database, etc.

To use Determinate progress, we need to set the style of the progress bar to Widget_ProgressBar_Horizontal or progressBarStyleHorizontal and set the amount of progress using android:progress attribute.

Following is the example which shows a Determinate progress bar that is 50% complete.

```
<ProgressBar
 android:id="@+id/pBar"
 style="?android:attr/progressBarStyleHorizontal"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:max="100"
 android:progress="50" />
```

By using setProgress(int) method, we can update the percentage of progress displayed in app or by calling incrementProgressBy(int) method, we can increase the value of current progress completed based on our requirements.

Generally, when the progress value reaches 100 then the progress bar is full. By using android:max attribute we can adjust this default value.

Android ProgressBar with Indeterminate Mode

Generally, we use the Indeterminate progress mode in progress bar when we don't know how long an operation will take or how much work has done.

In indeterminate mode the actual progress will not be shown, only the cyclic animation will be shown to indicate that some progress is happing like as shown in the above progress bar loading images. By using progressBar.setIndeterminate(true) in activity file programmatically or using android:indeterminate = "true" attribute in XML layout file, we can enable Indeterminate progress mode.

Following is the example to set Indeterminate progress mode in an XML layout file.

```
<ProgressBar
 android:id="@+id/progressBar1"
 style="?android:attr/progressBarStyleHorizontal"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:indeterminate="true"/>
```

This is how we can define the Progress modes in **ProgressBar** based on our requirements in android applications.

Android ProgressBar Control Attributes

The following are some of the commonly used attributes related to **ProgressBar** control in android applications.

Attribute	Description
android:id	It is used to uniquely identify the control
android:max	It is used to specify the maximum value of the progress can take
android:progress	It is used to specify default progress value.
android:background	It is used to set the background color for a progress bar.
android:indeterminate	It is used to enable the indeterminate progress mode.
android:padding	It is used to set the padding for left, right, top or bottom of a progress bar.

Android ProgressBar Example

Following is the example of defining one ProgressBar control, one TextView control and one Button control in RelativeLayout to start showing the progress in the progress bar on Button click in the android application.

Create a new android application using android studio and give names as ProgressBarExample. In case if you are not aware of creating an app in android studio check this article Android Hello World App. Now open an activity_main.xml file from \res\layout path and write the code like as shown below

activity_main.xml

```
<?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="match_parent">
<ProgressBar
 android:id="@+id/pBar"
 style="?android:attr/progressBarStyleHorizontal"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginLeft="100dp"
 android:layout_marginTop="200dp"
 android:minHeight="50dp"
 android:minWidth="200dp"
 android:max="100"
 android:indeterminate="false"
 android:progress="0" />
 <TextView
 android:id="@+id/tView"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_alignLeft="@+id/pBar"
 android:layout_below="@+id/pBar" />
 <Button
 android:id="@+id/btnShow"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginLeft="130dp"
 android:layout_marginTop="20dp"
 android:text="Start Progress"
 android:layout_below="@+id/tView"/>
</RelativeLayout>
```

Once we are done with the creation of layout with required controls, we need to load the XML layout resource from our activity onCreate() callback method, for that open main activity file MainActivity.java from \java\com.tutlane.progressbarexample path and write the code like as shown below.

MainActivity.java

```
package com.tutlane.progressbarexample;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
 private ProgressBar pgsBar;
 private int i = 0;
 private TextView txtView;
 private Handler hdlr = new Handler();
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 pgsBar = (ProgressBar) findViewById(R.id.pBar);
 txtView = (TextView) findViewById(R.id.tView);
 Button btn = (Button)findViewById(R.id.btnShow);
 btn.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View v) {
 i = pgsBar.getProgress();
 new Thread(new Runnable() {
 public void run() {
 while (i < 100) {
 i += 1;
 // Update the progress bar and display the current value in text view  hdlr.post(new
Runnable() {
 public void run() {
 pgsBar.setProgress(i);
 txtView.setText(i+"/"+pgsBar.getMax());
 }
 });
 try {
 // Sleep for 100 milliseconds to show the progress slowly. Thread.sleep(100);
 } catch (InterruptedException e) {
 e.printStackTrace();
 }
 }
 }
 }).start();
 }
 });
 }
}
```

If you observe the above result, we are able to start showing the progress of the task in progress bar when we click on Button in the android application.

2.3 Horizontal and Vertical ScrollView

A **HorizontalScrollView** is a FrameLayout. The android.widget.HorizontalScrollView class provides the functionality of horizontal scroll view. HorizontalScrollView is used to scroll the child elements or views in a horizontal direction. HorizontalScrollView only supports horizontal scrolling. For vertical scroll, android uses ScrollView.

