JSPM’Narhe Technical Campus
Master of Computer Applications(MCA)
Subject Name:-Mobile Application Development - (IT-31)
MCA-I SEM-II

Assignment No-1

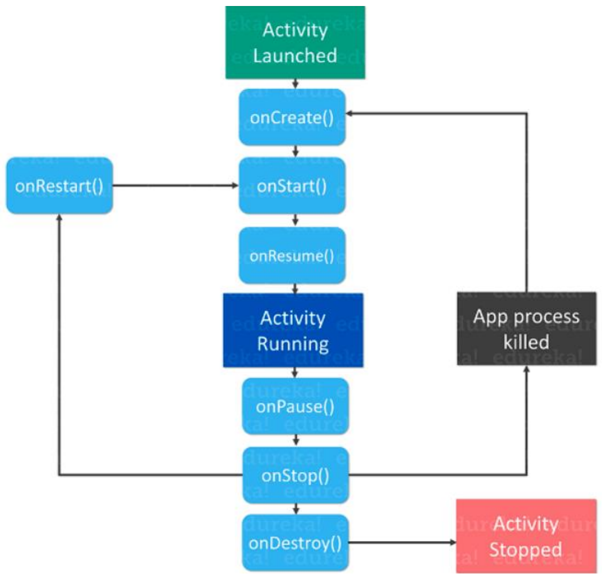
Roll no :- 69

Q1.Explain Android life cycle in details?

Android is the open-source operating system which is useful for many tasks. When you start or open your Android Application, it will undergo various states and that is called as Android Activity Life Cycle. Android is an open-source operating system which is based on Linux with a Java programming interface for mobile devices like Smartphones.As a user navigates through the app, Activity instances in your app transition through different stages in their life-cycle. The Activity class provides a number of callbacks that allow the activity to know that a state has changed: that the system is creating, stopping, or resuming an activity, or destroying the process in which the activity resides.

In general, activity lifecycle has seven callback methods:

1. **onCreate()**
2. **onStart()**
3. **onResume()**
4. **onPause()**
5. **onStop()**
6. **onRestart()**
7. **onDestroy()**



1. onCreate(): In this state, the activity is created.
2. onStart(): This callback method is called when the activity becomes visible to the user.
3. onResume(): The activity is in the foreground and the user can interact with it.
4. onPause(): Activity is partially obscured by another activity. Another activity that’s in the foreground is semi-transparent.
5. onStop(): The activity is completely hidden and not visible to the user.
6. onRestart(): From the Stopped state, the activity either comes back to interact with the user or the activity is finished running and goes away. If the activity comes back, the system invokes onRestart()
7. onDestroy(): Activity is destroyed and removed from the memory.

Activity Lifecycle Example

First we will create a new Android Project and name the activity as HomeActivity. In our case we have named our App project as Activity Lifecycle Example.
We will initialize a static String variable with the name of the underlying class using getSimpleName() method. In our case HOME_ACTIVITY_TAG is the name of the String variable which store class name HomeActivity.

```
private static final String HOME_ACTIVITY_TAG = HomeActivity.class.getSimpleName();
```

Now we will create a new method which will print message in Logcat.

```
private void showLog(String text){
  Log.d(HOME_ACTIVITY_TAG, text);
}
```

Now we will override all activity lifecycle method in Android and use showLog() method which we creating for printing message in Logcat.

```
@Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
      showLog("Activity Created");
   }
@Override
protected void onRestart(){
   super.onRestart();//call to restart after onStop
   showLog("Activity restarted");
}
@Override
protected void onStart() {
   super.onStart();//soon be visible
   showLog("Activity started");
}
@Override
protected void onResume() {
   super.onResume();//visible
   showLog("Activity resumed");
}
@Override
protected void onPause() {
   super.onPause();//invisible
   showLog("Activity paused");
}
@Override
protected void onStop() {
   super.onStop();
   showLog("Activity stopped");
}
@Override
protected void onDestroy() {
   super.onDestroy();
   showLog("Activity is being destroyed");
}
```

Complete JAVA code of HomeActivity.java:

```
package com.abhiandroid.activitylifecycleexample;

import android.os.Bundle;

import android.support.design.widget.FloatingActionButton;

import android.support.design.widget.Snackbar;

import android.support.v7.app.AppCompatActivity;

import android.support.v7.widget.Toolbar;

import android.util.Log;

import android.view.View;

import android.view.Menu;

import android.view.MenuItem;

  public class HomeActivity extends AppCompatActivity {

    private static final String HOME_ACTIVITY_TAG = HomeActivity.class.getSimpleName();

    private void showLog(String text){

      Log.d(HOME_ACTIVITY_TAG, text);

    }

    @Override

    public void onCreate(Bundle savedInstanceState) {

      super.onCreate(savedInstanceState);

      showLog("Activity Created");

    }

    @Override

    protected void onRestart(){

      super.onRestart();//call to restart after onStop

      showLog("Activity restarted");

    }

    @Override

    protected void onStart() {

      super.onStart();//soon be visible

      showLog("Activity started");

    }
```

```
        @Override
        protected void onResume() {
            super.onResume();//visible
            showLog("Activity resumed");
        }
        @Override
        protected void onPause() {
            super.onPause();//invisible
            showLog("Activity paused");
        }
        @Override
        protected void onStop() {
            super.onStop();
            showLog("Activity stopped");
        }
        @Override
        protected void onDestroy() {
            super.onDestroy();
            showLog("Activity is being destroyed");
        }
    }
```

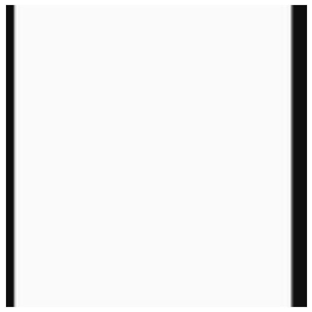
When creating an Activity we need to register this in AndroidManifest.xml file. Now question is why need to register? **Its actually because manifest file has the information which Android OS read very first.** When registering an activity other information can also be defined within manifest like Launcher Activity (An activity that should start when user click on app icon).

