MOBILE APPLICATION DEVELOPMENT
PART I

PART 1 contents at a glance:

1. Introduction to Android
2. Features of Android,
3. Android Versions,
4. Android Architecture,
5. Installing Android SDK Tools,
6. Android Development Tools (ADT),
7. Creating Android Virtual Devices (AVD)

Introduction to Android:
Android is an operating system and programming platform developed by Google for mobile phones and other mobile devices, such as tablets. It can run on many different devices from many different manufacturers. Android includes a software development kit (SDK) that helps you write original code and assemble software modules to create apps for Android users. Android also provides a marketplace to distribute apps.

Android Features:
Android is a Linux based operating system it is designed primarily for touch screen mobile devices such as smart phones and tablet computers. The operating system have developed a lot in last 15 years starting from black and white phones to recent smart phones or mini computers.

Features:

- It is open-source.
- Anyone can customize the Android Platform.
- There are a lot of mobile applications that can be chosen by the consumer.
- Head set layout
- Storage
- Connectivity: GSM/EDGE, IDEN, CDMA, Bluetooth, WI-FI, EDGE,3G,NFC, LTE,GPS.
- Messaging: SMS, MMS, C2DM (could to device messaging), GCM (Google could messaging)
- Multilanguage support
- Multi touch
- Video calling
- Screen capture
- External storage
- Streaming media support(H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, OggVorbis, WAV, JPEG, PNG, GIF, and BMP)
- Optimized graphics
- Resizable
- Multi tasking

Android Versions:

Name	Version number(s)	Initial release date	API level
No codename	1.0	September 23, 2008	1
	1.1	February 9, 2009	2
Cupcake	1.5	April 27, 2009	3
Donut	1.6	September 15, 2009	4
Eclair	2.0 – 2.1	October 26, 2009	5 – 7
Froyo	2.2 – 2.2.3	May 20, 2010	8
Gingerbread	2.3 – 2.3.7	December 6, 2010	9 – 10
Honeycomb	3.0 – 3.2.6	February 22, 2011	11 – 13
Ice Cream Sandwich	4.0 – 4.0.4	October 18, 2011	14 – 15
Jelly Bean	4.1 – 4.3.1	July 9, 2012	16 – 18
KitKat	4.4 – 4.4.4	October 31, 2013	19 – 20
Lollipop	5.0 – 5.1.1	November 12, 2014	21 – 22
Marshmallow	6.0 – 6.0.1	October 5, 2015	23
Nougat	7.0 – 7.1.2	August 22, 2016	24 – 25
Oreo	8.0 – 8.1	August 21, 2017	26 – 27
Pie	9.0	August 6, 2018	28
Android 10	10.0	September 3, 2019	29
Android 11	11.0	TBD	

Android Applications

Android applications are usually developed in the Java language using the Android Software Development Kit. Once developed, Android applications can be packaged easily and sold out either through a store such as **Google Play**,**SlideME**,**Opera Mobile Store**,**Mobango**,**F-droid** and the **Amazon Appstore**.

Android powers hundreds of millions of mobile devices in more than 190 countries around the world. It's the largest installed base of any mobile platform and growing fast. Every day more than 1 million new Android devices are activated worldwide.

Categories of Android applications

There are many android applications in the market. The top categories are:

- Entertainment
- Tools
- Communication
- Productivity
- Personalization
- Music and Audio

- Social
- Media and Video
- Travel and Local etc.
- Games applications
- Educational applications
- Business applications

Advantages of Android OS:

- Open Ecosystem
- Customizable UI
- Open Source
- Innovations Reach the Market Quicker
- Affordable Development
- APP Distribution
- Affordable
- Expandable memory
- Run many apps at the same time:-
- Cloud Storage etc

Disadvantages of Android OS:

- Low specification mobiles run slow
- Developing apps for different H/W devices is hard
- Running apps in background may consume more battery etc

SOFTWARE IDES TO DEVELOP ANDROID APPS:

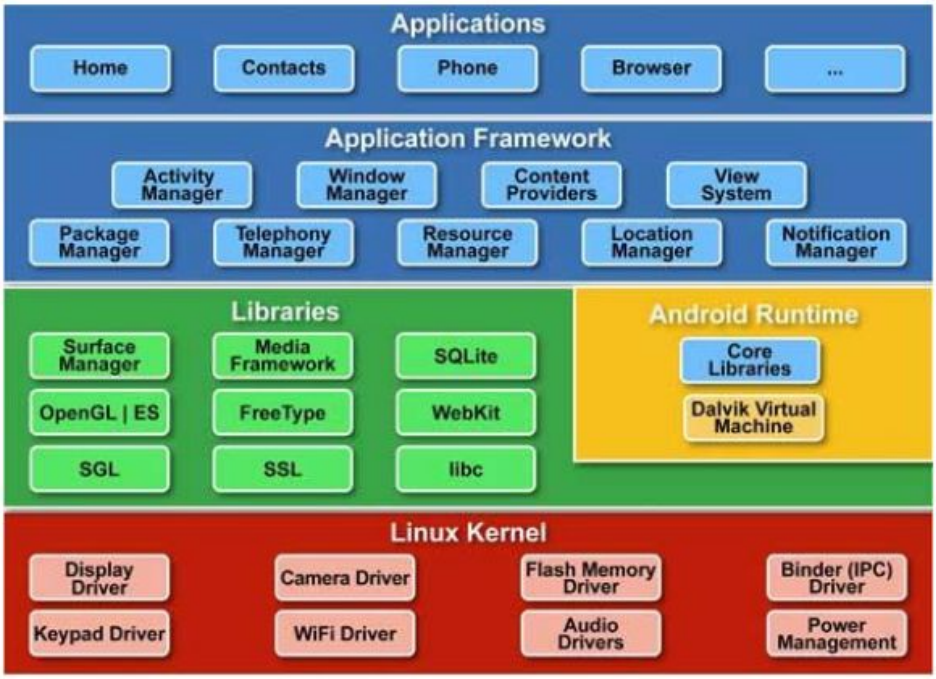
There are so many sophisticated Technologies are available to develop android applications, the familiar technologies, which are predominantly using tools as follows

- Android Studio
- Eclipse IDE
- Adobe Flash etc

ANDROID ARCHITECTURE:

Android Architecture

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.



Linux kernel

At the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

Android Libraries

This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access. A summary of some key core Android libraries available to the Android developer is as follows –

- **android.app** – Provides access to the application model and is the cornerstone of all Android applications.
- **android.content** – Facilitates content access, publishing and messaging between applications and application components.
- **android.database** – Used to access data published by content providers and includes SQLite database management classes.
- **android.opengl** – A Java interface to the OpenGL ES 3D graphics rendering API.
- **android.os** – Provides applications with access to standard operating system services including messages, system services and inter-process communication.
- **android.text** – Used to render and manipulate text on a device display.
- **android.view** – The fundamental building blocks of application user interfaces.
- **android.widget** – A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
- **android.webkit** – A set of classes intended to allow web-browsing capabilities to be built into applications.

Having covered the Java-based core libraries in the Android runtime, it is now time to turn our attention to the C/C++ based libraries contained in this layer of the Android software stack.

Android Runtime

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called **Dalvik Virtual Machine** which is a kind of Java Virtual Machine specially designed and optimized for Android.

The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

The Android framework includes the following key services –

- **Activity Manager** – Controls all aspects of the application lifecycle and activity stack.

- **Content Providers** – Allows applications to publish and share data with other applications.
- **Resource Manager** – Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
- **Notifications Manager** – Allows applications to display alerts and notifications to the user.
- **View System** – An extensible set of views used to create application user interfaces.

Applications

You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, and Games etc.

Installing Android Studio IDE:

1) Make sure that java is installed and java environment variable paths are created

2) download android studio from : https://developer.android.com/studio

3) After downloading android studio, click on EXE file and run the set up file

4) Go through the set up wizard and complete all steps .

Installing Android SDK Tools

1. The Android SDK contains a debugger, libraries, an emulator, documentation, sample code, and tutorials. Download the Android SDK from http://developer.android.com/sdk/ index.html.
2. Once the SDK is downloaded, unzip its content into the C:\Android\ folder.
3. SDK directory should be set folder where SDK is downloaded

or

1. Launch android studio
2. Go to SDK manager
3. Check whether SDK packages of required versions are downloaded
4. If not downloaded, download.

Android Development Tools (ADT)

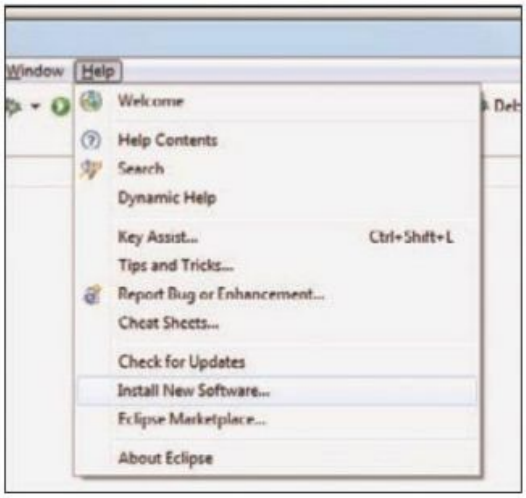
1. The Android Development Tools (ADT) plug-in for Eclipse is an extension to the Eclipse IDE that supports the creation and debugging of Android applications.

Using the ADT, you will be able to do the following in Eclipse:

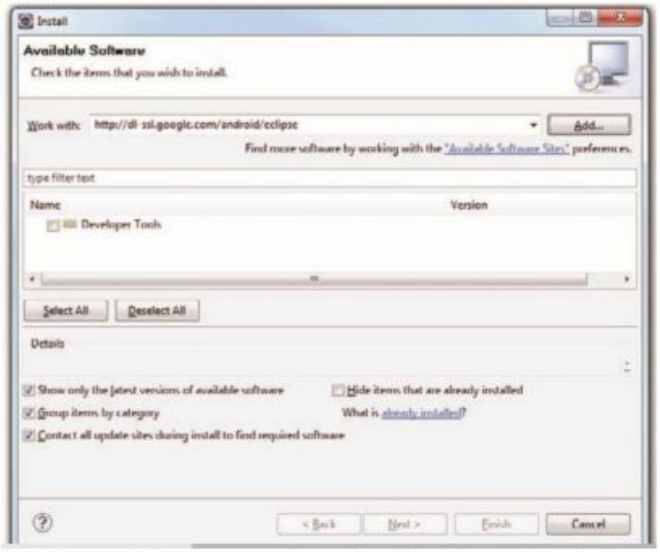
- Create new Android application projects.
- Access the tools for accessing your Android emulators and devices.
- Compile and debug Android applications.
- Export Android applications into Android Packages (APK).
- Create digital certificates for code-signing your APK

To install the ADT

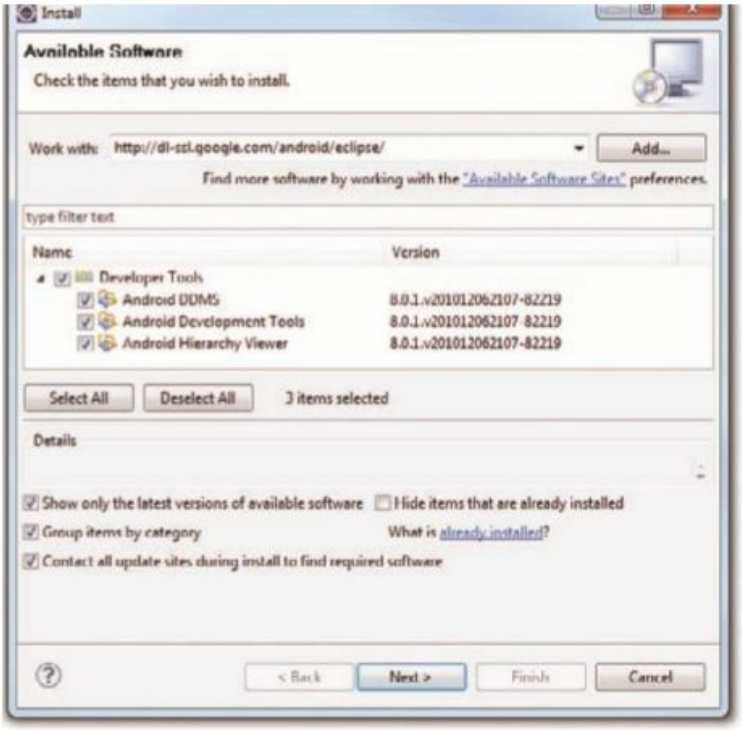
1. Once Eclipse is up and running, select the **Help ➪ Install New Software**... menu item.