Let's implement simple example of HorizontalScrollView.

activity_main.xml

Now, drag HorizontalScrollView from palette to activity_main.xml file and place some views or elements inside it.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceSmall"
android:text="Horizontal ScrollView Example"
android:id="@+id/textView"
android:layout_alignParentTop="true"
android:layout_centerHorizontal="true" />
<LinearLayout
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_marginTop="25dp">
<HorizontalScrollView
android:layout_width="match_parent"
android:layout_height="60dp"
android:id="@+id/horizontalScrollView">
<LinearLayout
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:orientation="horizontal">
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="New Button1"
android:id="@+id/button1" />
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="New Button2"
android:id="@+id/button2" />
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="New Button3"
android:id="@+id/button3" />
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="New Button4"
```

```
  android:id="@+id/button4" />
  <Button
```

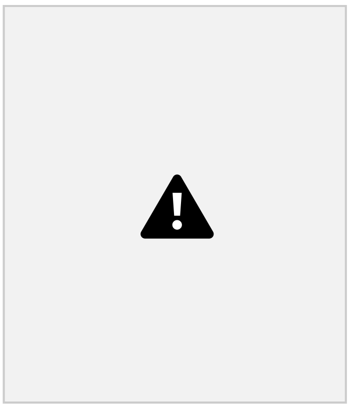
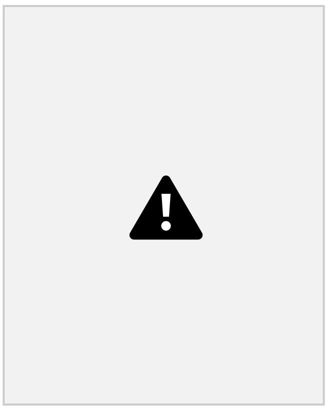
```
 android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="New Button5"
  android:id="@+id/button5" />
  <Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="New Button6"
  android:id="@+id/button6" />
  <Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="New Button7"
  android:id="@+id/button7" />
  <Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="New Button8"
  android:id="@+id/button8"/>
  </LinearLayout>
  </HorizontalScrollView>
  </LinearLayout>
</RelativeLayout>
```

This is auto generated code, we have not written any code here.

File: MainActivity.java

```
package com.example.test.horizantalscrollview;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 }
}
```

OUTPUT:

Android ScrollView (Vertical)

The android.widget.ScrollView class provides the functionality of scroll view. ScrollView is used to scroll the child elements of palette inside ScrollView. Android supports vertical scroll view as default scroll view. Vertical ScrollView scrolls elements vertically.

Android uses HorizontalScrollView for horizontal ScrollView.

Let's implement simple example of vertical ScrollView.

activity_main.xml

Now, drag ScrollView from palette to activity_main.xml file and place some palette element inside it.

```
 <?xml version="1.0" encoding="utf-8"?>
 <RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:paddingBottom="@dimen/activity_vertical_margin"
 android:paddingLeft="@dimen/activity_horizontal_margin"
 android:paddingRight="@dimen/activity_horizontal_margin"
 android:paddingTop="@dimen/activity_vertical_margin"
 tools:context="com.example.test.scrollviews.MainActivity">
 <TextView
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:textAppearance="?android:attr/textAppearanceMedium"
 android:text="Vertical ScrollView example"
 android:id="@+id/textView"
 android:layout_gravity="center_horizontal"
 android:layout_centerHorizontal="true"
 android:layout_alignParentTop="true" />
 <ScrollView android:layout_marginTop="30dp"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:id="@+id/scrollView">
 <LinearLayout
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:orientation="vertical" >
 <Button
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Button 1" />
 <Button
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Button 2" />
 <Button
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Button 3" />
 <Button
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Button 4" />
 <Button
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
```

```
 android:text="Button 5" />  <Button
 android:layout_width="fill_parent"
android:layout_height="wrap_content"
```

```
android:text="Button 6" />  <Button
 android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 7" />  <Button
 android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 8" />  <Button
 android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 9" />  <Button
 android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 10" />  <Button
 android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 11" />  <Button
 android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 12" />  <Button
 android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 13" />  <Button
 android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 14" />  <Button
 android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 15" />  <Button
 android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 16" />  <Button
 android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 17" />  <Button
 android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 18" />
```