Here is declaration example in AndroidManifest.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.abhiandroid.homeactivity">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".HomeActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Output Of Activity Lifecycle:

When you will run the above program you will notice a blank white screen will open up in Emulator. *You might be wondering where is default Hello world screen. Actually we have removed it by overriding onCreate() method.* Below is the blank white screen that will pop up.

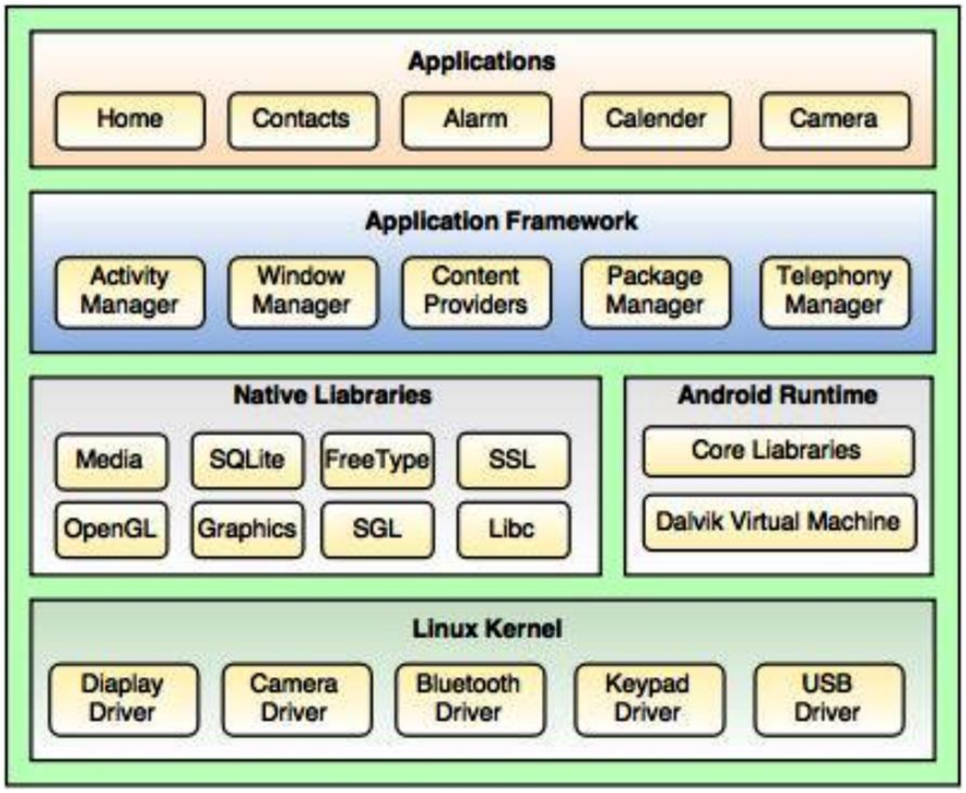


Q2.Explain Android Architecture in details?

Android architecture is a stack of software components. It is in the form of a software application, operating system, run-time environment, middleware, native libraries and services.

It is categorized into five parts as below:
1. Linux Kernel
2. Native Libraries
3. Android Runtime
4. Application Framework
5. Applications

The following diagram shows the architecture of Android



1. Linux Kernel
Linux is the heart of Android architecture.It provides a level of abstraction between the hardware devices and the upper layers of the Android software stack.The Android operating system is based on the Linux kernel.The Linux kernel is responsible for various device drivers such as Camera driver, Display driver, Bluetooth driver, Keypad driver, Memory management, Process management, Power management, etc.

2. Native Libraries
The native libraries such as Media, WebKit, SQLite, OpenGL, FreeType, C Runtime library (libc) etc. are situated on the top of a Linux kernel.Media library is responsible for playing and recording audio and video formats, FreeType is for font support, WebKit is for browser support, SQLite is for database, SSL is for Internet security etc.

3. Android Runtime
Android Runtime is the third section of the architecture and situated on the second layer from the bottom.Android Runtime includes core libraries and Dalvik Virtual Machine (DVM) which is responsible to run android application.Dalvik Virtual Machine (**DVM**) is like Java Virtual Machine (**JVM**) in Java, but DVM is optimized for mobile Devices.DVM makes use of the Linux core features like memory management and multi-threading, which are essential in the Java language.DVM provides fast performance and consumes less memory.

4. Application Framework

Application framework is situated on the top of the Native libraries and Android runtime.Android framework provides a lot of classes and interfaces for Android application development and higher level services to the applications in the form of Java classes.It includes Android API's such as Activity manager, Window manager, Content Provider, Telephony Manager, etc.Activity manger is responsible for controlling all the aspects of the application lifecycle and activity stack, Content provider is responsible for allowing the applications to publish and share the data with the other applications, View system is responsible for creating application user interfaces, etc.

5. Applications

Applications are situated on the top of the Application framework.The applications such as Home, Contact, Alarm, Calender, Camera, Browsers, etc. use the Android framework which uses Android runtime and libraries. Android runtime and Native libraries use Linux kernel.The user can write his/her application to be installed on this layer only.

Q3 What is use of Intent with example?

Android Intent is the *message* that is passed between components such as activities, content providers, broadcast receivers, services etc.It is generally used with startActivity() method to invoke activity, broadcast receivers etc.The **dictionary meaning** of intent is *intention or purpose*. So, it can be described as the intention to do action.The LabeledIntent is the subclass of android.content.Intent class.