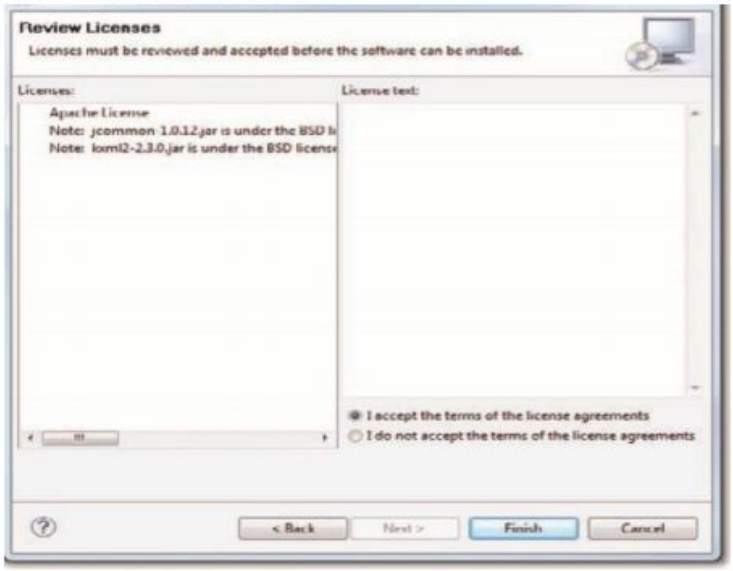
2. In the Install window that appears, type **http://dl-ssl.google.com/android/eclipse** in the text box and click Add.



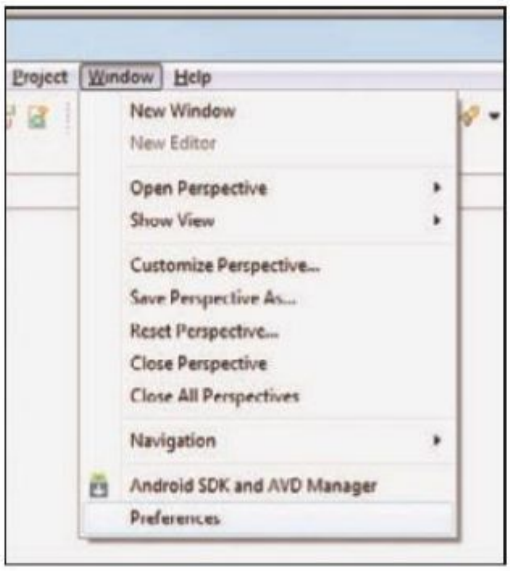
3. After a while, you will see the Developer Tools item appear in the middle of the window. Expand it, and it will reveal its content: Android DDMS, Android Development Tools, and Android Hierarchy Viewer. Check all of them and click Next.



4. You will be asked to review the licenses for the tools. Check the option to accept the license agreements. Click Finish to continue.



5. Once the ADT is installed, you will be prompted to restart Eclipse. After doing so, go to **Window ➪ Preferences**



6. In the Preferences window that appears, select Android. You will see an error message saying that the SDK has not been set up. Click OK to dismiss it.

7. Enter the location of the Android SDK folder. It would be C:\Android\android-sdk-windows. Click OK.

CREATING ANDROID VIRTUAL DEVICES (AVDS)

AVD is used for testing your Android applications

1. Select the **Window menu > AVD Manager**
2. Click on the **new** button, to create the AVD
3. Now a dialog appears, write the AVD name e.g. myavd. Now choose the target android version e.g. android2.2.
4. click the **create AVD**

Steps in Creating a new android application:

Create a New Android Project

1. Launch Android Studio.
2. Choose "Start a new Android Studio Project".
3. In "Select a Project Template" ⇒ select "Phone and Tablet" tab ⇒ "Empty Activity" ⇒ Next.
4. In "Configure your project" ⇒ Set "Name" to "Hello Android" (this will be the "Title" in your phone's app menu) ⇒ The "Package name" and "Save Location" will be updated automatically ⇒ In "Language", select "Java" ⇒ Leave the "Minimum API Level" and the rest to default ⇒ Finish.

MOBILE APPLICATION DEVELOPMENT
PART II

PART II contents at a glance:

1. Creating first android application,
2. Anatomy of android application,
3. Deploying Android app on USB connected Android device,
4. Core Building blocks or components of Android application,
5. Activity life cycle,
6. Understanding activities,
7. Exploring Intent objects,
8. Intent Types,
9. Linking activities using intents(explicit and implicit intents) & EXTRAS

1. Creating first android application:

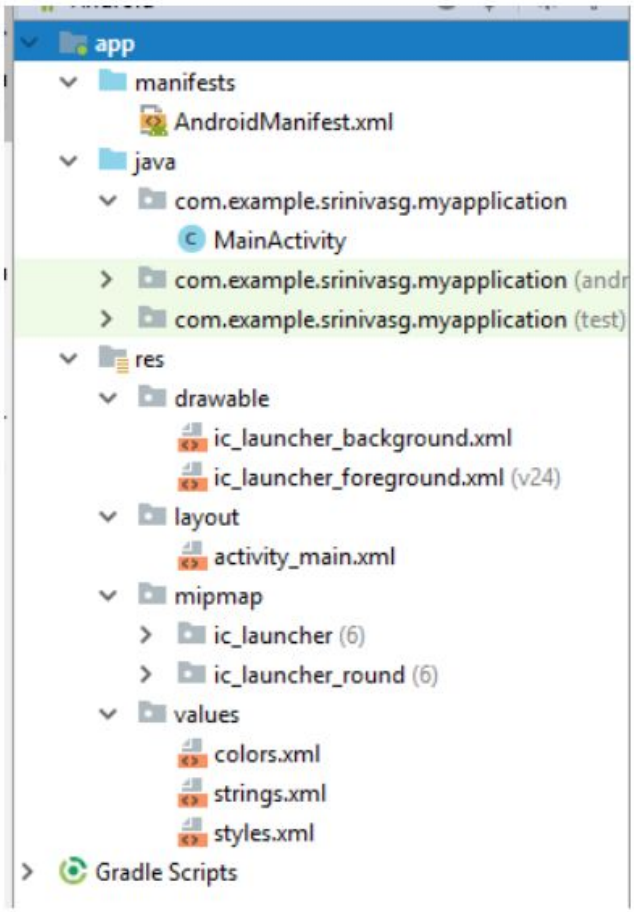
1. Open Android Studio.
2. Go to file-->New--->New project

   or

   Under the "Quick Start" menu, select "Start a new Android Studio project."

3. On the "Create New Project" window that opens, name your project "HelloWorld".
4. If you choose to, set the company name as desired*.
5. Note where the project file location is and change it if desired.
6. Click "Next."
7. Make sure on that "Phone and Tablet" is the only box that is checked.
8. If you are planning to test the app on your phone, make sure the minimum SDK is below your phone's operating system level.
9. Click "Next."
10. Select "Empty Activity."
11. Click "Next."
12. Leave all of the Activity name fields as they are.
13. Click "Finish."

2. Anatomy of android application:



The various files are as follows:
AndroidManifest.xml

This is the manifest file for your Android application. Here you specify the permissions needs by your application, package name, icon launcher, application name, and other features (such as Intent-filters, activity, receives, etc.).

MainActivity.java

Located inside java/your_package/ -- this is source file for your project (JAVA language). Here, the **MainActivity.java** file is the source file for your Activity. You write code for your application in this file. The java file is listed under package name for your project,

res/drawable
The folder contains all assets used by your application, such as files, text files, images, etc.

res/layout
The folder contains all layout used by application, with XML format.
Here, define the user interface for the application, define components and attributes (such as id, default value, width, height, position,...)

- **res/mipmap**

The folder contains all icons used by application (icon launcher, icons button, etc.) with multiple resolution format (hdpi, mdpi, xhdpi, xxhdpi, xxxhdpi). [about density]

- **res/values**

This folder contains 3 files default: **strings.xml** (define STRINGs with NAME), **colors.xml** (defined COLORs code by NAME), **styles.xml**. Here you save all the string contants in your application (*objects that do not change later*) , and when want use only call NAME.

3. Deploying Android app on USB connected Android device

1. enable "USB debugging " option in mobile
2. enable "allow apps from external sources" option in mobile
3. connect your mobile to system using USB cable
4. run your app and select your mobile device shown under tab "connected devices"

4. Android Application components or core building blocks:

Android Application Components or Building blocks of Android App:

There are four types of App Components:

1. Activities
2. Services
3. Content Providers
4. Broadcast Receivers

1) Activities:

An *activity* represents a single screen with a user interface.
example 1, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails.
When we switch to next activity, the previous activity is pushed into stack called back stack. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others. As such, a different app can start any one of these activities (if the email app allows it).
example2, a camera app can start the activity in the email app that composes new mail, in order for the user to share a picture.

2) Services:
A *service* is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.

3) Content Providers:
A *content provider* manages a shared set of app data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your app can access. Through the content provider, other apps can query or even modify the data (if the content provider allows it).
Content providers manages data sharing between applications.

4) Broadcast Receivers:
A *broadcast receiver* is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs.

INTENTS:
Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an *intent*. Intents bind individual components to each other at runtime (you can think of them as the messengers that request an action from other components), whether the component belongs to your app or another.
For activities and services, an intent defines the action to perform (for example, to "view" or "send" something) and may specify the URI of the data to act on (among other things that the component being started might need to know). For example, an intent might convey a request for an activity to show an image or to open a web page.

THE MANIFEST FILE
Before the Android system can start an app component, the system must know that the component exists by reading the app's AndroidManifest.xml file (the "manifest" file). Your app must declare all its components in this file, which must be at the root of the app project directory.

The manifest does a number of things in addition to declaring the app's components, such as:

- Identify any user permissions the app requires, such as Internet access or read-access to the user's contacts.
- Declare the minimum API Level required by the app, based on which APIs the app uses.
- Declare hardware and software features used or required by the app, such as a camera, bluetooth services, or a multitouch screen.

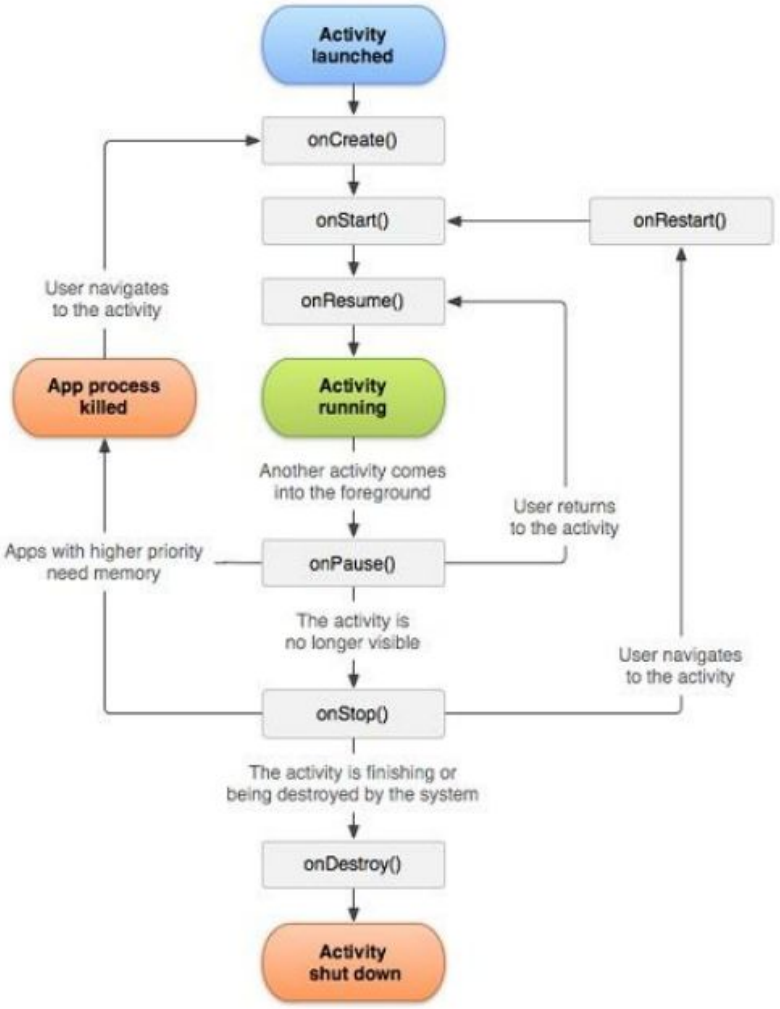
Android Virtual Device (AVD)

It is used to test the android application without the need for mobile or tablet etc. It can be created in different configurations to emulate different types of real devices.

Android Emulator is used to run, debug and test the android application. If you don't have the real device, it can be the

best way to run, debug and test the application.

5. Android Activity Life cycle:



- The **entire lifetime** of an activity happens between the first call to onCreate(Bundle) through to a single final call to onDestroy(). An activity will do all setup of "global" state in onCreate(), and release all remaining resources in onDestroy(). For example, if it has a thread running in the background to download data from the network, it may create that thread in onCreate() and then stop the thread in onDestroy().
- The **visible lifetime** of an activity happens between a call to onStart() until a corresponding call to onStop(). During this time the user can see the activity on-screen, though it may not be in the foreground and interacting with the user. Between these two methods you can maintain resources that are needed to show the activity to the user. For example, you can register a BroadcastReceiver in onStart() to monitor for changes that impact your UI, and unregister it in onStop() when the user no longer sees what you are displaying. The onStart() and onStop() methods can be called multiple times, as the activity becomes visible and hidden to the user.
- The **foreground lifetime** of an activity happens between a call to onResume() until a corresponding call to onPause(). During this time the activity is in front of all other activities and interacting with the user. An activity can frequently

go between the resumed and paused states -- for example when the device goes to sleep, when an activity result is delivered, when a new intent is delivered -- so the code in these methods should be fairly lightweight.