```
 <Button
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Button 19" />
 <Button
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Button 20" />
 </LinearLayout>
 </ScrollView>
</RelativeLayout>
```

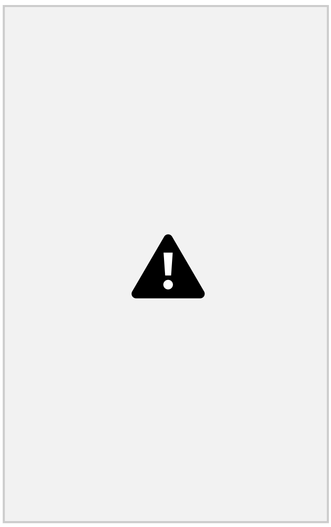
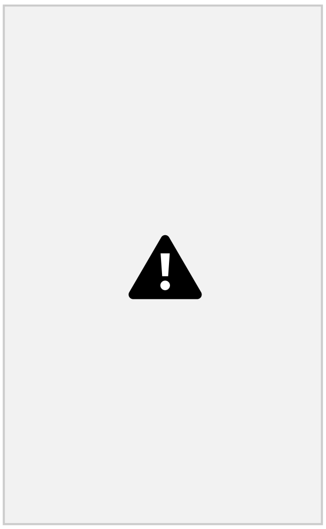
Activity class

In activity class, we have not changed any code.

File: MainActivity.java

```
package com.example.test.scrollviews;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 }
}
```

OUTPUT

2.4 AutoCompleteTextView, TextWatcher to EditText

- In android, **AutoCompleteTextView** is an editable text view which is used to show the list of suggestions based on the user typing text. The list of suggestions will be shown as a dropdown menu from which the user can choose an item to replace the content of the textbox.
- The AutoCompleteTextView is a subclass of EditText class so we can inherit all the properties of EditText in AutoCompleteTextView based on our requirements.
- Following is the pictorial representation of using AutoCompleteTextView in android applications.

- Generally, the dropdown list of suggestions can be obtained from the data adaptor and those suggestions will be appeared only after giving the number characters defined in the Threshold limit.
- The Threshold property of AutoCompleteTextView is used to define the minimum number of characters

the user must type to see the list of suggestions.
 The dropdown list of suggestions can be closed at any time in case if no item is selected from the list or by pressing the back or enter key.

In android, we can create an AutoCompleteTextView control in two ways either in the XML layout file or create it in the Activity file programmatically.

Android AutoCompleteTextView Control Example

Following is the example of defining AutoCompleteTextView control in LinearLayout to bind the data to defined control using a data adapter and getting the selected list item value in the android application.

activity_main.xml

```
<?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="match_parent"
 android:paddingLeft="20dp"
 android:paddingRight="20dp"
 android:orientation="vertical"
 android:id="@+id/linear_Layout">
 <AutoCompleteTextView
 android:id="@+id/ac_Country"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_marginTop="100dp"
 android:hint="Enter Country Name"/>
</LinearLayout>
```

MainActivity.java

```
package com.tutlane.autocompletetextviewexample;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
 String[] Countries = { "India", "USA", "Australia", "UK", "Italy", "Ireland", "Africa" };
@Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
 android.R.layout.simple_dropdown_item_1line, Countries);
 AutoCompleteTextView actv = (AutoCompleteTextView)findViewById(R.id.ac_Country);
actv.setThreshold(1);
 actv.setAdapter(adapter);
 actv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
 @Override
 public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
Toast.makeText(getApplicationContext(), "Selected Item: " + parent.getSelectedItem(),
Toast.LENGTH_SHORT).show();
 }
 });
 }
 }
```

Android EditText with TextWatcher (Searching data from ListView)
Android EditText is a subclass of TextView. EditText is used for entering and modifying text. While using EditText width, we must specify its input type in inputType property of EditText which configures the keyboard according to input.
EditText uses TextWatcher interface to watch change made over EditText. For doing this, EditText calls the addTextChangedListener() method.
Methods of TextWatcher
1. beforeTextChanged(CharSequence arg0, int arg1, int arg2, int arg3): It is executed before making any change over EditText.
2. onTextChanged(CharSequence cs, int arg1, int arg2, int arg3): It is executed while making any change over EditText.
3. afterTextChanged(Editable arg0): It is executed after change made over EditText.

Example of EditText with TextWatcher()
In this example, we will implement EditText with TextWatcher to search data from ListView.