Android intents are mainly used to:

- Start the service
- Launch an activity
- Display a web page
- Display a list of contacts
- Broadcast a message
- Dial a phone call etc.

There are two types of intents in android: implicit and explicit.

1) Implicit Intent

Implicit Intent doesn't specifiy the component. In such case, intent provides information of available components provided by the system that is to be invoked.

For example, you may write the following code to view the webpage.

```
Intent intent=new Intent(Intent.ACTION_VIEW);

intent.setData(Uri.parse("http://www.javatpoint.com"));

startActivity(intent);
```

2) Explicit Intent

Explicit Intent specifies the component. In such case, intent provides the external class to be invoked.

```
Intent i = new Intent(getApplicationContext(), ActivityTwo.class);

startActivity(i);
```

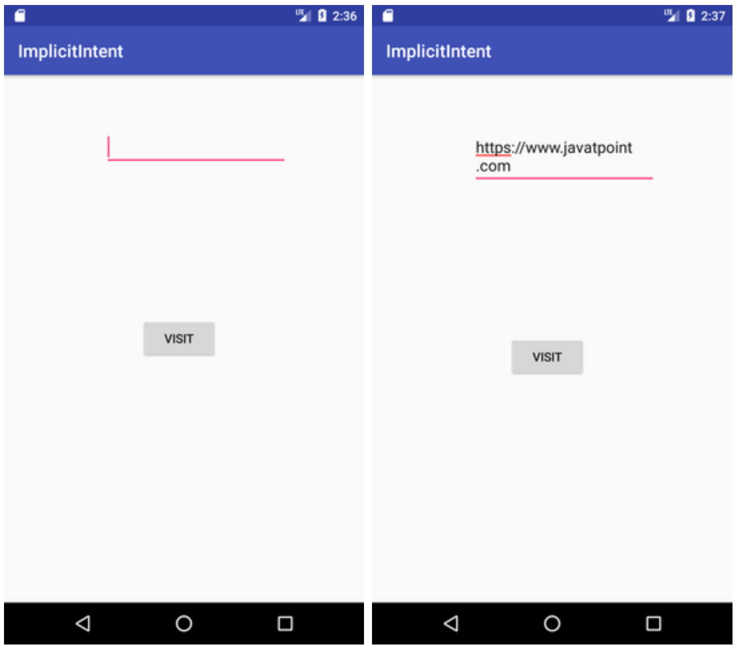
Android Implicit Intent Example

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/androi
d"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.implicitintent.MainActivity">

    <EditText
        android:id="@+id/editText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="60dp"
        android:ems="10"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.575"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="8dp"
        android:layout_marginLeft="156dp"
        android:layout_marginTop="172dp"
        android:text="Visit"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editText" />
</android.support.constraint.ConstraintLayout>
```

MainActivity.java

```
package example.javatpoint.com.implicitintent;

import android.content.Intent;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

    Button button;
    EditText editText;

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

        button = findViewById(R.id.button);
        editText =  findViewById(R.id.editText);

        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String url=editText.getText().toString();
                Intent intent=new Intent(Intent.ACTION_VIEW, Uri.parse(url));
                startActivity(intent);
            }
        });
    }
}
```



Android Explicit Intent Example

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/re
s/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="example.javatpoint.com.explicitintent.FirstActivity">

<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="First Activity"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.454"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.06" />

<Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="392dp"
        android:onClick="callSecondActivity"
        android:text="Call second activity"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

MainActivityOne.java

```
package example.javatpoint.com.explicitintent;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class FirstActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_first);
}
public void callSecondActivity(View view){
        Intent i = new Intent(getApplicationContext(), SecondActivity.class);
        i.putExtra("Value1", "Android By Javatpoint");
        i.putExtra("Value2", "Simple Tutorial");
        // Set the request code to any code you like, you can identify the
        // callback via this code
        startActivity(i);
}

}
```

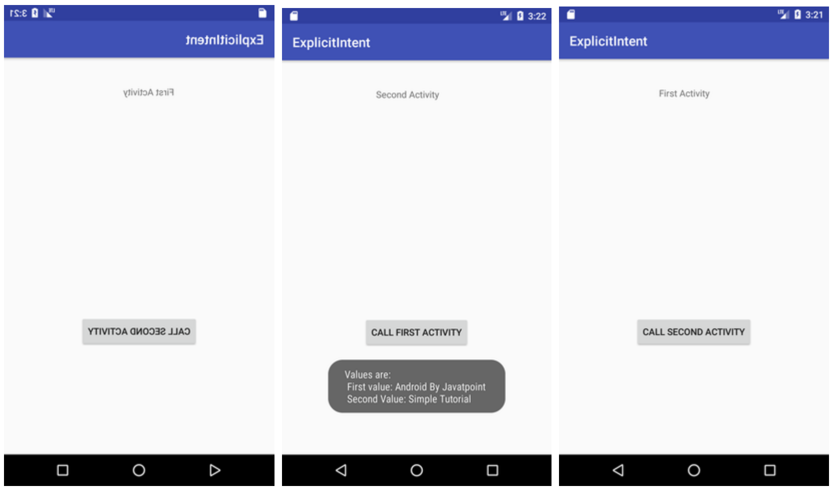
activitytwo_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/re
s/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="example.javatpoint.com.explicitintent.SecondActivity">

<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="Second Activity"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.454"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.06" />

<Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="392dp"
        android:onClick="callFirstActivity"
        android:text="Call first activity"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

MainActivityTwo.java

```
package example.javatpoint.com.explicitintent;

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

public class SecondActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        Bundle extras = getIntent().getExtras();
        String value1 = extras.getString("Value1");
        String value2 = extras.getString("Value2");
        Toast.makeText(getApplicationContext(),"Values are:\n First value: "+value1+
                "\n Second Value: "+value2, Toast.LENGTH_LONG).show();
}
public void callFirstActivity(View view){
        Intent i = new Intent(getApplicationContext(), FirstActivity.class);
```