The entire lifecycle of an activity is defined by the following Activity methods. All of these are hooks that you can override to do appropriate work when the activity changes state. All activities will implement onCreate(Bundle) to do their initial setup; many will also implement onPause() to commit changes to data and otherwise prepare to stop interacting with the user. You should always call up to your superclass when implementing these methods.

Android activity life cycle methods:

Method	Description	Next
onCreate()	Called when the activity is first created.	onStart()
onRestart()	Called after your activity has been stopped, just prior to it being started again. Always followed by onStart()	onStart()
onStart()	Called when the activity is becoming visible to the user. Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden.	onResume()or onStop()
onResume()	Called when the activity will start interacting with the user. Always followed by onPause().	onPause()
onPause()	Called when the system is about to start resuming a previous activity.	onResume() or onStop()
onStop()	Called when the activity is no longer visible to the user. Followed by either onRestart() if this activity is coming back to interact with the user, or onDestroy() if this activity is going away.	onRestart() or onDestroy()
onDestroy()	The final call you receive before your activity is destroyed.	nothing

PROGRAM TO IMPLEMENT ACTIVITY LIFE CYCLE METHODS:

```
package example.myapplication.com.activitylifecycle;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity
{

@Override
protected void onCreate(Bundle savedInstanceState)
{
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   Log.d("lifecycle","onCreate invoked");
}
```

```
@Override
protected void onStart()
{
   super.onStart();
   Log.d("lifecycle","onStart invoked");
}

@Override
protected void onResume()
{
   super.onResume();
   Log.d("lifecycle","onResume invoked");
}
@Override
protected void onPause()
{
   super.onPause();
   Log.d("lifecycle","onPause invoked");
}
@Override
protected void onStop()
{
   super.onStop();
   Log.d("lifecycle","onStop invoked");
}
@Override
protected void onRestart()
{
   super.onRestart();
   Log.d("lifecycle","onRestart invoked");
}
@Override
protected void onDestroy()
{
   super.onDestroy();
   Log.d("lifecycle","onDestroy invoked");
    }
}
```

Intents:

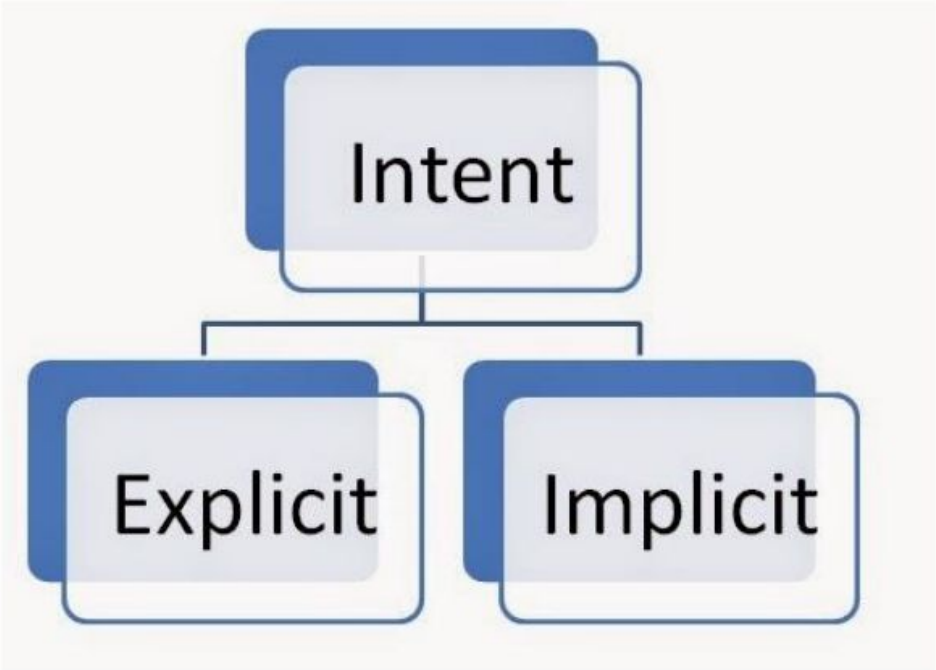
Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an *intent*.

Intent and Intent Filter

- **Intent** is an object carrying an *intent* ie. message from one component to another component with-in the application or outside the application
- The intents can communicate messages among any of the three core components of an application - activities, services, and broadcast receivers.
- For example, via the startActivity() method you can define that the intent should be used to start an activity.
- An intent can contain data via a Bundle. This data can be used by the receiving component.
- Separate mechanisms for delivering intents to each type of component

S.N.	Method & Description
1	**Context.startActivity()** The Intent object is passed to this method to launch a new activity or get an existing activity to do something new.
2	**Context.startService()** The Intent object is passed to this method to initiate a service or deliver new instructions to an ongoing service.
3	**Context.sendBroadcast()** The Intent object is passed to this method to deliver the message to all interested broadcast receivers.

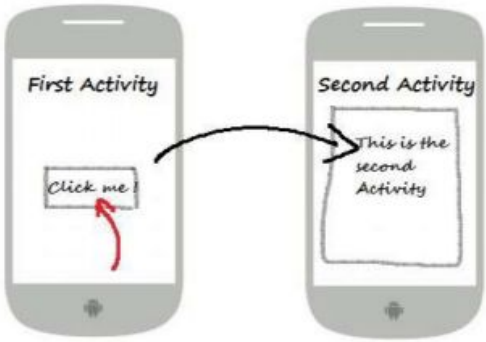
Types of Intents:



Intent : Explicit Intent

- These intents designate the target component by its name and they are typically used for application-internal messages - such as an activity starting a subordinate service or launching a sister activity.

- Example



Sample code for explicit intent:

```
Intent i = new Intent(this,TargetActivity.class)
startActivity(i);
```

the above code launches an activity from same application. the activity name is Target Activity.

Explicit Intent With Extras:

Intent : Explicit Intent

- For example if we want to start a new Activity and want to pass some value then the below code will do that.

```
Intent intent = new Intent(this, TargetActivity.class);
intent.putExtra("Key1", "ABC");  // putting value to
extra
intent.putExtra("Key2", "123"); // Starts
TargetActivity
startActivity(intent);
```

- First we are defining the new Activity then putting the value in Extra and finally starting the activity using startActivity() method. To get this value in TargetActivity like this

```
Bundle extras = getIntent().getExtras();
String inputString1 = extras.getString("Key1");
String inputString2 = extras.getString("Key2");
```

Intent : Implicit Intent

- These intents do not name a target and the field for the component name is left blank. Implicit intents are often used to activate components in other applications.

Example

Intent : Implicit Intent

- For example if we want to open a url in a browser than it will be implicit intent and we will not mentioned which activity will be responsible for this action
- Implicit Intent by specifying a URI

```
Intent i = new Intent(Intent.ACTION_VIEW,
Uri.parse("http://www.facebook.com"));
// Starts Implicit Activity
startActivity(i);
```

- Here we didn't specify the activity that will be responsible to open the url of facebook. So Android system will handle this action.

- If multiple activity is responsible to handle this action then a dialog will show to the user for the selection of appropriate activity.

MOBILE APPLICATION DEVELOPMENT
PART 3

PART 3 contents at a glance:

1. Working with Views(UI Widgets)-Button,
2. Toast, ToggleButton,
3. CheckBox, RadioButton,
4. Spinner,
5. WebView,
6. EditText,DatePicker,
7. TimePicker
8. Analog and Digital clock,
9. Handling UI events (onClick listeners etc)

Android Widgets:

There are given a lot of **android widgets** with simplified examples such as Button, EditText, AutoCompleteTextView, ToggleButton, DatePicker, TimePicker, ProgressBar etc.

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.

Working with button:

```
//code to display the sum of two numbers
package com.example.com.sum;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
  private EditText edittext1, edittext2;
  private Button buttonSum;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    addListenerOnButton();
  }
  public void addListenerOnButton() {
```

```
        edittext1 = (EditText) findViewById(R.id.editText1);
        edittext2 = (EditText) findViewById(R.id.editText2);
        buttonSum = (Button) findViewById(R.id.button);
        buttonSum.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View view) {
            String value1=edittext1.getText().toString();
            String value2=edittext2.getText().toString();
            int a=Integer.parseInt(value1);
            int b=Integer.parseInt(value2);
            int sum=a+b;
          Toast.makeText(getApplicationContext(),String.valueOf(sum), Toast.LENGTH_LONG).show();
          }
});  } }
```

Android Toast:

Andorid Toast can be used to display information for the short period of time. A toast contains message to be displayed quickly and disappears after sometime.

The android.widget.Toast class is the subclass of java.lang.Object class.

You can also create custom toast as well for example toast displaying image. You can visit next page to see the code for custom toast.

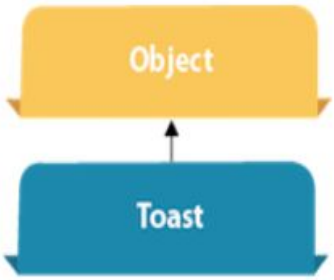
Toast class:

Toast class is used to show notification for a particular interval of time. After sometime it disappears. It doesn't block the user interaction.

Constants of Toast class:

There are only 2 constants of Toast class.

1. public static final int LENGTH_LONG : displays view for the long duration of time.
2. public static final int LENGTH_SHORT : displays view for the short duration of time.



Methods of Toast class:

1. public static Toast makeText(Context context, CharSequence text, int duration) : makes the toast containing text and duration.
2. public void show() : displays toast.
3. public void setMargin (float horizontalMargin, float verticalMargin) : changes the horizontal and vertical margin difference.

Working with Toast:

```
// code to display the toast
package com.example.toast;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //Displaying Toast with Hello  message
    Toast.makeText(getApplicationContext(),"Hello ",Toast.LENGTH_
SHORT).show();
  }
}
```

Android ToggleButton:

Android Toggle Button can be used to display checked/unchecked (On/Off) state on the button.

It is beneficial if user have to change the setting between two states. It can be used to On/Off Sound, Wifi, Bluetooth etc.

Since Android 4.0, there is another type of toggle button called *switch* that provides slider control.

Android ToggleButton and Switch both are the subclasses of CompoundButton class.

The 3 XML attributes of ToggleButton class.

1. android:disabledAlpha : Returns the text when button is not in the checked state.
2. android:textOff : The text for the button when it is not checked.
3. android:textOn : The text for the button when it is checked.

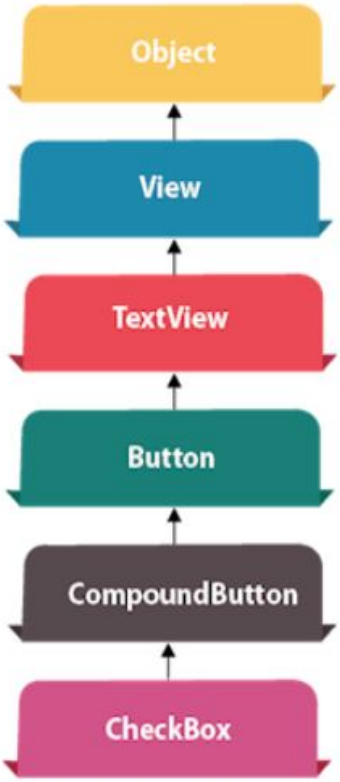
Methods of ToggleButton class :

1. CharSequence getTextOff() : Returns the text when button is not in the checked state.
2. CharSequence getTextOn() : Returns the text for when button is in the checked state.
3. void setChecked(boolean checked) : Changes the checked state of this button.

Working with ToggleButton:

```
// code to check which toggle button is ON/OFF
Package com.example.togglebutton;
import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import android.widget.ToggleButton;
public class MainActivity extends AppCompatActivity {
  private ToggleButton toggleButton1, toggleButton2;
  private Button buttonSubmit;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    addListenerOnButtonClick();
  }
  public void addListenerOnButtonClick(){
    //Getting the ToggleButton and Button instance from the layout x
ml file
    toggleButton1=(ToggleButton)findViewById(R.id.toggleButton);
    toggleButton2=(ToggleButton)findViewById(R.id.toggleButton2);
    buttonSubmit=(Button)findViewById(R.id.button);

    //Performing action on button click
    buttonSubmit.setOnClickListener(new View.OnClickListener(){
      @Override
      public void onClick(View view) {
        StringBuilder result = new StringBuilder();
        result.append("ToggleButton1 : ").append(toggleButton1.getT
ext());
        result.append("\nToggleButton2 : ").append(toggleButton2.ge
tText());
        //Displaying the message in toast
        Toast.makeText(getApplicationContext(), result.toString(),Toas
t.LENGTH_LONG).show();
      }
    }); } }
```



Android 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 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.

Methods of CheckBox class:

There are many inherited methods of View, TextView, and Button classes in the CheckBox class.