```
 <?xml version="1.0" encoding="utf-8"?>
 <RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:paddingBottom="@dimen/activity_vertical_margin"
 android:paddingLeft="@dimen/activity_horizontal_margin"
 android:paddingRight="@dimen/activity_horizontal_margin"
 android:paddingTop="@dimen/activity_vertical_margin"
 tools:context="com.example.test.searchfromlistview.MainActivity">
 <EditText
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:id="@+id/editText"
 android:inputType="text"
 android:layout_alignParentTop="true"
 android:layout_alignParentLeft="true"
 android:layout_alignParentStart="true"
 android:layout_alignParentRight="true"
 android:layout_alignParentEnd="true" />
 <ListView
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:id="@+id/listView"
 android:layout_below="@+id/editText"
 android:layout_alignParentLeft="true"
 android:layout_alignParentStart="true" />
</RelativeLayout>
```

File: list_item.xml

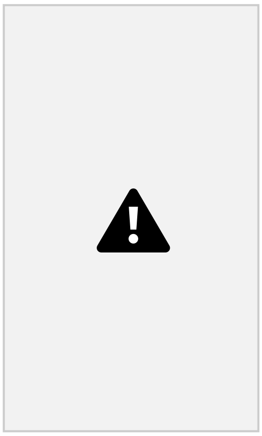
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
 xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="vertical"
 android:layout_width="match_parent"
 android:layout_height="match_parent">

<TextView android:id="@+id/product_name"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:padding="10dip"
 android:textSize="16dip"
 android:textStyle="bold"/>
</LinearLayout>
```

Activity class

```
package com.example.test.searchfromlistview;

import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.ListView;
import android.support.v7.app.AppCompatActivity;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
 private ListView lv;
 private EditText editText;
 private ArrayAdapter<String> adapter;
 private String products[] = {"Apple", "Banana","Pinapple", "Orange", "Papaya", "Melon",
"Grapes", "Water Melon","Lychee", "Guava", "Mango", "Kivi"};
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);

 lv = (ListView) findViewById(R.id.listView);
 editText = (EditText) findViewById(R.id.editText);
 adapter = new ArrayAdapter<String>(this, R.layout.list_item, R.id.product_name, products);
lv.setAdapter(adapter);
```

```
 editText.addTextChangedListener(new TextWatcher() {

 @Override
 public void onTextChanged(CharSequence cs, int arg1, int arg2, int arg3) {
adapter.getFilter().filter(cs);
 }
 @Override
 public void beforeTextChanged(CharSequence arg0, int arg1, int arg2, int arg3) {
Toast.makeText(getApplicationContext(),"before text
change",Toast.LENGTH_LONG).show();
 }
 @Override
 public void afterTextChanged(Editable arg0) {
 Toast.makeText(getApplicationContext(),"after text change",Toast.LENGTH_LONG).show();  }
 });
 }
}
```

OUTPUT:



2.5 ImageSlider, ImageSwitcher, SearchView

ImageSlider

Android Image Slider

Android image slider slides one entire screen to another screen. Image slider is created by ViewPager which is provided by support library. To implement image slider, you need to inherit ViewPager class which extends PagerAdapter.

We need to override following methods of PagerAdapter class.

1. **isViewFromObject(View, Object):** This method checks the view whether it is associated with key and returned by instantiateItem().
2. **instantiateItem(ViewGroup, int):** This method creates the page position passed as an argument.
3. **destroyItem(ViewGroup, int, Object):** It removes the page from its current position from container. In this example we simply removed object using removeView().
4. **getCount():** It returns the number of available views in ViewPager.

Example of Image Slider

Let's see an example of android image slider.

activity_main.xml

In activity_main.xml file, we have wrapped ViewPager inside RelativeLayout.

```
<?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="fill_parent"
 android:layout_height="fill_parent"
 android:paddingBottom="@dimen/activity_vertical_margin"
 android:paddingLeft="@dimen/activity_horizontal_margin"
 android:paddingRight="@dimen/activity_horizontal_margin"
 android:paddingTop="@dimen/activity_vertical_margin"
 tools:context="com.example.test.imageslider.MainActivity">

  <android.support.v4.view.ViewPager
 android:id="@+id/viewPage"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent" />
</RelativeLayout>
```

Activity class:

File: MainActivity.java

```
package com.example.test.imageslider;

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

public class MainActivity extends AppCompatActivity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);

 ViewPager mViewPager = (ViewPager) findViewById(R.id.viewPage);
 ImageAdapter adapterView = new ImageAdapter(this);
 mViewPager.setAdapter(adapterView);
 }
}
```