```
                startActivity(i);
        }
        }
```



Values are:
First value: Android By Javatpoint
Second Value: Simple Tutorial

Q4 Write a program for notification?

MainActivity.java

```
package com.example.notificationdemo;


import android.app.Activity;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.support.v4.app.NotificationCompat;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;


public class MainActivity extends Activity {
   Button b1;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
```

```
    b1 = (Button)findViewById(R.id.button);
    b1.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        addNotification();
      }
    });
  }

  private void addNotification() {
    NotificationCompat.Builder builder =
      new NotificationCompat.Builder(this)
      .setSmallIcon(R.drawable.abc)
      .setContentTitle("Notifications Example")
      .setContentText("This is a test notification");

    Intent notificationIntent = new Intent(this, MainActivity.class);
    PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notificationIntent,
      PendingIntent.FLAG_UPDATE_CURRENT);
    builder.setContentIntent(contentIntent);

    // Add as notification
    NotificationManager manager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
    manager.notify(0, builder.build());
  }
}
```

notification.xml

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

  <TextView
```

```
        android:layout_width="fill_parent"
        android:layout_height="400dp"
        android:text="Hi, Your Detailed notification view goes here...." />
</LinearLayout>
```

NotificationView.java.

```
package com.example.notificationdemo;

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

public class NotificationView extends Activity{
    @Override
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.notification);
    }
}
```

activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="MainActivity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Notification Example"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
```

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

   <TextView
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tutorials point "
      android:textColor="#ff87ff09"
      android:textSize="30dp"
      android:layout_below="@+id/textView1"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="48dp" />

   <ImageButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/imageButton"
      android:src="@drawable/abc"
      android:layout_below="@+id/textView2"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="42dp" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Notification"
      android:id="@+id/button"
      android:layout_marginTop="62dp"
      android:layout_below="@+id/imageButton"
      android:layout_centerHorizontal="true" />

</RelativeLayout>
```

strings.xml

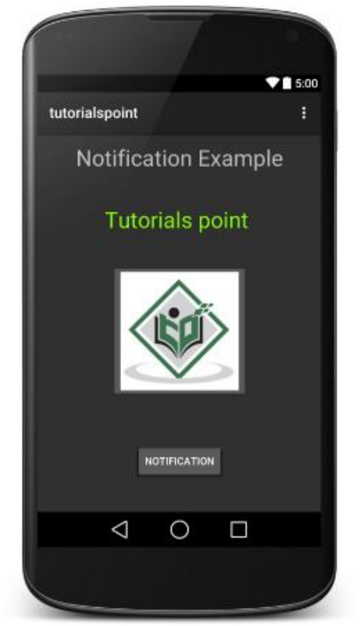
```
<?xml version="1.0" encoding="utf-8"?>
```

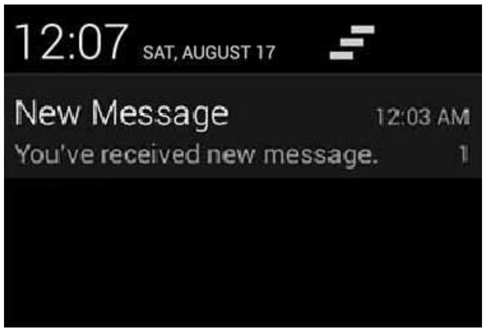
```
<resources>
  <string name="action_settings">Settings</string>
  <string name="app_name">tutorialspoint </string>
</resources>
```

AndroidManifest.xml

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

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.notificationdemo.MainActivity"
      android:label="@string/app_name" >

      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>

    </activity>

    <activity android:name=".NotificationView"
      android:label="Details of notification"
      android:parentActivityName=".MainActivity">
      <meta-data
      android:name="android.support.PARENT_ACTIVITY"
      android:value=".MainActivity"/>
    </activity>
```

```
   </application>
</manifest>
```





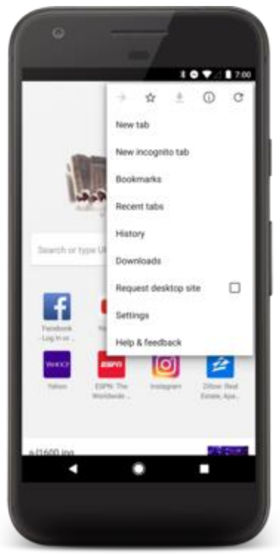
Q5 How to use menu in your application ?

Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience, you should use the `Menu` APIs to present user actions and other options in your activities.

There are 3 types of menus in Android:

1. Option Menu
2. Context Menu
3. Pop-up Menu

Option Menu



The options menu is the primary collection of menu items for an activity. It's where you should place actions that have a overall impact on the app, such as Search, Compose Email and Settings.

You can declare items for the options menu from either your Activity subclass or a Fragment subclass. If both your activity and fragment(s) declare items for the options menu, they are combined in the UI. The activity's items appear first, followed by those of each fragment in the order in which each fragment is added to the activity. If necessary, you can re-order the menu items with the android:orderInCategory attribute in each <item> you need to move.

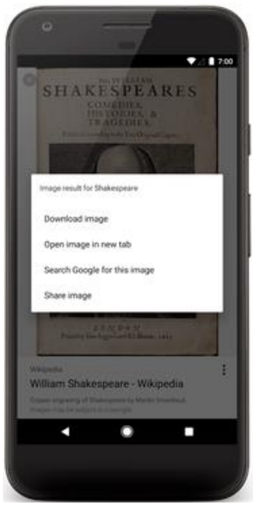
To specify the options menu for an activity, override onCreateOptionsMenu() (fragments provide their own onCreateOptionsMenu() callback). In this method, you can inflate your menu resource (defined in XML) into the Menu provided in the callback. For example:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

Handling click events

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Context Menu


7:00
Image result for Shakespeare
Download image
Open image in new tab
Search Google for this image
Share image
Wikipedia
William Shakespeare - Wikipedia

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.

To provide a floating context menu:

Register the View to which the context menu should be associated by calling registerForContextMenu() and pass it the View.

If your activity uses a ListView or GridView and you want each item to provide the same context menu, register all items for a context menu by passing the ListView or GridView to registerForContextMenu().

Implement the onCreateContextMenu() method in your Activity or Fragment.

When the registered view receives a long-click event, the system calls your onCreateContextMenu() method. This is where you define the menu items, usually by inflating a menu resource. For example:

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

MenuInflater allows you to inflate the context menu from a menu resource. The callback method parameters include the View that the user selected and a ContextMenu.ContextMenuInfo object that provides additional information about the item selected. If your activity has several views that each provide a different context menu, you might use these parameters to determine which context menu to inflate.

Implement onContextItemSelected().

When the user selects a menu item, the system calls this method so you can perform the appropriate action. For example:

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
    switch (item.getItemId()) {
        case R.id.edit:
            editNote(info.id);
            return true;
        case R.id.delete:
            deleteNote(info.id);
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

Enabling the contextual action mode for individual views

Implement the ActionMode.Callback interface