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

Android CheckBox Example:

```
//code to display checked items
Package com.example.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();
    }
});  } }
```

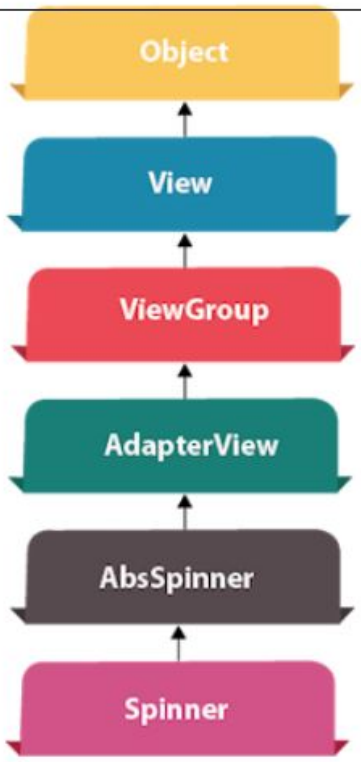
Android RadioButton:

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:

```
Package com.example.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();
    } } }
```



Android Spinner:

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 AbsSpinner class.

Android Spinner Example:

```
// code to display item on the spinner and perform event handling
package com.example.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  } }
```

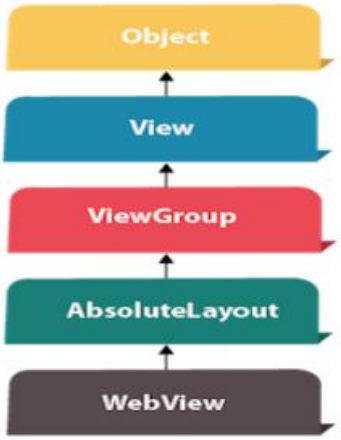
Android WebView:

Android WebView is used to display web page in android. The web page can be loaded from same application or URL. It is used to display online content in android activity.Android WebView uses webkit engine to display web page.

The android.webkit.WebView is the subclass of AbsoluteLayout class.

The **loadUrl()** and **loadData()** methods of Android WebView class are used to load and display web page.

Android WebView Example:




```
package com.example.webview;

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

public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    WebView mywebview = (WebView) findViewById(R.id.webView);
    String data = "<html><body><h1>Hello, Javatpoint!</h1></body></html>";
    mywebview.loadData(data, "text/html", "UTF-8");
```

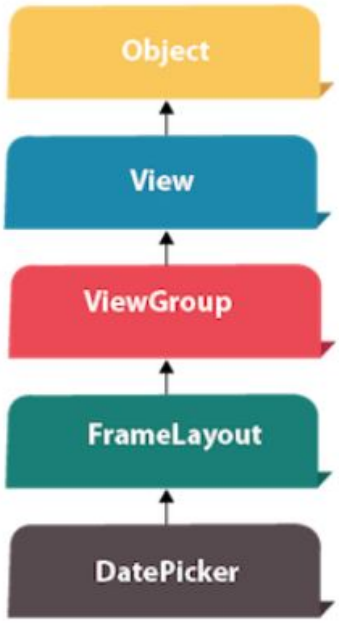
```
    mywebview.loadData(data,"UTF-8");
  } }
```

Android DatePicker:

Android DatePicker is a widget to select date. It allows you to select date by day, month and year.
Like DatePicker, android
also provides TimePicker to select time.
The android.widget.DatePicker is the subclass of FrameLayout class.

Android DatePicker Example:

```
Package com.example.datepicker;
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 displayDate;
   TextView textview1;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      textview1=(TextView)findViewById(R.id.textView1);
      picker=(DatePicker)findViewById(R.id.datePicker);
      displayDate=(Button)findViewById(R.id.button1);

      textview1.setText("Current Date: "+getCurrentDate());
```




```
    displayDate.setOnClickListener(new View.OnClickListener(){
        @Override
        public void onClick(View view) {

            textview1.setText("Change Date: "+getCurrentDate());
        } }); }
   public String getCurrentDate(){
```

```
    StringBuilder builder=new StringBuilder();;
    builder.append((picker.getMonth() + 1)+"/");//month is 0 based
    builder.append(picker.getDayOfMonth()+"/");
    builder.append(picker.getYear());
    return builder.toString();
  } }
```

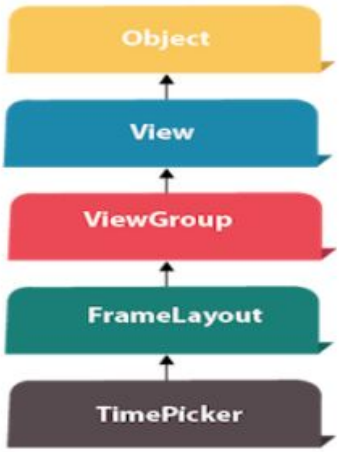
Android TimePicker:

Android TimePicker widget is used to select date. It allows you to select time by hour and minute. You cannot select time by seconds.

The android.widget.TimePicker is the subclass of FrameLayout class.

Android TimePicker Example:

```
package com.example.timepicker;
 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 {
  TextView textview1;
  TimePicker timepicker;
  Button changetime;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    textview1=(TextView)findViewById(R.id.textView1);
    timepicker=(TimePicker)findViewById(R.id.timePicker);
    //Uncomment the below line of code for 24 hour view
    timepicker.setIs24HourView(true);
    changetime=(Button)findViewById(R.id.button1);
    textview1.setText(getCurrentTime());
  changetime.setOnClickListener(new View.OnClickListener(){
      @Override
  public void onClick(View view) {
        textview1.setText(getCurrentTime());
      } }); }
  public String getCurrentTime(){
  String currentTime="Current Time: "+timepicker.getCurrentHour()+":"+timepicker.getCurrentMin
ute();
    return currentTime;
```



```
} }
```

Android Analog clock and Digital clock :

The **android.widget.AnalogClock** and **android.widget.DigitalClock** classes provides the functionality to display analog and digital clocks.

Android analog and digital clocks are used to show time in android application.

Android AnalogClock is the subclass of View class.

Android DigitalClock is the subclass of TextView class.

Working with analog and digital clocks:

Activity_main.xml:

Drag and drop analog and digital clocks onto the layout.

```
<AnalogClock
    android:id="@+id/analogClock1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginLeft="136dp"
    android:layout_marginTop="296dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
  <DigitalClock
    android:id="@+id/digitalClock1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/analogClock1"
    android:layout_centerHorizontal="true"
    android:layout_marginLeft="176dp"
    android:layout_marginTop="84dp"
    android:text="DigitalClock"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

MainActivity.java:

```
package com.example.analogdigital;
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);
  } }
```

Android EditText:

Android system supports EditText, which is a subclass of TextView supplied with text editing operations. We often use EditText in our Android applications in order to provide an input or text field, especially in forms.

EditText in layout file:

```
<EditText
    android:id="@+id/Username"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:ems="10"
    android:hint="Username"
   android:inputType=”text” />
```

EditText Example:

MainActivity.java:

```
package com.example.demo;
import android.os.Bundle;
import android.app.Activity;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends Activity {
  EditText eText;
  Button btn;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    eText = (EditText) findViewById(R.id.edittext);
    btn = (Button) findViewById(R.id.button);
    btn.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        String str = eText.getText().toString();
        Toast msg = Toast.makeText(getBaseContext(),str,Toast.LENGTH_LONG);
        msg.show();
      } });}}
```

MOBILE APPLICATION DEVELOPMENT
UNIT IV

Unit IV contents at a glance:

1. **Option menu,**
2. **Popup menu,**
3. **Working with images-ImageView,**
4. **ImageSwitcher,**
5. **AlertDialog,**
6. **Alarm manager,**
7. **SMS messaging,**
8. **Sending E-mail,**
9. **Media Player,**
10. **Using camera for taking pictures, recording video,**
11. **Handling Telephony Manager**

1. Option Menu:

Android Option Menus are the primary menus of android. They can be used for settings, search, delete item etc.

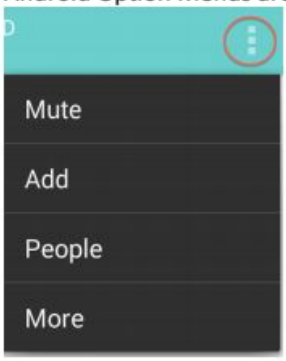


Figure: sample option menu

- **class: MenuInflater**
- **methods :inflate(),onCreateOptionsMenu(Menu menu)**
- **event handling methods: onOptionsItemSelected()**
- **Create a menu XML file**

The option menu items are to included in menu XML file.

Sample menu options are shown below:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="example.javatpoint.com.optionmenu.MainActivity">

    <item  android:id="@+id/item1"
        android:title="Item 1"/>
    <item  android:id="@+id/item2"
        android:title="Item 2"/>
    <item  android:id="@+id/item3"
        android:title="Item 3"
        app:showAsAction="withText"/>
```

```
13. </menu>
```

program to implement option menu:

You have to create a folder

res->menu->option_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
  <item android:id="@+id/search_item"
    android:title="Search" />
  <item android:id="@+id/upload_item"
    android:title="Upload" />
  <item android:id="@+id/copy_item"
    android:title="Copy" />
  <item android:id="@+id/print_item"
    android:title="Print" />
  <item android:id="@+id/share_item"
    android:title="Share" />
  <item android:id="@+id/bookmark_item"
    android:title="BookMark" />
</menu>
```

MainActivity.java

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.options_menu, menu);
    return true;
  }
  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    Toast.makeText(this, "Selected Item: " +item.getTitle(), Toast.LENGTH_SHORT).show();
    switch (item.getItemId()) {
      case R.id.search_item:
        // do your code
        return true;
      case R.id.upload_item:
```

```
            // do your code
            return true;
        case R.id.copy_item:
            // do your code
            return true;
        case R.id.print_item:
            // do your code
            return true;
        case R.id.share_item:
            // do your code
            return true;
        case R.id.bookmark_item:
            // do your code
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
  }
}
```

2. Context Menu:

A context menu is a **floating menu** that appears when the user performs a long-click on an element. It provides actions that affect the selected content or context frame.

The contextual action mode displays action items that affect the selected content in a bar at the top of the screen and allows the user to select multiple items.

- **Create a Listview and register the listview for context menu**
- **class: MenuInflater, ListView, ArrayAdapter**
- **methods :inflate(),onCreateContextMenu(Menu menu)**
- **event handling methods: onContextItemSelected()**
- **Create a menu XML file**

Sample program for context menu:

main_menu.xml

Create a separate menu_main.xml file in menu directory for menu items.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/call"
      android:title="Call" />
   <item android:id="@+id/sms"
      android:title="SMS" />
</menu>
```

Activity class

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
ListView listView;
String contacts[]={"Ajay","Sachin","Sumit","Tarun","Yogesh"};
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
listView=(ListView)findViewById(R.id.listView);
ArrayAdapter<String> adapter=new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,contacts);
listView.setAdapter(adapter);
// Register the ListView  for Context menu
registerForContextMenu(listView);
}
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo)
{
super.onCreateContextMenu(menu, v, menuInfo);
MenuInflater inflater = getMenuInflater();
inflater.inflate(R.menu.menu_main, menu);
menu.setHeaderTitle("Select The Action");
}
@Override
public boolean onContextItemSelected(MenuItem item){
if(item.getItemId()==R.id.call){
Toast.makeText(getApplicationContext(),"calling code",Toast.LENGTH_LONG).show();
}
else if(item.getItemId()==R.id.sms){
Toast.makeText(getApplicationContext(),"sending sms code",Toast.LENGTH_LONG).show();
```

```
}else{
return false;
}
return true;
}
}
```

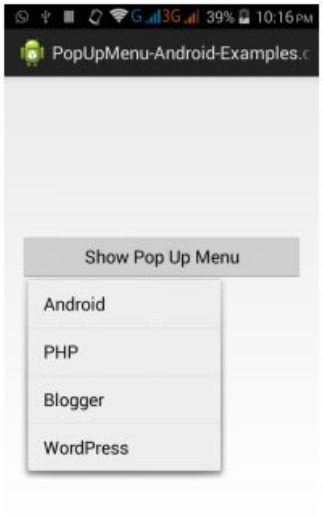
3. Popup Menu:

A popup menu **displays a list of items in a vertical list that's anchored to the view** that invoked the menu.

The android **Popup Menu provides an overflow style menu for action**s that are related to specific content.

The popup menu won't support any item shortcuts and item icons.

The **android.widget.PopupMenu** is the direct subclass of java.lang.Object class.



- **class: PopupMenu**
- **methods :inflate(),setOnMenuItemClickListener(MainActivity.this)**
- **event handling methods: onMenuItemClick(MenuItem item)**
- **Create a menu XML file**

Program:

create folder

re->menu->popup_menu.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
   <item android:id="@+id/search_item"
      android:title="Search" />
   <item android:id="@+id/upload_item"
      android:title="Upload" />
   <item android:id="@+id/copy_item"
      android:title="Copy" />
   <item android:id="@+id/print_item"
      android:title="Print" />
   <item android:id="@+id/share_item"
      android:title="Share" />
   <item android:id="@+id/bookmark_item"
      android:title="BookMark" />
</menu>
```

activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >

  <Button
    android:id="@+id/btnShow"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Show Popup Menu"
    android:layout_marginTop="200dp" android:layout_marginLeft="100dp"/>
</LinearLayout>
```

MainActivity.java

```
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.PopupMenu;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity implements PopupMenu.OnMenuItemClickListener {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Button btn = (Button) findViewById(R.id.btnShow);
    btn.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        PopupMenu popup = new PopupMenu(MainActivity.this, v);
        popup.setOnMenuItemClickListener(MainActivity.this);
        popup.inflate(R.menu.popup_menu);
        popup.show();
      }
    });
  }
  @Override
  public boolean onMenuItemClick(MenuItem item) {
    Toast.makeText(this, "Selected Item: " +item.getTitle(), Toast.LENGTH_SHORT).show();
    switch (item.getItemId()) {
      case R.id.search_item:
        // do your code
        return true;
      case R.id.upload_item:
        // do your code
        return true;
```

```
        case R.id.copy_item:
          // do your code
          return true;
        case R.id.print_item:
          // do your code
          return true;
        case R.id.share_item:
          // do your code
          return true;
        case R.id.bookmark_item:
          // do your code
          return true;
        default:
          return false;
      }
    }
}
```

4. ImageView:

In Android, ImageView class is used to display an image file in application.
The ImageView handles all the loading and scaling of the image for you. Note the **scaleType attribute** which defines how the images will be scaled to fit in your layout.
Some of them scaleTypes configuration properties are **center, center_crop, fit_xy, fitStart etc.**

IN ORDER TO USE IMAGES, FIRST ADD IMAGE TO RESOURCES AS SHOWN BELOW:

Put your images into folder "res/drawable-ldpi", "res/drawable-mdpi" or "res/drawable-hdpi".