ImageAdapter class

Now create ImageAdapter class which extends PagerAdapter for android image slider.
Place some images in drawable folder which are to be slid.

```
package com.example.test.imageslider;

import android.content.Context;
import android.support.v4.view.PagerAdapter;
import android.support.v4.view.ViewPager;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;

public class ImageAdapter extends PagerAdapter{
 Context mContext;

 ImageAdapter(Context context) {
 this.mContext = context;
 }
 @Override
 public boolean isViewFromObject(View view, Object object) {
 return view == ((ImageView) object);
 }

 private int[] sliderImageId = new int[]{
 R.drawable.image1, R.drawable.image2, R.drawable.image3,R.drawable.image4,
R.drawable.image5,
 };

 @Override
 public Object instantiateItem(ViewGroup container, int position) {
 ImageView imageView = new ImageView(mContext);
 imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
 imageView.setImageResource(sliderImageId[position]);
 ((ViewPager) container).addView(imageView, 0);
 return imageView;
 }

 @Override
 public void destroyItem(ViewGroup container, int position, Object object) {
 ((ViewPager) container).removeView((ImageView) object);
 }

 @Override
 public int getCount() {
 return sliderImageId.length;
 }
}
```

ImageSwitcher

In android, ImageSwitcher is a specialized view switcher that will provide a smooth transition animation effect to the images while switching from one image to another.

To use ImageSwitcher in our application, we need to define an XML component .xml like as shown below.

```
<ImageSwitcher android:id="@+id/imgSw"
 android:layout_width="match_parent"
 android:layout_height="250dp">
</ImageSwitcher>
```

After that we need to create an instance of ImageSwitcher and use setFactory() method to implement the ViewFactory interface to return the ImageView. We need to add the required animation effects to ImageSwitcher based on our requirements.
If you observe above code snippet, we created an instance of ImageSwitcher,
used setFactory() method to implement ViewFactory interface and added required animation effects.
Apart from the above methods, the ImageSwitcher class is having different methods available, those are

Method	Description
setImageDrawable	This method is used to set a new drawable on the next ImageView in the switcher.
setImageResource	This method is used to set a new image on the ImageSwitcher with the given resource id.
setImageURI	This method is used to set a new image on the ImageSwitcher with the given Uri.

Now we will see how to use ImageSwitcher to add smooth transition animation effects while switching between the images in android application with examples.

Android ImageSwitcher Example

Following is the example of switching between the images with a smooth animation transition effect using imageswitcher in the android application.

Create a new android application using android studio and give names as ImageSwitcherExample. In case if you are not aware of creating an app in android studio check this article Android Hello World App.
Now open activity_main.xml file from \res\layout folder path and write the code like as shown below.

activity_main.xml

```
<?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="match_parent"
 android:paddingLeft="10dp"
 android:paddingRight="10dp">
 <ImageSwitcher android:id="@+id/imgSw"
 android:layout_width="match_parent"
 android:layout_height="250dp"/>
 <Button
 android:id="@+id/btnPrevious"
 android:layout_below="@+id/imgSw"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Previous" android:layout_marginLeft="100dp" />
 <Button
 android:id="@+id/btnNext"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_alignBottom="@+id/btnPrevious"
 android:layout_toRightOf="@+id/btnPrevious"
 android:text="Next" />
</RelativeLayout>
```

Now open your main activity file MainActivity.java from \java\com.tutlane.imageswitcherexample path and write the code like as shown below

MainActivity.java

```
package com.tutlane.imageswitcherexample;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageSwitcher;
import android.widget.ImageView;
import android.widget.ViewSwitcher;

public class MainActivity extends AppCompatActivity {
 private Button previousbtn, nextbtn;
 private ImageSwitcher imgsw;
 private int[] images = {R.drawable.bangkok,R.drawable.bangkok2};
 private int position = 0;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 previousbtn = (Button)findViewById(R.id.btnPrevious);
 nextbtn = (Button)findViewById(R.id.btnNext);
 imgsw = (ImageSwitcher) findViewById(R.id.imgSw);
 imgsw.setFactory(new ViewSwitcher.ViewFactory() {
 @Override
 public View makeView() {
 ImageView imgVw= new ImageView(MainActivity.this);
 imgVw.setImageResource(images[position]);
 return imgVw;
 }
 });
 imgsw.setInAnimation(this, android.R.anim.slide_in_left);
 imgsw.setOutAnimation(this, android.R.anim.slide_out_right);
 previousbtn.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View v) {
 if(position>0)
 position--;
 else if(position<0)
 position = 0;
 imgsw.setImageResource(images[position]);
 }
 });
 nextbtn.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View v) {
 if(position<images.length)
 position++;
 if(position>=images.length)
 position = images.length-1;
 imgsw.setImageResource(images[position]);
 }
 });
 }
}
```

If you observe above code, we used a set of images (bangkok, bangkok2) from drawable folder, you need to add your images in drawable folder and by using imageswitcher setFactory() and setImageResource() methods we are switching the images with slide in / slide out animation effect.