```
private ActionMode.Callback actionModeCallback = new ActionMode.Callback() {

    // Called when the action mode is created; startActionMode() was called
    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        // Inflate a menu resource providing context menu items
        MenuInflater inflater = mode.getMenuInflater();
        inflater.inflate(R.menu.context_menu, menu);
        return true;
    }

    // Called each time the action mode is shown. Always called after onCreateActionMode, but
    // may be called multiple times if the mode is invalidated.
    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        return false; // Return false if nothing is done
    }

    // Called when the user selects a contextual menu item
    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_share:
                shareCurrentItem();
                mode.finish(); // Action picked, so close the CAB
                return true;
            default:
                return false;
        }
    }

    // Called when the user exits the action mode
    @Override
    public void onDestroyActionMode(ActionMode mode) {
        actionMode = null;
    }
};
```

Call startActionMode() to enable the contextual action mode when appropriate, such as in response to a long-click on a View

```
someView.setOnLongClickListener(new View.OnLongClickListener() {
    // Called when the user long-clicks on someView
    public boolean onLongClick(View view) {
        if (actionMode != null) {
            return false;
        }

        // Start the CAB using the ActionMode.Callback defined above
        actionMode = getActivity().startActionMode(actionModeCallback);
        view.setSelected(true);
        return true;
    }
});
```

Enabling batch contextual actions in a ListView or GridView

If you have a collection of items in a ListView or GridView (or another extension of AbsListView) and want to allow users to perform batch actions, you should:

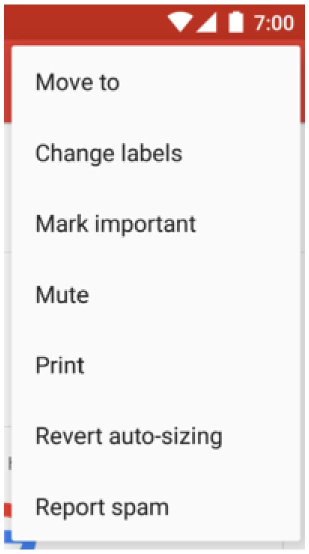
Implement the AbsListView.MultiChoiceModeListener interface and set it for the view group with setMultiChoiceModeListener(). In the listener's callback methods, you can specify the actions for the contextual action bar, respond to click events on action items, and handle other callbacks inherited from the ActionMode.Callback interface.

Call setChoiceMode() with the CHOICE_MODE_MULTIPLE_MODAL argument.

For example:

```
ListView listView = getListView();
listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
listView.setMultiChoiceModeListener(new MultiChoiceModeListener() {

    @Override
    public void onItemCheckedStateChanged(ActionMode mode, int position,
                                          long id, boolean checked) {
        // Here you can do something when items are selected/de-selected,
        // such as update the title in the CAB
    }

    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
        // Respond to clicks on the actions in the CAB
        switch (item.getItemId()) {
            case R.id.menu_delete:
                deleteSelectedItems();
                mode.finish(); // Action picked, so close the CAB
                return true;
            default:
                return false;
        }
    }

    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        // Inflate the menu for the CAB
        MenuInflater inflater = mode.getMenuInflater();
        inflater.inflate(R.menu.context, menu);
        return true;
    }
    @Override
    public void onDestroyActionMode(ActionMode mode) {
        // Here you can make any necessary updates to the activity when
        // the CAB is removed. By default, selected items are deselected/unchecked.
    }

    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        // Here you can perform updates to the CAB due to
        // an <code><a
href="/reference/android/view/ActionMode.html#invalidate()">invalidate()</a></code> request
        return false;
    }
});
```

PopUp Menu



A popup menu displays a list of items in a vertical list that is anchored(sticked) to the view that invoked the menu. It's good for providing an overflow of actions that relate to specific content or to provide options for a second part of a command.

If you define your menu in XML, here's how you can show the popup menu:

Instantiate a PopupMenu with its constructor, which takes the current application Context and the View to which the menu should be anchored.

Use MenuInflater to inflate your menu resource into the Menu object returned by PopupMenu.getMenu().

Call PopupMenu.show().

For example, here's a button with the android:onClick attribute that shows a popup menu:

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_overflow_holo_dark"
    android:contentDescription="@string/descr_overflow_button"
    android:onClick="showPopup" />
```

The activity can then show the popup menu like this:

KotlinJava

```
public void showPopup(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    MenuInflater inflater = popup.getMenuInflater();
    inflater.inflate(R.menu.actions, popup.getMenu());
    popup.show();}
```

Handling click events

To perform an action when the user selects a menu item, you must implement the PopupMenu.OnMenuItemClickListener interface and register it with your PopupMenu by calling setOnMenuItemclickListener(). When the user selects an item, the system calls the onMenuItemClick() callback in your interface.