Below is an ImageView code in XML:

Make sure to save lion image in drawable folder

```
<ImageView
android:id="@+id/simpleImageView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:src="@drawable/lion" />
```

Attributes of ImageView:

Now let's we discuss some important attributes that helps us to configure a ImageView in your xml file.

1. id: id is an attribute used to uniquely identify a image view in android. Below is the example code in which we set the id of a image view.

```
<ImageView
android:id="@+id/simpleImageView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
/>
```

2. src: src is an attribute used to set a source file or you can say image in your imageview to make your layout attractive.

Below is the example code in which we set the source of a imageview lion which is saved in drawable folder.

```
<ImageView
android:id="@+id/simpleImageView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:src="@drawable/lion" /><!--set the source of an image view-->
```

or

```
/*Add in Oncreate() funtion after setContentView()*/
ImageView simpleImageView=(ImageView) findViewById(R.id.simpleImageView);
simpleImageView.setImageResource(R.drawable.lion);//set the source in java class
```

3. background: background attribute is used to set the background of a ImageView. We can set a color or a drawable in the background of a ImageView.

Below is the example code in which we set the black color in the background and an image in the src attribute of image view.

```
<ImageView
  android:id="@+id/simpleImageView"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:src="@drawable/lion"
  android:background="#000"/><!--black color in background of a image view-->
```

4. padding: padding attribute is used to set the padding from left, right, top or bottom of the Imageview.

- paddingRight: set the padding from the right side of the image view.
- paddingLeft: set the padding from the left side of the image view.
- paddingTop: set the padding from the top side of the image view.
- paddingBottom: set the padding from the bottom side of the image view.
- padding: set the padding from the all side's of the image view.

Below is the example code of padding attribute in which we set the 30dp padding from all the side's of a image view.

```
<ImageView
   android:id="@+id/simpleImageView"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:background="#000"
   android:src="@drawable/lion"
   android:padding="30dp"/><!--set 30dp padding from all the sides-->
```

5. scaleType: scaleType is an attribute used to control how the image should be re-sized or moved to match the size of this image view. The value for scale type attribute can be fit_xy, center_crop, fitStart etc.

Below is the example code of scale type in which we set the scale type of image view to fit_xy.

```
<ImageView
android:id="@+id/simpleImageView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:src="@drawable/lion"
android:scaleType="fitXY"/><!--set scale type fit xy-->
```

5. Image Switcher:

Android image switcher provides an animation over images to transition from one image to another. In order to use image switcher, we need to implement ImageSwitcher component in .xml file.

The *setFactory()* method of ImageSwitcher provide implementation of *ViewFactory* interface. ViewFactory interface implements its unimplemented method and returns an ImageView.

component: ImageSwitcher
method: setFactory()
Interface:ViewFactory

syntax:

```
<ImageSwitcher
  android:id="@+id/imageSwitcher1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_centerHorizontal="true"
  android:layout_centerVertical="true" >
</ImageSwitcher>
```

Steps for Implementation of ImageSwitcher:

1. Get the reference of ImageSwitcher in class using findViewById() method, or you can also create an object dynamically.
2. Set a factory using switcherid.setFactory()
3. Set an IN-ANIMATION using switcherid.setInAnimation()
4. Set an OUT-ANIMATION using switcherid.setOutAnimation()

Methods:

1. setFactory(ViewFactory factory): This method is used to create a new view for ImageSwitcher. By using this method we create a new ImageView and replace the old view with that.

2. setImageDrawable(Drawable drawable): This method is used to set an image in ImageSwitcher. In this we pass an image for Drawable.

3.**loadAnimation(Context context, int id)**: This method is used whenever we need to define an object of Animation class through AnimationUtils class by calling a static method loadAnimation.

Below we create an object of Animation class and load an animation by using AnimationUtils class.

```
// load an animation by using AnimationUtils class
Animation in = AnimationUtils.loadAnimation(this, android.R.anim.slide_in_left);
 Animation out = AnimationUtils.loadAnimation(this, android.R.anim.slide_out_right);
```

4.setInAnimation(Animation inAnimation): This method is used to set the animation of the appearance of the object on the screen.

Below we create an object of Animation class and load an animation by using AnimationUtils class and then set the Animation on ImageSwitcher.

ImageSwitcher simpleImageSwitcher=(ImageSwitcher)findViewById(R.id. simpleImageSwitcher); // initiate a ImageSwitcher
Animation in = AnimationUtils.loadAnimation(this,android.R.anim.slide_in_left); // load an animation
simpleImageSwitcher.setInAnimation(in); // set in Animation for ImageSwitcher

5. **setOutAnimation(Animation outAnimation): T**his method does the opposite of setInAnimation().

Whenever we use set a new Image in ImageView, it first removes the old view using an animation set with the setOutAnimation() method, and then places the new one using the animation set by the setInAnimation() method.

Program for Image Switcher:

XML File:


```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">

   <ImageSwitcher
      android:id="@+id/simpleImageSwitcher"
      android:layout_width="match_parent"
      android:layout_height="200dp"
      android:layout_gravity="center_horizontal"
      android:layout_marginTop="50dp" />
```

```
    <Button
        android:id="@+id/buttonNext"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="150dp"
        android:layout_gravity="center"
        android:background="#050"
        android:textColor="#fff"
        android:textStyle="bold"
        android:text="NEXT" />

  </LinearLayout>
```

Activity File:

```
  package com.example.ip_d.imageswitcherexample;

  import android.os.Bundle;
  import android.support.v7.app.AppCompatActivity;
  import android.view.View;
  import android.view.animation.Animation;
  import android.view.animation.AnimationUtils;
  import android.widget.Button;
  import android.widget.ImageSwitcher;
  import android.widget.ImageView;
  import android.widget.LinearLayout;
  import android.widget.ViewSwitcher;

  public class MainActivity extends AppCompatActivity {
      private ImageSwitcher simpleImageSwitcher;
      Button btnNext;


      // Array of Image IDs to Show In ImageSwitcher
      int imageIds[] = {R.drawable.image1, R.drawable.images2, R.drawable.image3, R.drawable.images4,
  R.drawable.images5};
      int count = imageIds.length;
      // to keep current Index of ImageID array
      int currentIndex = -1;

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

          setContentView(R.layout.activity_main);

          // get The references of Button and ImageSwitcher
          btnNext = (Button) findViewById(R.id.buttonNext);
          simpleImageSwitcher = (ImageSwitcher) findViewById(R.id.simpleImageSwitcher);
          // Set the ViewFactory of the ImageSwitcher that will create ImageView object when asked
          simpleImageSwitcher.setFactory(new ViewSwitcher.ViewFactory() {
```

```
        public View makeView() {
            // TODO Auto-generated method stub

            // Create a new ImageView and set it's properties
            ImageView imageView = new ImageView(getApplicationContext());
            // set Scale type of ImageView to Fit Center
            imageView.setScaleType(ImageView.ScaleType.FIT_CENTER);
            // set the Height And Width of ImageView To FIll PARENT
            imageView.setLayoutParams(new        ImageSwitcher.LayoutParams(LinearLayout.LayoutParams.FILL_PARENT,
LinearLayout.LayoutParams.FILL_PARENT));
            return imageView;
        }
    });

    // Declare in and out animations and load them using AnimationUtils class
    Animation in = AnimationUtils.loadAnimation(this, android.R.anim.slide_in_left);
    Animation out = AnimationUtils.loadAnimation(this, android.R.anim.slide_out_right);

    // set the animation type to ImageSwitcher
    simpleImageSwitcher.setInAnimation(in);
    simpleImageSwitcher.setOutAnimation(out);


    // ClickListener for NEXT button
    // When clicked on Button ImageSwitcher will switch between Images
    // The current Image will go OUT and next Image  will come in with specified animation
    btnNext.setOnClickListener(new View.OnClickListener() {

        public void onClick(View v) {
            // TODO Auto-generated method stub
            currentIndex++;
            //  Check If index reaches maximum then reset it
            if (currentIndex == count)
                currentIndex = 0;
            simpleImageSwitcher.setImageResource(imageIds[currentIndex]); // set the image in ImageSwitcher
        }
    });

  }

}
```

6. AlertDialog:

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:

1. title,
2. message and
3. action buttons.

Android AlertDialog is the subclass of Dialog class.

1)In order to make an alert dialog, you need to make an object of AlertDialogBuilder which an inner class of AlertDialog. Its syntax is given below

AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);

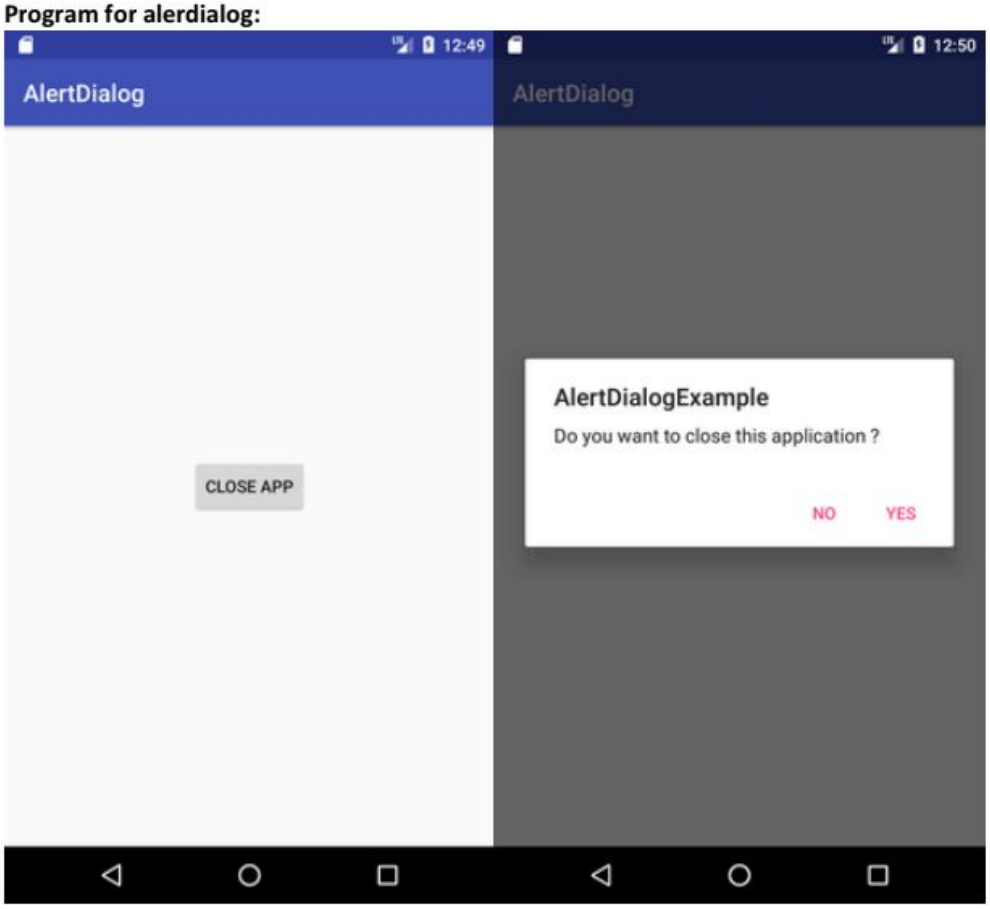
2)Now you have to set the positive (yes) or negative (no) button using the object of the AlertDialogBuilder class. Its syntax is

alertDialogBuilder.**setPositiveButton(**CharSequence text,
DialogInterface.OnClickListener listener)
alertDialogBuilder.**setNegativeButton(C**harSequence text,
DialogInterface.OnClickListener listener)

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.

Program for alerdialog:




```
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();
     }
   });
  }
}
```

7. AlarmManager:

Android AlarmManager allows you to access system alarm.

By the help of Android AlarmManager in android, you can *schedule your application to run at a specific time* in the future. It works whether your phone is running or not.