Android SearchView

Android SearchView provides user interface to search query submitted over search provider. SearchView widget can be implemented over ToolBar/ActionBar or inside a layout.

SearchView is by default collapsible and set to be iconified using setIconifiedByDefault(true) method of SearchView class. For making search field visible, SearchView uses setIconifiedByDefault(false) method.

Methods of SearchView

1. public boolean onQueryTextSubmit(String query): It searches the query on the submission of content over SearchView editor. It is case dependent.
2. public boolean onQueryTextChange(String newText): It searches the query at the time of text change over SearchView editor.

Example of SearchView

Let's see the example of SearchView over layout, searching data in a ListView.

activity_main.xml

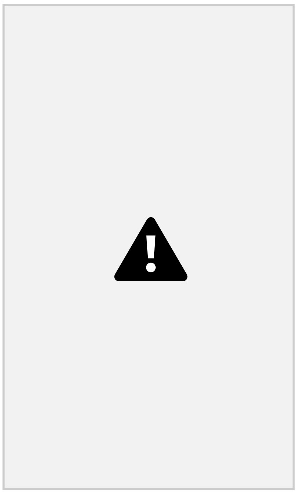
Create an activity_main.xml file in layout folder containing ScrollView and ListView.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:paddingBottom="@dimen/activity_vertical_margin"
 android:paddingLeft="@dimen/activity_horizontal_margin"
 android:paddingRight="@dimen/activity_horizontal_margin"
 android:paddingTop="@dimen/activity_vertical_margin"
 tools:context="com.example.test.searchview.MainActivity">
 <ListView
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:id="@+id/lv1"
 android:divider="#ad5"
 android:dividerHeight="2dp"
 android:layout_below="@+id/searchView"/>
 <SearchView
 android:id="@+id/searchView"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:queryHint="Search Here"
 android:iconifiedByDefault="false"
 android:layout_alignParentTop="true" />
</RelativeLayout>
```

File: MainActivity.java

```
package com.example.test.searchview;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.Filter;
import android.widget.ListView;
import android.widget.SearchView;
import android.widget.Toast;
import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {
 SearchView searchView;
 ListView listView;
 ArrayList<String> list;
 ArrayAdapter<String > adapter;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 searchView = (SearchView) findViewById(R.id.searchView);
 listView = (ListView) findViewById(R.id.lv1);
 list = new ArrayList<>();
 list.add("Apple");
 list.add("Banana");
 list.add("Pineapple");
 list.add("Orange");
 list.add("Lychee");
 list.add("Gavava");
 list.add("Peech");
 list.add("Melon");
 list.add("Watermelon");
 list.add("Papaya");
 adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,list);
listView.setAdapter(adapter);
 searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
@Override
 public boolean onQueryTextSubmit(String query) {

 if(list.contains(query)){
 adapter.getFilter().filter(query);
 }else{
 Toast.makeText(MainActivity.this, "No Match found",Toast.LENGTH_LONG).show();  }
 return false;
 }
 @Override
 public boolean onQueryTextChange(String newText) {
 // adapter.getFilter().filter(newText);
 return false;
 }
 });
 }
}
```

OUTPUT:



2.6 TabLayout and FrameLayout

Android TabLayout

 TabLayout is used to implement horizontal tabs. TabLayout is released by Android after the deprecation of ActionBar.TabListener (API level 21).

 TabLayout is introduced in design support library to implement tabs.

 Tabs are created using newTab() method of TabLayout class. The title and icon of Tabs are set through setText(int) and setIcon(int) methods of TabListener interface respectively. Tabs of layout are attached over TabLayout using the method addTab(Tab) method.

```
TabLayout tabLayout = (TabLayout)findViewById(R.id.tabLayout);
tabLayout.addTab(tabLayout.newTab().setText("Tab 1"));
tabLayout.addTab(tabLayout.newTab().setText("Tab 2"));
tabLayout.addTab(tabLayout.newTab().setText("Tab 3"));
```

We can also add tab item to TabLayout using TabItem of android design widget.