For example:

```
public void showMenu(View v) {
    PopupMenu popup = new PopupMenu(this, v);

    // This activity implements OnMenuItemClickListener
    popup.setOnMenuItemClickListener(this);
    popup.inflate(R.menu.actions);
    popup.show();}

@Override
public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.archive:
            archive(item);
            return true;
        case R.id.delete:
            delete(item);
            return true;
        default:
            return false;
    }
}
```

Q6 Write a Program addition of two number with using editText and button ?

STEP-1: First of all go to the xml file

STEP-2: Now go to the text and write the code for adding 3 textview,2 textedit and Button and Assign ID to each component. Assign margin top, left, right for the location.

STEP-3: Now, open up the activity java file.

STEP-4: Declare few variables and the values entered in the Text Views can be read by using an id which we have set in the XML code above.

STEP-5: Add the click listener to the Add button.

STEP-6: When the Add button has been clicked, add the values and store it into the sum variable.

STEP-7: To show the output in the result text view, set the sum in the textview.

Activity_Main.xml

```
<!--The activity_main.xml is a layout file

an XML-based layout is a file that defines

the different widgets to be used in the UI
```

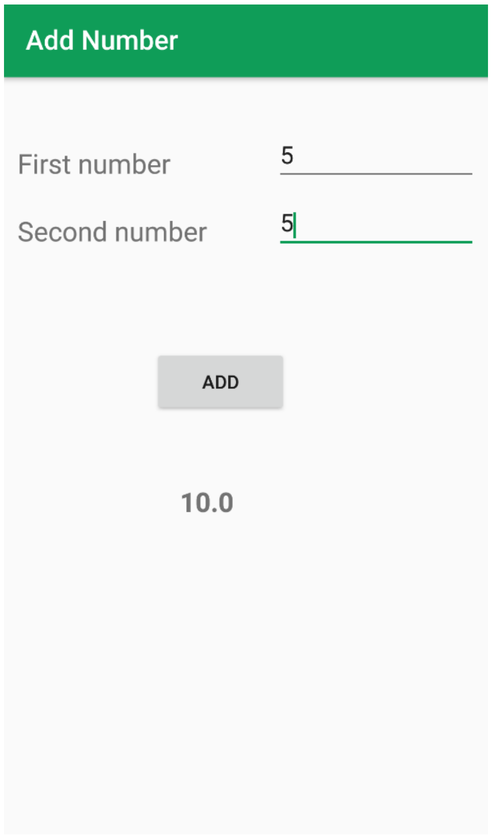
```
and the relations between those widgets
and their containers. -->
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity"
        tools:layout_editor_absoluteY="81dp">

        <!-- Text view for result view-->
        <TextView
                android:id="@+id/textView_answer"
                android:layout_width="100dp"
                android:layout_height="25dp"
                android:layout_marginLeft="130dp"
                android:layout_marginTop="300dp"
                android:text="0"
                android:textSize="20dp"
                android:textStyle="bold" />

        <!--take the input first number-->
        <EditText
                android:id="@+id/editText_first_no"
                android:layout_width="150dp"
                android:layout_height="40dp"
                android:layout_marginLeft="200dp"
                android:layout_marginTop="40dp"
                android:inputType="number" />
        <!-- for message input first number-->
        <TextView
                android:id="@+id/textView_first_no"
                android:layout_width="150dp"
                android:layout_height="25dp"
                android:layout_marginLeft="10dp"
                android:layout_marginTop="50dp"
                android:text="First number"
                android:textSize="20dp" />

        <!--view message -->
        <TextView
                android:id="@+id/textView_second_no"
                android:layout_width="150dp"
                android:layout_height="25dp"
                android:layout_marginLeft="10dp"
                android:layout_marginTop="100dp"
                android:text="Second number"
                android:textSize="20dp" />

        <!-- take input for second number -->

        <EditText
                android:id="@+id/editText_second_no"
                android:layout_width="150dp"
                android:layout_height="40dp"
                android:layout_marginLeft="200dp"
                android:layout_marginTop="90dp"
                android:inputType="number"
                tools:ignore="MissingConstraints" />

        <!-- button for run add logic and view result -->

        <Button
                android:id="@+id/add_button"
                android:layout_width="100dp"
                android:layout_height="50dp"
                android:layout_marginLeft="110dp"
                android:layout_marginTop="200dp"
                android:text="ADD" />
```

```
</RelativeLayout>
```

MainActivity.java

```
package org.geeksforgeeks.addtwonumbers;

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.TextView;

public class MainActivity extends AppCompatActivity {

        // define the global variable

        // variable number1, number2 for input input number
        // Add_button, result textView

        EditText number1;
        EditText number2;
        Button Add_button;
        TextView result;
        int ans=0;

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

                // by ID we can use each component which id is assign in xml file
                number1=(EditText) findViewById(R.id.editText_first_no);
                number2=(EditText) findViewById(R.id.editText_second_no);
                Add_button=(Button) findViewById(R.id.add_button);
                result = (TextView) findViewById(R.id.textView_answer);

                // Add_button add clicklistener
                Add_button.setOnClickListener(new View.OnClickListener() {

                        public void onClick(View v) {
                                // num1 or num2 double type
                                // get data which is in edittext, convert it to string
                                // using parse Double convert it to Double type
                                double num1 = Double.parseDouble(number1.getText().toString());
                                double num2 = Double.parseDouble(number2.getText().toString());
                                // add both number and store it to sum
                                double sum = num1 + num2;
                                // set it ot result textview
                                result.setText(Double.toString(sum));
                        }
                });
        }
}
```