The Android AlarmManager holds a CPU wake lock that *provides guarantee not to sleep the phone* until broadcast is handled.

classes: AlarmManager, PendingIntent

Sample Program:

```
package example.com.alarmmanager;

import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
  Button start;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    start= findViewById(R.id.button);

    start.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        startAlert();
      }
    });
  }

  public void startAlert(){
    EditText text = findViewById(R.id.time);
    int i = Integer.parseInt(text.getText().toString());
    Intent intent = new Intent(this, MyBroadcastReceiver.class);
    PendingIntent pendingIntent = PendingIntent.getBroadcast(
        this.getApplicationContext(), 234324243, intent, 0);
    AlarmManager alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);
    alarmManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis()
        + (i * 1000), pendingIntent);
    Toast.makeText(this, "Alarm set in " + i + " seconds",Toast.LENGTH_LONG).show();
  }
}
```

Let's create BroadcastReceiver class that starts alarm.

File: MyBroadcastReceiver.java

```
package example.javatpoint.com.alarmmanager;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.media.MediaPlayer;
import android.widget.Toast;

public class MyBroadcastReceiver extends BroadcastReceiver {
  MediaPlayer mp;
  @Override
  public void onReceive(Context context, Intent intent) {
    mp=MediaPlayer.create(context, R.raw.alarm);
    mp.start();
    Toast.makeText(context, "Alarm....", Toast.LENGTH_LONG).show();
  }
}
```

File: AndroidManifest.xml

You need to provide a receiver entry in AndroidManifest.xml file.

```
1. <receiver android:name="MyBroadcastReceiver" >
2. </receiver>
```

8. SMS Messaging:
We can send sms in android via intent.
we have to class SmsManager.

class: SmsManager, PendingIntent
Methods: PendingIntent.getActivity(),SmsManager.getDefault(),sendTextMessage()

```
//Getting intent and PendingIntent instance
Intent intent=new Intent(getApplicationContext(),MainActivity.class);
PendingIntent pi=PendingIntent.getActivity(getApplicationContext(), 0, intent,0);

//Get the SmsManager instance and call the sendTextMessage method to send message
SmsManager sms=SmsManager.getDefault();
sms.sendTextMessage("8802177690", null, "hello javatpoint", pi,null);
```

Write the permission code in Android-Manifest.xml file

You need to write SEND_SMS permission as given below:

1. **<uses-permission android:name="android.permission.SEND_SMS"/>**

Program for sms:

```
package com.example.sendsms;
import android.os.Bundle;
import android.app.Activity;
import android.app.PendingIntent;
import android.content.Intent;
import android.telephony.SmsManager;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

EditText mobileno,message;
Button sendsms;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

mobileno=(EditText)findViewById(R.id.editText1);
message=(EditText)findViewById(R.id.editText2);
sendsms=(Button)findViewById(R.id.button1);

//Performing action on button click
sendsms.setOnClickListener(new OnClickListener() {

@Override
```

```
public void onClick(View arg0) {
String no=mobileno.getText().toString();
String msg=message.getText().toString();

//Getting intent and PendingIntent instance
Intent intent=new Intent(getApplicationContext(),MainActivity.class);
PendingIntent pi=PendingIntent.getActivity(getApplicationContext(), 0, intent,0);

//Get the SmsManager instance and call the sendTextMessage method to send message
SmsManager sms=SmsManager.getDefault();
sms.sendTextMessage(no, null, msg, pi,null);

Toast.makeText(getApplicationContext(), "Message Sent successfully!",
Toast.LENGTH_LONG).show();
}
});
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
// Inflate the menu; this adds items to the action bar if it is present.
getMenuInflater().inflate(R.menu.activity_main, menu);
return true;
}

}
```

9.Sending E-Mail:
We can easily send email in android via intent.

sample code for sending email:

```
Intent email = new Intent(Intent.ACTION_SEND);
email.putExtra(Intent.EXTRA_EMAIL, new String[]{ to});
email.putExtra(Intent.EXTRA_SUBJECT, subject);
email.putExtra(Intent.EXTRA_TEXT, message);

//need this to prompts email client only
email.setType("message/rfc822");

startActivity(Intent.createChooser(email, "Choose an Email client :"));
```

complete program for email:

```
package com.example.sendemail;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends Activity {
  EditText editTextTo,editTextSubject,editTextMessage;
  Button send;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    editTextTo=(EditText)findViewById(R.id.editText1);
    editTextSubject=(EditText)findViewById(R.id.editText2);
    editTextMessage=(EditText)findViewById(R.id.editText3);

    send=(Button)findViewById(R.id.button1);

    send.setOnClickListener(new OnClickListener(){

      @Override
      public void onClick(View arg0) {
        String to=editTextTo.getText().toString();
        String subject=editTextSubject.getText().toString();
        String message=editTextMessage.getText().toString();


        Intent email = new Intent(Intent.ACTION_SEND);
         email.putExtra(Intent.EXTRA_EMAIL, new String[]{ to});
         email.putExtra(Intent.EXTRA_SUBJECT, subject);
         email.putExtra(Intent.EXTRA_TEXT, message);
```

```
            //need this to prompts email client only
            email.setType("message/rfc822");

            startActivity(Intent.createChooser(email, "Choose an Email client :"));

        }

    });
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.activity_main, menu);
    return true;
  }

}
```

10. Media Player:
We can play and control the audio files in android by the help of MediaPlayer class.

MediaPlayer class

The android.media.MediaPlayer class is used to control the audio or video files.

Methods of MediaPlayer class

There are many methods of MediaPlayer class. Some of them are as follows:

Method	Description
public void setDataSource(String path)	sets the data source (file path or http url) to use.
public void prepare()	prepares the player for playback synchronously.
public void start()	it starts or resumes the playback.
public void stop()	it stops the playback.
public void pause()	it pauses the playback.
public boolean isPlaying()	checks if media player is playing.
public void seekTo(int millis)	seeks to specified time in miliseconds.
public void setLooping(boolean looping)	sets the player for looping or non-looping.
public boolean isLooping()	checks if the player is looping or non-looping.
public void selectTrack(int index)	it selects a track for the specified index.
public int getCurrentPosition()	returns the current playback position.
public int getDuration()	returns duration of the file.
public void setVolume(float leftVolume,float rightVolume)	sets the volume on this player.

sample program for mediaplayer(audio):

```
package com.example.audiomediaplayer1;

import android.media.MediaPlayer;
import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.MediaController;
import android.widget.VideoView;

public class MainActivity extends Activity {

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

MediaPlayer mp=new MediaPlayer();
try{
mp.setDataSource("/sdcard/Music/maine.mp3");//Write your location here  or copy audio file into raw folder under res
folder
```

```
mp.prepare();
mp.start();

}catch(Exception e){e.printStackTrace();}

}
```

11. Camera:

Camera is mainly used to capture picture and video. We can control the camera by using methods of camera api.

Android provides the facility to work on camera by 2 ways:

1. By Camera Intent
2. By Camera API

Understanding basic classes of Camera Intent and API

Intent

By the help of 2 constants of MediaStore class, we can capture picture and video without using the instance of Camera class.

1. ACTION_IMAGE_CAPTURE (for image capturing)
2. ACTION_VIDEO_CAPTURE (for video capturing)

Camera

It is main class of camera api, that can be used to take picture and video.

SurfaceView

It represents a surface view ore preview of live camera.

MediaRecorder

It is used to record video using camera. It can also be used to record audio files as we have seen in the previous example of media framework.

Sample Program for camera:


```
package com.example.simplecamera;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;

public class MainActivity extends Activity {
private static final int CAMERA_REQUEST = 1888;
ImageView imageView;
public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

imageView = (ImageView) this.findViewById(R.id.imageView1);
Button photoButton = (Button) this.findViewById(R.id.button1);

photoButton.setOnClickListener(new View.OnClickListener() {

@Override
public void onClick(View v) {
Intent cameraIntent = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(cameraIntent, CAMERA_REQUEST);
}
});
}

protected void onActivityResult(int requestCode, int resultCode, Intent data) {
```

```
if (requestCode == CAMERA_REQUEST) {
Bitmap photo = (Bitmap) data.getExtras().get("data");
imageView.setImageBitmap(photo);
}
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
// Inflate the menu; this adds items to the action bar if it is present.
getMenuInflater().inflate(R.menu.activity_main, menu);
return true;
}

}
```

12. Telephony Manager:

The android.telephony.TelephonyManager class provides information about the telephony services such as subscriber id,

- sim serial number,
- phone network type etc.
- Moreover, you can determine the phone state etc.

Class:

```
TelephonyManager  tm=(TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
```

Methods:

```
getDeviceId()
getSimSerialNumber()
getNetworkCountryIso()
getSimCountryIso()
getDeviceSoftwareVersion()
getVoiceMailNumber()
```

Sample Program for telephony manager:

```
package com.telephonymanager;

import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.telephony.TelephonyManager;
import android.view.Menu;
import android.widget.TextView;

public class MainActivity extends Activity {
  TextView textView1;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      textView1=(TextView)findViewById(R.id.textView1);

      //Get the instance of TelephonyManager
      TelephonyManager  tm=(TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
```

```
        //Calling the methods of TelephonyManager the returns the information
        String IMEINumber=tm.getDeviceId();
        String subscriberID=tm.getDeviceId();
        String SIMSerialNumber=tm.getSimSerialNumber();
        String networkCountryISO=tm.getNetworkCountryIso();
        String SIMCountryISO=tm.getSimCountryIso();
        String softwareVersion=tm.getDeviceSoftwareVersion();
        String voiceMailNumber=tm.getVoiceMailNumber();

        //Get the phone type
        String strphoneType="";

        int phoneType=tm.getPhoneType();

        switch (phoneType)
        {
            case (TelephonyManager.PHONE_TYPE_CDMA):
                    strphoneType="CDMA";
                      break;
            case (TelephonyManager.PHONE_TYPE_GSM):
                    strphoneType="GSM";
                      break;
            case (TelephonyManager.PHONE_TYPE_NONE):
                     strphoneType="NONE";
                       break;
         }

        //getting information if phone is in roaming
        boolean isRoaming=tm.isNetworkRoaming();

        String info="Phone Details:\n";
        info+="\n IMEI Number:"+IMEINumber;
        info+="\n SubscriberID:"+subscriberID;
        info+="\n Sim Serial Number:"+SIMSerialNumber;
        info+="\n Network Country ISO:"+networkCountryISO;
        info+="\n SIM Country ISO:"+SIMCountryISO;
        info+="\n Software Version:"+softwareVersion;
        info+="\n Voice Mail Number:"+voiceMailNumber;
        info+="\n Phone Network Type:"+strphoneType;
        info+="\n In Roaming? :"+isRoaming;

        textView1.setText(info);//displaying the information in the textView
    }


}
```


8:47
TelephonyManager
Phone Details:
IMEI Number:000000000000000
SubscriberID:000000000000000
Sim Serial
Number:89014103211118510720
Network Country ISO:us
SIM Country ISO:us
Software Version:null
Voice Mail Number:+15552175049
Phone Network Type:GSM
In Roaming? :false

MOBILE APPLICATION DEVELOPMENT
UNIT V

Unit V contents at a glance:

1. **Storing the data persistently-Introducing the Data Storage Options:**
   - **The preferences,**
   - **The Internal Storage,**
   - **The External Storage,**
   - **The Content Provider**
2. **The SQLite database(CRUD operations-Insert,delete,update,select),**
3. **Connecting with the SQLite database**
4. **operations-Insert, Delete, Update, Fetch,**
5. **Publishing android applications-preparing for publishing,**
6. **Deploying APK files**

1.Storing the data persistently-Introducing the Data Storage Options:
Android provides many ways of storing data of an application. They are as given below:

1. The Shared preferences,
2. The Internal Storage,
3. The External Storage,
4. The Content Provider

1. The Shared preferences:
Android shared preference is used to store and retrieve primitive information. In android, string, integer, long, number etc. are considered as primitive data type. Shared Preferences allow you to save and retrieve data in the form of key, value pair.
In order to use shared preferences, you have to **call a method getSharedPreferences()** that returns a SharedPreference instance pointing to the file that contains the values of preferences.