```
<android.support.design.widget.TabItem
 android:text="@string/tab_text"/>
```

Example of TabLayout using ViewPager

Let's create an example of TabLayout using ViewPager and Fragment.

File: activity.xml -Create an activity.xml file with TabLayout and ViewPager view components.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
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"
 tools:context="tablayout.example.com.tablayout.MainActivity">
 <android.support.design.widget.TabLayout
 android:id="@+id/tabLayout"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:background="#1db995">
 </android.support.design.widget.TabLayout>
 <android.support.v4.view.ViewPager
 android:id="@+id/viewPager"
```

```
 android:layout_width="355dp"
 android:layout_height="455dp"
 app:layout_constraintTop_toBottomOf="@+id/tabLayout"
 tools:layout_editor_absoluteX="8dp" />
</android.support.constraint.ConstraintLayout>
```

File: build.gradle

Now gave the dependency library of TabLayout in build.gradle file.

```
implementation 'com.android.support:design:26.1.0'
```

File: MainActivity.java

In this file, we implement two additional listener addOnPageChangeListener(listener) of ViewPager which makes slides the different fragments of tabs and addOnTabSelectedListener(listener) of TabLayout which select the current tab on tab selection.

```
package tablayout.example.com.tablayout;

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

public class MainActivity extends AppCompatActivity {
 TabLayout tabLayout;
 ViewPager viewPager;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);

 tabLayout=(TabLayout)findViewById(R.id.tabLayout);
 viewPager=(ViewPager)findViewById(R.id.viewPager);

 tabLayout.addTab(tabLayout.newTab().setText("Home"));
 tabLayout.addTab(tabLayout.newTab().setText("Sport"));
 tabLayout.addTab(tabLayout.newTab().setText("Movie"));
 tabLayout.setTabGravity(TabLayout.GRAVITY_FILL);

 final MyAdapter adapter = new MyAdapter(this,getSupportFragmentManager(),
tabLayout.getTabCount());
 viewPager.setAdapter(adapter);

 viewPager.addOnPageChangeListener(new
TabLayout.TabLayoutOnPageChangeListener(tabLayout));

 tabLayout.addOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
@Override
 public void onTabSelected(TabLayout.Tab tab) {
 viewPager.setCurrentItem(tab.getPosition());
 }

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

```
 }

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

 }
 });
 }
}
```

File: MyAdapter.java

```
package tablayout.example.com.tablayout;

import android.content.Context;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentPagerAdapter;
import android.support.v4.app.FragmentManager;
public class MyAdapter extends FragmentPagerAdapter {
 private Context myContext;
 int totalTabs;

 public MyAdapter(Context context, FragmentManager fm, int totalTabs) {
 super(fm);
 myContext = context;
 this.totalTabs = totalTabs;
 }

 // this is for fragment tabs
 @Override
 public Fragment getItem(int position) {
 switch (position) {
 case 0:
 HomeFragment homeFragment = new HomeFragment();
 return homeFragment;
 case 1:
 SportFragment sportFragment = new SportFragment();
 return sportFragment;
 case 2:
 MovieFragment movieFragment = new MovieFragment();
 return movieFragment;
 default:
 return null;
 }
 }
// this counts total number of tabs
 @Override
 public int getCount() {
 return totalTabs;
 }
}
```

Now create different fragment files for all different tabs.

File: HomeFragment.java

```
package tablayout.example.com.tablayout;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class HomeFragment extends Fragment {
 public HomeFragment() {
 // Required empty public constructor
 }
 @Override
 public View onCreateView(LayoutInflater inflater, ViewGroup container,
 Bundle savedInstanceState) {
 // Inflate the layout for this fragment
 return inflater.inflate(R.layout.fragment_home, container, false);
 }
}
```

File: fragment_home.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="tablayout.example.com.tablayout.HomeFragment">
 <!-- TODO: Update blank fragment layout -->
 <TextView
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:gravity="center"
 android:text="@string/home_fragment" />

</FrameLayout>
```

File: SportFragment.java

```
package tablayout.example.com.tablayout;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class SportFragment extends Fragment {
 public SportFragment() {
 // Required empty public constructor
 }
 @Override
 public View onCreateView(LayoutInflater inflater, ViewGroup container,
 Bundle savedInstanceState) {
 // Inflate the layout for this fragment
 return inflater.inflate(R.layout.fragment_sport, container, false);
 }

}
```

File: fragment_sport.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="tablayout.example.com.tablayout.SportFragment">

 <!-- TODO: Update blank fragment layout -->
 <TextView
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:gravity="center"
 android:text="@string/sport_fragment" />

</FrameLayout>
```

File: MovieFragment.java

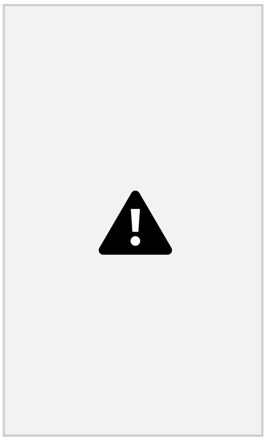
```
package tablayout.example.com.tablayout;