After Complete layout xml file it will be shown as given below Output

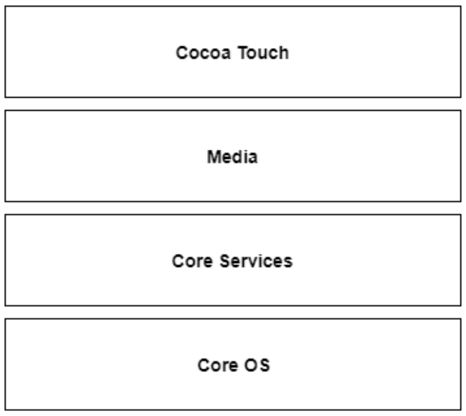

Add Number
First number
5
Second number
5
ADD
10.0

Q7 Explain IOS architecture in detail?

he iOS is the operating system created by Apple Inc. for mobile devices. The iOS is used in many of the mobile devices for apple such as iPhone, iPod, iPad etc. The iOS is used a lot and only lags behind Android in terms of popularity.

The iOS architecture is layered. It contains an intermediate layer between the applications and the hardware so they do not communicate directly. The lower layers in iOS provide the basic services and the higher layers provide the user interface and sophisticated graphics.

The layered architecture of iOS is given as follows −



Apple iOS Architecture

CORE OS Layer:
All the IOS technologies are built under the lowest level layer i.e. Core OS layer. These technologies include:
1. Core Bluetooth Framework
2. External Accessories Framework
3. Accelerate Framework
4. Security Services Framework
5. Local Authorization Framework etc.

It supports 64 bit which enables the application to run faster.

CORE SERVICES Layer:
Some important frameworks are present in the CORE SERVICES Layer which helps the iOS operating system to cure itself ad provide better functionality. It is the 2nd lowest layer in the Architecture as shown above. Below are some important frameworks present in this layer:
1. **Address Book Framework-**
   The Address Book Framework provides access to the contact details of the user.
2. **Cloud Kit Framework-**
   This framework provides a medium for moving data between your app and iCloud.
3. **Core Data Framework-**
   This is the technology that is used for managing the data model of a Model View Controller app.
4. **Core Foundation Framework-**
   This framework provides data management and service features for iOS applications.
5. **Core Location Framework-**
   This framework helps to provide the location and heading information to the application.

6. **Core Motion Framework-**
   All the motion-based data on the device is accessed with the help of the Core Motion Framework.
7. **Foundation Framework-**
   Objective C covering too many of the features found in the Core Foundation framework.
8. **HealthKit Framework-**
   This framework handles the health-related information of the user.
9. **HomeKit Framework-**
   This framework is used for talking with and controlling connected devices with the user's home.
10. **Social Framework-**
    It is simply an interface that will access users' social media accounts.
11. **StoreKit Framework-**
    This framework supports for buying of contents and services from inside iOS apps.

MEDIA Layer:
With the help of the media layer, we will enable all graphics video, and audio technology of the system. This is the second layer in the architecture. The different frameworks of MEDIA layers are:

1. **ULKit Graphics-**
   This framework provides support for designing images and animating the view content.
2. **Core Graphics Framework-**
   This framework support 2D vector and image-based rendering ad it is a native drawing engine for iOS.
3. **Core Animation-**
   This framework helps in optimizing the animation experience of the apps in iOS.
4. **Media Player Framework-**
   This framework provides support for playing the playlist and enables the user to use their iTunes library.
5. **AV Kit-**
   This framework provides various easy-to-use interfaces for video presentation, recording, and playback of audio and video.
6. **Open AL-**
   This framework is an Industry Standard Technology for providing Audio.
7. **Core Images-**
   This framework provides advanced support for motionless images.
8. **GL Kit-**
   This framework manages advanced 2D and 3D rendering by hardware-accelerated interfaces.

COCOA TOUCH:
COCOA Touch is also known as the application layer which acts as an interface for the user to work with the iOS Operating system. It supports touch and motion events and many more features. The COCOA TOUCH layer provides the following frameworks :

1. **EvenKit Framework-**
   This framework shows a standard system interface using view controllers for viewing and changing events.
2. **GameKit Framework-**
   This framework provides support for users to share their game-related data online using a Game Center.
3. **MapKit Framework-**
   This framework gives a scrollable map that one can include in your user interface of the app.
4. **PushKit Framework-**
   This framework provides registration support.

Features of iOS operating System:
Let us discuss some features of the iOS operating system-

1. Highly Securer than other operating systems.

2. iOS provides multitasking features like while working in one application we can switch to another application easily.
3. iOS's user interface includes multiple gestures like swipe, tap, pinch, Reverse pinch.
4. iBooks, iStore, iTunes, Game Center, and Email are user-friendly.
5. It provides Safari as a default Web Browser.
6. It has a powerful API and a Camera.
7. It has deep hardware and software integration

Applications of IOS Operating System:

Here are some applications of the iOS operating system-

1. iOS Operating System is the Commercial Operating system of Apple Inc. and is popular for its security.
2. iOS operating system comes with pre-installed apps which were developed by Apple like Mail, Map, TV, Music, Wallet, Health, and Many More.
3. Swift Programming language is used for Developing Apps that would run on IOS Operating System.
4. In iOS Operating System we can perform Multitask like Chatting along with Surfing on the Internet.

Advantages of IOS Operating System:

The iOS operating system has some advantages over other operating systems available in the market especially the Android operating system. Here are some of them-

1. More secure than other operating systems.
2. Excellent UI and fluid responsive
3. Suits best for Business and Professionals
4. Generate Less Heat as compared to Android.