SharedPreferences sharedpreferences = getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);

The **first parameter is the key** and the **second parameter is the MODE**

MODE_PRIVATE By setting this mode, the file can only be accessed using calling application
MODE_WORLD_READABLE This mode allow other application to read the preferences
MODE_WORLD_WRITEABLE This mode allow other application to write the preferences

SharedPreferences can be implemented by using either **Preferences Activity class file or using EDITOR class(getsharedPreferences())

SAVING DATA IN SHARED PREFERENCES:

1. We can save something in the sharedPreferences by using **SharedPreferences.Editor** class.
2. call edit() of SharedPreference instance and will receive it in an editor object.

```
Editor editor = sharedpreferences.edit();
editor.putString("key", "value");
editor.commit();
```

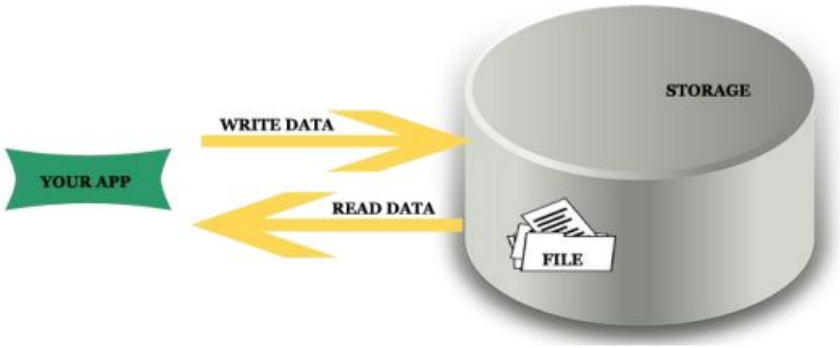
important methods:

[I] putInt(): to save primitive type data
[II] commit(): To save the values kept in primitive functions
[III] getInt(): To get saved Integer value
[IV] remove(String key): To remove any key based data from preference file
[V] clear(): To clear all data

sample code for Saving and Retrieving Data in SharedPreference:

```
public static final String PREFS_GAME ="com.androidv.GamePlay";
public static final String GAME_SCORE= "GameScore";

//======== Code to save data ==================

SharedPreferences sp = getSharedPreferences(PREFS_GAME ,Context.MODE_PRIVATE);
sp.edit().putInt(GAME_SCORE,100).commit();

//========= Code to get saved/ retrieve data ==============

SharedPreferences sp = getSharedPreferences(PREFS_GAME ,Context.MODE_PRIVATE);
int sc  = sp.getInt(GAME_SCORE,0);
Log.d("sachin","achieved score  is "+ sc);
```

2. Internal Storage:



For reading or writing data, the following classes and methods are used:

1. FileInputStream[provides methods for reading files-read() etc]
2. FileOutputStream[provides methods for writing files-write() etc]
3. openFileInput(file)[used to open a file and read it]
4.openFileOutput(file)[used to create and save a file]

- Android Internal storage is the storage of the private data on the device memory. The stored data in memory is allowed to **read and write files.**
- When files are stored in internal storage these file can only be accessed by the application itself not by other applications.
- These files in storage exist till the application stays over the device, as you uninstall associated files get removed automatically.
- The files are stored in directory data/data which is followed by the application package name.
- User can explicitly grant the permission to other apps to access files.
- To make the data private i.e you can use MODE_PRIVATE as discussed below about the modes.
- Technique is best suited when data can only be access by the application neither by the user nor by other apps.
- The data is stored in a file which contains data in bit format so it's required to convert data before adding it to a file or before extracting from a file.

Modes of Internal Storage:

MODE_PRIVATE — In private mode the data stored earlier is always overridden by the current data i.e every time you try to commit a new write to a file which removes or override the previous content.
MODE_APPEND — In this mode the data is append to the existing content i.e keep adding data.

openFileOutput(): This method is used to create and save a file. Its syntax is given below:

```
FileOutputStream fOut = openFileOutput("file name",Context.MODE_PRIVATE);
```

The method openFileOutput() returns an instance of FileOutputStream.

openFileInput(): This method is used to open a file and read it. It returns an instance of FileInputStream. Its syntax is given below:
FileInputStream fin = openFileInput(file);

Write data to file in Internal Storage(write()):

- Define the filename as string and also define data you wanna write to file as string or in any format generated from app or any other source.
- Use FileOutputStream method by creating its object as defined.
- Convert data into byte stream before writing over file because file accepts only byte format further close the file using file object.

```
String File_Name= "Demo.txt"; //gives file name
String Data="Hello!!"; //define data

FileOutputStream fileobj = openFileOutput( File_Name, Context.MODE_PRIVATE);
byte[] ByteArray = Data.getBytes(); //Converts into bytes stream
fileobj.write(ByteArray); //writing to file
fileobj.close(); //File closed
```

Reading file(read())

In order to read from the file you just created , call the openFileInput() method with the name of the file. It returns an instance of FileInputStream. Its syntax is given below –

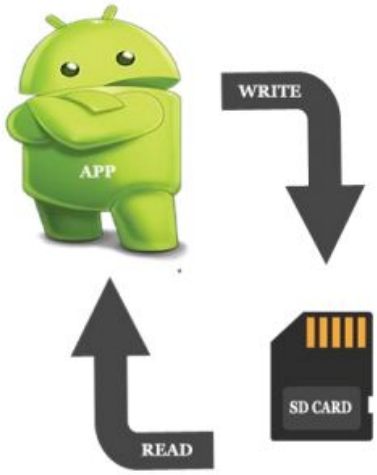
```
FileInputStream fin = openFileInput(file);
```

you can call read method to read one character at a time from the file and then you can print it. Its syntax is given below –

```
int c;
String temp="";
while( (c = fin.read()) != -1){
  temp = temp + Character.toString((char)c);
}

//string temp contains all the data of the file.
fin.close();
```

3. External Storage:

The data is stored in a file specified by the user itself and user can access these file. These files are only accessible till the application exits or you have SD card mounted on your device.



we can use the **following classes and methods for reading or writing data using SD card:**

1. **FileInputStream**[provides methods for reading files-read() etc]
2. **FileOutputStream**[provides methods for writing files-write() etc]
3. **isExternalStorageAvailable()**

4.isExternalStorageReadOnly()

5. **getExternalStorageState()** is a static method of Environment to determine if external storage is presently available or not.

Note: It is necessary to add external storage the permission to read and write. For that you need to **add permission in android Manifest file.**

Open AndroidManifest.xml file and **add permissions t**o it just after the package name.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

External storage such as SD card can also store application data, there's no security enforced upon files you save to the external storage.

In general there are two types of External Storage:

- **Primary External Storage**: In built shared storage which is "accessible by the user by plugging in a USB cable and mounting it as a drive on a host computer". Example: When we say Nexus 5 32 GB.
- **Secondary External Storage**: Removable storage. Example: SD Card

getExternalStorageState() is a static method of Environment to determine if external storage is presently available or not.

Methods to Store Data In Android:

- getExternalStorageDirectory() – Older way to access external storage in API Level less than 7. It is absolute now and not recommended. It directly get the reference to the root directory of your external storage or SD Card.
- getExternalFilesDir(String type) – It is recommended way to enable us to create private files specific to app and files are removed as app is uninstalled. Example is app private data.

- getExternalStoragePublicDirectory() : This is current recommended way that enable us to keep files public and are not deleted with the app uninstallation.

sample code to check external storage available or not:

```
if (!isExternalStorageAvailable() || isExternalStorageReadOnly()) {
  saveButton.setEnabled(false);
 }

private static boolean isExternalStorageReadOnly() {
 String extStorageState = Environment.getExternalStorageState();
 if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(extStorageState)) {
  return true;
 }
 return false;
}

 private static boolean isExternalStorageAvailable() {
 String extStorageState = Environment.getExternalStorageState();
 if (Environment.MEDIA_MOUNTED.equals(extStorageState)) {
  return true;
 }
 return false;
}
```

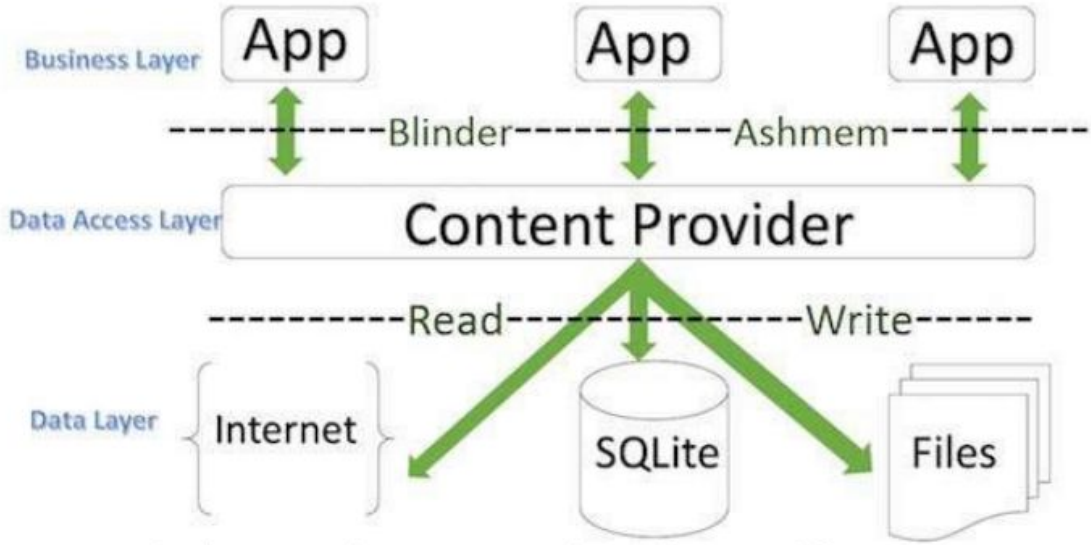
4. Content Providers:

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class.

It is a component which

i)- hides database details (database name, table name, column info. etc.)

ii)- allows the application to share data among multiple applications.



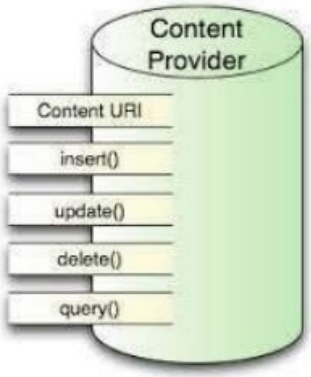
Content providers let you centralize content in one place and have many different applications access it as needed.

data is stored in an SQlite database

A content provider is implemented as a subclass of ContentProvider class

```
public class My Application extends  ContentProvider
{
}
```

list of methods which you need to override in Content Provider class:



- onCreate() This method is called when the provider is started.
- query() This method receives a request from a client. The result is returned as a Cursor object.
- insert()This method inserts a new record into the content provider.
- delete() This method deletes an existing record from the content provider.
- update() This method updates an existing record from the content provider.
- getType() This method returns the MIME type of the data at the given URI.

- Steps to create Content Provider:

  i)- Create ContentProvider subclass (android.ContentProvider).
  ii)- Register ContentProvider in AndroidManifest.xml file using provider element. (<provider>).
  To Register ContentProvider use following attributes:
  i)- android:name -> ContentProvider subclass name.
  ii)- android:authorities -> authority used for ContentProvider
  iii)- android:exported -> true/false.
  true: ContentProvider can be used in other applications.
  false: ContentProvider can be used in local application.
  By default, it is "true".

Content URIs

To query a content provider, you specify the query string in the form of a URI which has following format –

<prefix>://<authority>/<data_type>/<id>

SQLite Database:

SQLite is a opensource SQL database that stores data to a text file on a device.
SQLite is an open-source relational database i.e. used to perform database operations on android devices such as storing, manipulating or retrieving persistent data from the database.

SQLiteDatabase class

It contains methods to be performed on sqlite database such as create, update, delete, select etc.

Methods of SQLiteDatabase class

Method	Description
void execSQL(String sql)	executes the sql query not select query.
long insert(String table, String nullColumnHack, ContentValues values)	inserts a record on the database. The table specifies the table name, nullColumnHack doesn't allow completely null values. If second argument is null, android will store null values if values are empty. The third argument specifies the values to be stored.
int update(String table, ContentValues values, String whereClause, String[] whereArgs)	updates a row.
Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)	returns a cursor over the resultset.

Database - Package

The main package is android.database.sqlite that contains the classes to manage your own databases

Database - Creation

In Android, the SQLiteDatabase namespace defines the functionality to connect and manage a database. It provides functionality to create, delete, manage and display database content.

Create a Database or connecting to SQLite database:

Simple steps to create a database and handle are as following.

1. Create "SQLiteDatabase" object.
2. Open or Create database and create connection.
3. Perform insert, update or delete operation.
4. Create Cursor to display data from table of database.
5. Close the database connectivity.

Following tutorial helps you to create database and insert records in it.

Step 1: Instantiate "SQLiteDatabase" object

```
SQLiteDatabase db;
```

Before you can use the above object, you must import the android.database.sqlite.SQLiteDatabase namespace in your application.

```
db=openOrCreateDatabase(String path, int mode, SQLiteDatabase.CursorFactory factory)
```

This method is used to create/open database. As the name suggests, it will open a database connection if it is already there, otherwise it will create a new one.