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

public class MovieFragment extends Fragment {


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

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

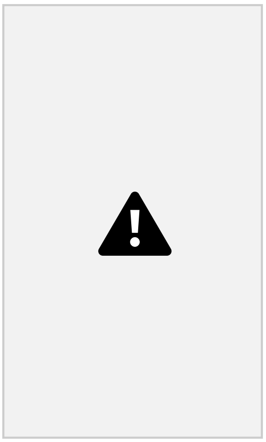
File: fragment_movie.xml

```
<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="tablayout.example.com.tablayout.MovieFragment">

 <!-- TODO: Update blank fragment layout -->
 <TextView
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:gravity="center"
 android:text="@string/movie_fragment" />

</FrameLayout>
```

File: strings.xml

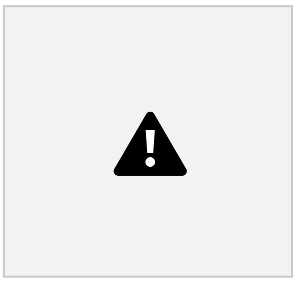
```
<resources>
 <string name="app_name">TabLayout</string>

 <!-- TODO: Remove or change this placeholder text -->
 <string name="home_fragment">Home Fragment</string>
 <string name="sport_fragment">Sport Fragment</string>
 <string name="movie_fragment">Movie Fragment</string>

</resources>
```





Android FrameLayout with Examples

 In android, Framelayout is a ViewGroup subclass that is used to specify the position of View instances it contains on the top of each other to display only single View inside the FrameLayout.
 In simple manner, we can say FrameLayout is designed to block out an area on the screen to display a single item.
 Following is the pictorial representation of frame layout in android applications.



 In android, FrameLayout will act as a placeholder on the screen and it is used to hold a single child view.
 In FrameLayout, the child views are added in a stack and the most recently added child will show on the top. We can add multiple children views to FrameLayout and control their position by using gravity attributes in FrameLayout.

Android FrameLayout Example

Following is the example of creating a FrameLayout with different controls in android application. Create a new android application using android studio and give names as FrameLayout. In case if you are not aware of creating an app in android studio check this article Android Hello World App. Now open an activity_main.xml file from \res\layout path and write the code like as shown below
activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:orientation="vertical">
 <ImageView
 android:id="@+id/imgvw1"
```

```
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:scaleType="centerCrop"
 android:src="@drawable/flimg" />
 <TextView
 android:id="@+id/txtvw1"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_marginTop="40dp"
 android:background="#4C374A"
 android:padding="10dp"
 android:text="Grand Palace, Bangkok"
 android:textColor="#FFFFFF"
 android:textSize="20sp" />
 <TextView
 android:id="@+id/txtvw2"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_gravity="right|bottom"
 android:background="#AA000000"
 android:padding="10dp"
 android:text="21/Aug/2017"
 android:textColor="#FFFFFF"
 android:textSize="18sp" />
</FrameLayout>
```

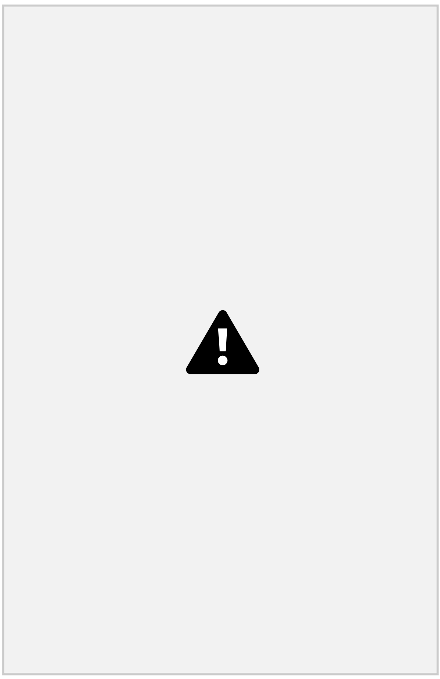
If you observe above code we used ImageView to show the image (flimg) from drawable folder in framelayout. So add your image to drawable folder and replace @drawable/flimg path with your image path.
Once we are done with the creation of layout, we need to load the XML layout resource from our

activity onCreate() callback method, for that open main activity file MainActivity.java from \java\com.tutlane.framelayout path and write the code like as shown below.

MainActivity.java

```
package com.tutlane.linearlayout;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 }
}
```

OUTPUT:



Reference:
https://www.javatpoint.com/android-tutorial
https://www.tutlane.com/tutorial/android
https://www.geeksforgeeks.org/custom-checkbox-in-android/