Disadvantages of IOS Operating System:

Let us have a look at some disadvantages of the iOS operating system-

1. More Costly.
2. Less User Friendly as Compared to Android Operating System.
3. Not Flexible as it supports only IOS devices.
4. Battery Performance is poor.

Q8 What is Fragment and Fragment Life Cycle ?

Fragment

A Fragment represents a reusable portion of your app's UI. A fragment defines and manages its own layout, has its own lifecycle, and can handle its own input events. Fragments cannot live on their own--they must be *hosted* by an activity or another fragment. The fragment's view hierarchy becomes part of, or *attaches to*, the host's view hierarchy.

Modularity

Fragments introduce modularity and reusability into your activity's UI by allowing you to divide the UI into discrete chunks. Activities are an ideal place to put global elements around your app's user interface, such as a navigation drawer. Conversely, fragments are better suited to define and manage the UI of a single screen or portion of a screen.

Consider an app that responds to various screen sizes. On larger screens, the app should display a static navigation drawer and a list in a grid layout. On smaller screens, the app should display a bottom navigation bar and a list in a linear layout. Managing all of these variations in the activity can be unwieldy. Separating the navigation elements from the content can make this process more manageable. The activity is then responsible for displaying the correct navigation UI while the fragment displays the list with the proper layout.

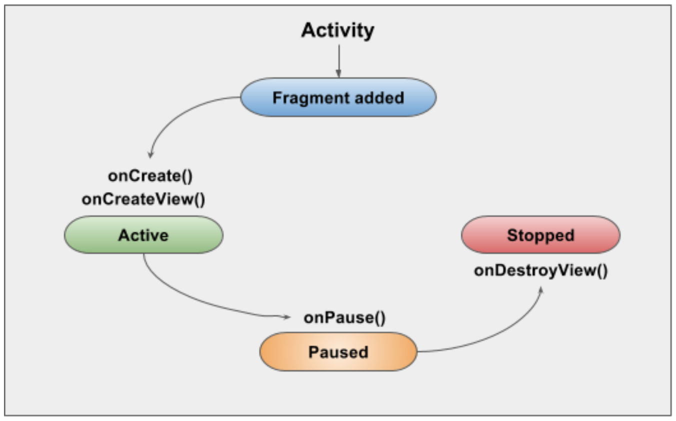
Fragment Lifecycle

Like an Activity, a Fragment has its own lifecycle. Understanding the relationship between Activity and Fragment lifecycles helps you design fragments that can save and restore variables and communicate with activities.
An Activity that hosts a Fragment can send information to that Fragment, and receive information from that Fragment. This chapter describes the mechanisms for passing data and how to manage the Fragment lifecycle within an Activity.

Fragment lifecycle states

The Fragment is added by an Activity (which acts as the host of the Fragment). Once added, the Fragment goes through three states, as shown in the figure below:

- Active (or resumed)
- Paused
- Stopped



Using Fragment lifecycle callbacks

As you would with Activity lifecycle methods, you can override Fragment lifecycle methods to perform important tasks when the Fragment is in certain states. Most apps should implement at least the following methods for every Fragment:

onCreate(): Initialize essential components and variables of the Fragment in this callback. The system calls this method when the Fragment is created. Anything initialized in onCreate() is preserved if the Fragment is paused and resumed.

onCreateView(): Inflate the XML layout for the Fragment in this callback. The system calls this method to draw the Fragment UI for the first time. As a result, the Fragment is visible in the Activity. To draw a UI for your Fragment, you must return the root View of your Fragment layout. Return null if the Fragment does not have a UI.

onPause(): Save any data and states that need to survive beyond the destruction of the Fragment. The system calls this method if any of the following occurs:

The user navigates backward.

The Fragment is replaced or removed, or another operation is modifying the Fragment.

The host Activity is paused.

A paused Fragment is still alive (all state and member information is retained by the system), but it will be destroyed if the Activity is destroyed. If the user presses the Back button and the Fragment is returned from the back stack, the lifecycle resumes with the onCreateView() callback.

The Fragment class has other useful lifecycle callbacks:

onResume(): Called by the Activity to resume a Fragment that is visible to the user and actively running.

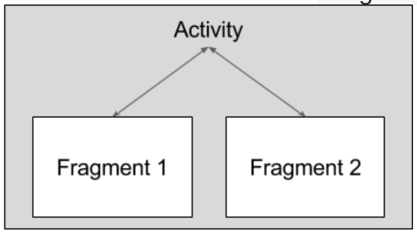
onAttach(): Called when a Fragment is first attached to a host Activity. Use this method to check if the Activity has implemented the required listener callback for the Fragment (if a listener interface was defined in the Fragment). After this method, onCreate()is called.

onActivityCreated(): Called when the Activity onCreate() method has returned. Use it to do final initialization, such as retrieving views or restoring state. It is also useful for a Fragment that uses setRetainInstance() to retain its instance, as this callback tells the Fragment when it is fully associated with the new Activity instance. The onActivityCreated() method is called after onCreateView() and before onViewStateRestored().

onDestroyView(): Called when the View previously created by onCreateView() has been detached from the Fragment. This call can occur if the host Activity has stopped, or the Activity has removed the Fragment. Use it to perform some action, such as logging a message, when the Fragment is no longer visible. The next time the Fragment needs to be displayed, a new View is created. The onDestroyView() method is called after onStop() and before onDestroy().

Communicating between a Fragment and an Activity

The Activity that hosts a Fragment can send information to that Fragment, and receive information from that Fragment, as described in "Sending data from a fragment to its host activity" in this chapter. However, a Fragment can't communicate directly with another Fragment. All Fragment-to-Fragment communication is done through the Activity that hosts them. One Fragment communicates with the Activity, and the Activity communicates to the other Fragment.


Activity
Fragment 1
Fragment 2