Example,

```
db=openOrCreateDatabase("XYZ_Database",SQLiteDatabase.CREATE_IF_NECESSARY,null);
```

Step 2:Database table creation:

```
db.execSQL("CREATE TABLE IF NOT EXISTS student(Username VARCHAR,Password VARCHAR);");
```

Step 3:Database - Insertion

we can create table or insert data into table using execSQL method defined in SQLiteDatabase class. Its syntax is given below

```
db.execSQL("CREATE TABLE IF NOT EXISTS student(Username VARCHAR,Password VARCHAR);");
db.execSQL("INSERT INTO student VALUES('admin','admin');");
```

execSQL(String sql, Object[] bindArgs)

This method not only insert data , but also used to update or modify already existing data in database using bind arguments

Step 4:Database - Fetching (creation of cursor)

We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called rawQuery and it will return a resultset with the cursor pointing to the table.

```
Cursor resultSet = mydatbase.rawQuery("Select * from TutorialsPoint",null);
resultSet.moveToFirst();
String username = resultSet.getString(0);
String password = resultSet.getString(1);
```

Database - Helper class

For managing all the operations related to the database , an helper class has been given and is called SQLiteOpenHelper.

```
public class DBHelper extends SQLiteOpenHelper {
  public DBHelper(){
    super(context,DATABASE_NAME,null,1);
  }
  public void onCreate(SQLiteDatabase db) {}
  public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}
}
```

Program for implementing database operations using SQLiteDatabase class:

activity_main.xml

```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_x="50dp"
      android:layout_y="20dp"
      android:text="Student Details"
      android:textSize="30sp" />

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_x="20dp"
      android:layout_y="110dp"
      android:text="Enter Rollno:"
      android:textSize="20sp" />

   <EditText
      android:id="@+id/Rollno"
      android:layout_width="150dp"
      android:layout_height="wrap_content"
      android:layout_x="175dp"
      android:layout_y="100dp"
```

```
    android:inputType="number"
    android:textSize="20sp" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="20dp"
    android:layout_y="160dp"
    android:text="Enter Name:"
    android:textSize="20sp" />

<EditText
    android:id="@+id/Name"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:layout_x="175dp"
    android:layout_y="150dp"
    android:inputType="text"
    android:textSize="20sp" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="20dp"
    android:layout_y="210dp"
    android:text="Enter Marks:"
    android:textSize="20sp" />

<EditText
    android:id="@+id/Marks"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:layout_x="175dp"
    android:layout_y="200dp"
    android:inputType="number"
    android:textSize="20sp" />

<Button
    android:id="@+id/Insert"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:layout_x="25dp"
    android:layout_y="300dp"
    android:text="Insert"
    android:textSize="30dp" />

<Button
    android:id="@+id/Delete"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:layout_x="200dp"
    android:layout_y="300dp"
    android:text="Delete"
```

```
    android:textSize="30dp" />

  <Button
    android:id="@+id/Update"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:layout_x="25dp"
    android:layout_y="400dp"
    android:text="Update"
    android:textSize="30dp" />

  <Button
    android:id="@+id/View"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:layout_x="200dp"
    android:layout_y="400dp"
    android:text="View"
    android:textSize="30dp" />

  <Button
    android:id="@+id/ViewAll"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    android:layout_x="100dp"
    android:layout_y="500dp"
    android:text="View All"
    android:textSize="30dp" />

</AbsoluteLayout>
```

MainActivity.java

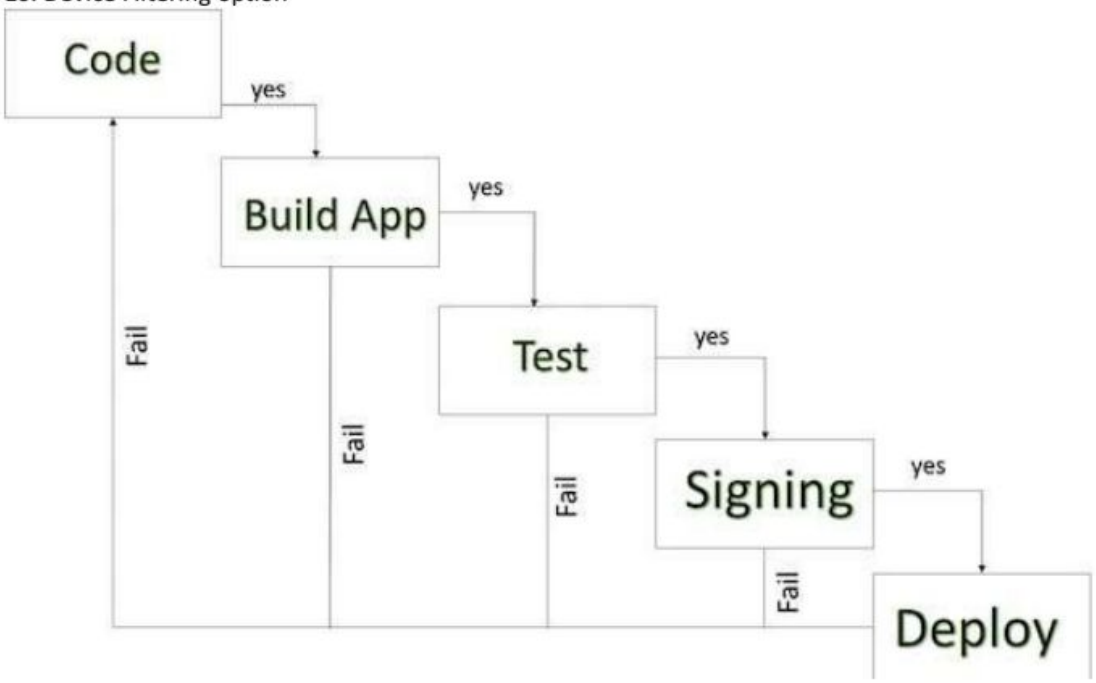
```
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;


public class MainActivity extends AppCompatActivity implements android.view.View.OnClickListener {

  EditText Rollno,Name,Marks;
  Button Insert,Delete,Update,View,ViewAll;
  SQLiteDatabase db;
```

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);

   Rollno=(EditText)findViewById(R.id.Rollno);
   Name=(EditText)findViewById(R.id.Name);
   Marks=(EditText)findViewById(R.id.Marks);
   Insert=(Button)findViewById(R.id.Insert);
   Delete=(Button)findViewById(R.id.Delete);
   Update=(Button)findViewById(R.id.Update);
   View=(Button)findViewById(R.id.View);
   ViewAll=(Button)findViewById(R.id.ViewAll);

   Insert.setOnClickListener(this);
   Delete.setOnClickListener(this);
   Update.setOnClickListener(this);
   View.setOnClickListener(this);
   ViewAll.setOnClickListener(this);

   // Creating database and table
   db=openOrCreateDatabase("StudentDB", Context.MODE_PRIVATE, null);
   db.execSQL("CREATE TABLE IF NOT EXISTS student(rollno VARCHAR,name VARCHAR,marks VARCHAR);");
}
public void onClick(View view)
{
   // Inserting a record to the Student table
   if(view==Insert)
   {
     db.execSQL("INSERT INTO student VALUES('"+Rollno.getText()+"','"+Name.getText()+
"','"+Marks.getText()+"');");

   }
   // Deleting a record from the Student table
   if(view==Delete)
   {

     Cursor c=db.rawQuery("SELECT * FROM student WHERE rollno='"+Rollno.getText()+"'", null);
     if(c.moveToFirst())
     {
       db.execSQL("DELETE FROM student WHERE rollno='"+Rollno.getText()+"'");
      }

   }
   // Updating a record in the Student table
   if(view==Update)
   {
      Cursor c=db.rawQuery("SELECT * FROM student WHERE rollno='"+Rollno.getText()+"'", null);
     if(c.moveToFirst())
         {
       db.execSQL("UPDATE student SET name='" + Name.getText() + "',marks='" + Marks.getText() +
```

```
                "' WHERE rollno='"+Rollno.getText()+"'");
            }
          }
      // Display a record from the Student table
      if(view==View)
      {
        Cursor c=db.rawQuery("SELECT * FROM student WHERE rollno='"+Rollno.getText()+"'", null);
        if(c.moveToFirst())
        {
          Name.setText(c.getString(1));
          Marks.setText(c.getString(2));
        }
       }
      // Displaying all the records
      if(view==ViewAll)
      {
        Cursor c=db.rawQuery("SELECT * FROM student", null);
        if(c.getCount()==0)
        {

 return;
        }
        StringBuffer buffer=new StringBuffer();
        while(c.moveToNext())
        {
          buffer.append("Rollno: "+c.getString(0)+"\n");
          buffer.append("Name: "+c.getString(1)+"\n");
          buffer.append("Marks: "+c.getString(2)+"\n\n");
        }

}
   }

}
```

Publishing android applications-preparing for publishing :

Generate signed APK and do Google play registration,do the following steps:
1. Create an account
2. Familiarize yourself with Developer Console
3. Fill in the necessary account details
4. Link your merchant account
5. Upload your app
6. Alpha and beta testing of app
7. Provide details for store listing
8. Add pricing and distribution details
9. Publishing the application
10. Device Filtering option



Deploying APK files:

Once you have signed your APK files, you need a way to get them onto your users' devices. The following sections describe the various ways to deploy your APK files. Three methods are covered:

- Deploying manually using the adb.exe tool
- Hosting the application on a web server
- Publishing through the Android Market

Besides these methods, you can install your applications on users' devices through e-mails, SD card, and so on. As long as you can transfer the APK file onto the user's device, you can install the application.

SharedPreferences Sample program:

Java File:

```
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
  EditText ed1,ed2,ed3;
  Button b1;

  public static final String MyPREFERENCES = "MyPrefs" ;
  public static final String Name = "nameKey";
  public static final String Phone = "phoneKey";
  public static final String Email = "emailKey";

  SharedPreferences sharedpreferences;

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

    ed1=(EditText)findViewById(R.id.editText);
    ed2=(EditText)findViewById(R.id.editText2);
    ed3=(EditText)findViewById(R.id.editText3);

    b1=(Button)findViewById(R.id.button);
    sharedpreferences = getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);

    b1.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        String n  = ed1.getText().toString();
        String ph  = ed2.getText().toString();
        String e  = ed3.getText().toString();

        SharedPreferences.Editor editor = sharedpreferences.edit();

        editor.putString(Name, n);
        editor.putString(Phone, ph);
        editor.putString(Email, e);
        editor.commit();
        Toast.makeText(MainActivity.this,"Thanks",Toast.LENGTH_LONG).show();
      }
    });
  }
```

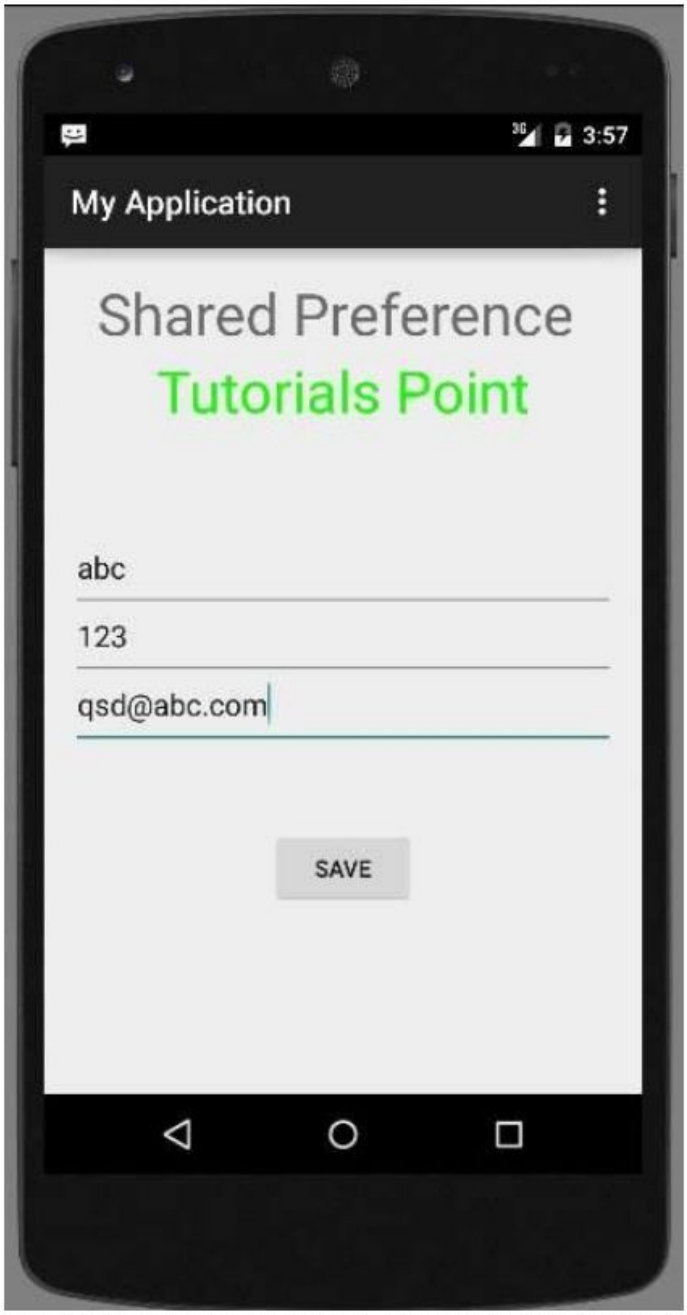
```
}
```

XML file:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
  android:layout_height="match_parent" android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Shared Preference "
    android:id="@+id/textView"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="35dp" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials Point"
    android:id="@+id/textView2"
    android:layout_below="@+id/textView"
    android:layout_centerHorizontal="true"
    android:textSize="35dp"
    android:textColor="#ff16ff01" />

  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:layout_below="@+id/textView2"
    android:layout_marginTop="67dp"
    android:hint="Name"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText2"
    android:layout_below="@+id/editText"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:hint="Pass" />
```

```
<EditText
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/editText3"
  android:layout_below="@+id/editText2"
  android:layout_alignParentLeft="true"
  android:layout_alignParentStart="true"
  android:layout_alignParentRight="true"
  android:layout_alignParentEnd="true"
  android:hint="Email" />

<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Save"
  android:id="@+id/button"
  android:layout_below="@+id/editText3"
  android:layout_centerHorizontal="true"
  android:layout_marginTop="50dp" />

</RelativeLayout>
```

output:



Now when you press save button, the text will be saved in the shared preferences. Now press back button and exit the application.