VJTech Academy

Inspiring Your Success...

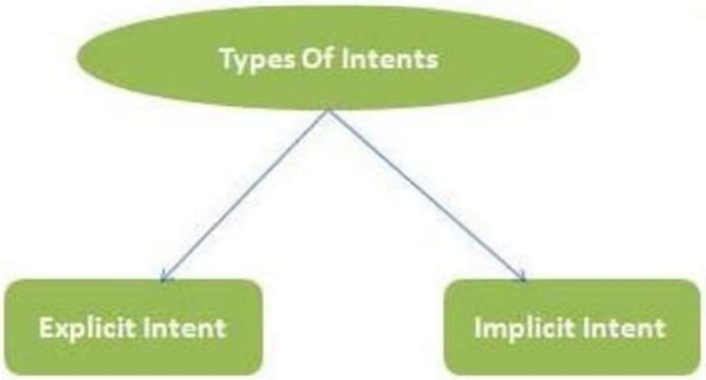
UNIT-V Activity and Multimedia with databases

Intent in Android:

- Android uses Intent for communicating between the components of an application and also from one application to another application.
- Intent are the objects which is used in android for passing the information among Activities in an Application and from one app to another also.
- Intent is used for communicating between the Application components and it also provides the connectivity between two apps
- 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.

Types of Intents:

- There are two types of Intents: Explicit Intent and Implicit Intent



1. **Implicit Intent:**

- In Implicit Intents we do need to specify the name of the component.
- We just specify the Action which has to be performed and further this action is handled by the component of another application.
- The basic example of implicit Intent is to open any web page
- Refer to below example to understand Implicit Intents more clearly. We have to open a website using intent in your application.

```
Intent obj = new Intent(Intent.ACTION_VIEW);
obj.setData(Uri.parse("https://www.vjtechacademy.in"));
startActivity(obj);
```

- In this example we have just specified an action. Now when we will run this code then Android will automatically start your web browser and it will open VJTech Academy home page.
- In this intent, you do not use any class name to pass through Intent().
- **Android Implicit Intent Example:** Let's see the simple example of implicit intent that displays a web page.

STEP-1: Create activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/b1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="300dp"
        android:layout_marginEnd="20dp"
        android:layout_marginStart="20dp"
        android:text="Visit Website" />
</RelativeLayout>
```

STEP-2: Create MainActivity.java file

```
package com.vjtech.implicitintent;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity
{

    Button button;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button = findViewById(R.id.b1);
        button.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View view)
            {
                Intent obj=new Intent(Intent.ACTION_VIEW);

obj.setData(Uri.parse("https://www.vjtechacademy.in"));
                startActivity(obj);
            }
        });
    }
}
```

2. Explicit Intent:

- Explicit Intents are used to connect the application internally.
- In Explicit we use the name of component which will be affected by Intent.
- For Example: If we know class name then we can navigate the app from One Activity to another activity using Intent.
- Explicit Intent work internally within an application to perform navigation and data transfer. The below given code snippet will help you understand the concept of Explicit Intents

```
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
startActivity(intent);
```

- In above example, SecondActivity is the JAVA class name where the activity will now be navigated.
- **Android Explicit Intent Example:** Let's see the simple example of android explicit example that calls one activity from another and vice versa.

STEP-1: Create first activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="100dp"
        android:layout_marginStart="150dp"
        android:layout_marginTop="100dp"
        android:text="First Activity" />
    <Button
        android:id="@+id/b1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="300dp"
        android:layout_marginEnd="20dp"
        android:layout_marginStart="20dp"
        android:onClick="callSecondActivity"
        android:text="Call second activity" />
</RelativeLayout>
```

STEP-2: Create first MainActivity.java file

```
package com.vjtech.explicitintent;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity
{

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void callSecondActivity(View view)
    {
    Intent i = new Intent(getApplicationContext(),MainActivity2.class);
    startActivity(i);
    }
}
```

STEP-3: Create second activity_main2.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="100dp"
        android:layout_marginStart="150dp"
        android:layout_marginTop="100dp"
        android:text="Second Activity" />
    <Button
        android:id="@+id/b1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="300dp"
        android:layout_marginEnd="20dp"
        android:layout_marginStart="20dp"
        android:onClick="callFirstActivity"
        android:text="Call first activity" />
</RelativeLayout>
```

STEP-4: Create second MainActivity2.java file

```
package com.vjtech.explicitintent;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity2 extends AppCompatActivity
{

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);
    }
    public void callFirstActivity(View view)
    {
     Intent i = new Intent(getApplicationContext(),MainActivity.class);
     startActivity(i);
    }
}
```

Intent Filter in Android:

- Intent Filter are the components which decide the behavior of an intent.
- In implicit and explicit intent, we can navigate from one activity to another, that can be achieve by declaring intent filter.
- We can declare an Intent Filter for an Activity in manifest file.
- Intent filters specify the type of intents that an Activity, service or Broadcast receiver can respond to.
- It declares the functionality of its parent component (i.e., activity, services or broadcast receiver). It declares the capability of any activity or services or a broadcast receiver.

Syntax of Intent Filters:

```
<activity android:name=".MainActivity">
   <intent-filter android:icon="@drawable/icon" android:label="@string/label">
       <action android:name="android.intent.action.MAIN" />
       <category android:name="android.intent.category.LAUNCHER" />
       <data android:mimeType="text/plain"/>
   </intent-filter>
</activity>
```

Attributes of Intent Filter:

1. **android:icon** An icon represents the activity, service or broadcast receiver when a user interact with it or when it appears to user in an application. To set an icon you need to give reference of drawable resource as declared android:icon="@drawable/icon".
2. **android:label** label represents the title of an activity on the toolbar. You can have different Labels for different Activities as per your requirement or choice.

Elements in Intent Filter:

There are following three elements in an intent filter: Every intent filter must contain action element in it. Data and category element is optional for it.

1. **Action**
   - It represents an activities action, what an activity is going to do. It is declared with the name attribute as given below

```
<action android:name = "string" />
```

- An Intent Filter element must contain one or more action element. Action is a string that specifies the action to perform.

2. Category

- This attribute of Intent filter dictates the behavior or nature of an Intent. There is a string which contains some additional information about the intent which will be handled by a component.
- The syntax of category is as follows:

```
<category android:name="string" />
```

- The category of intent: CATEGORY_BROWSABLE, CATEGORY_LAUNCHER.
- BROWSABLE – Browsable category, activity allows itself to be opened with web browser to open the reference link provided in data.
- LAUNCHER – Launcher category puts an activity on the top of stack, whenever application will start, the activity containing this category will be opened first.

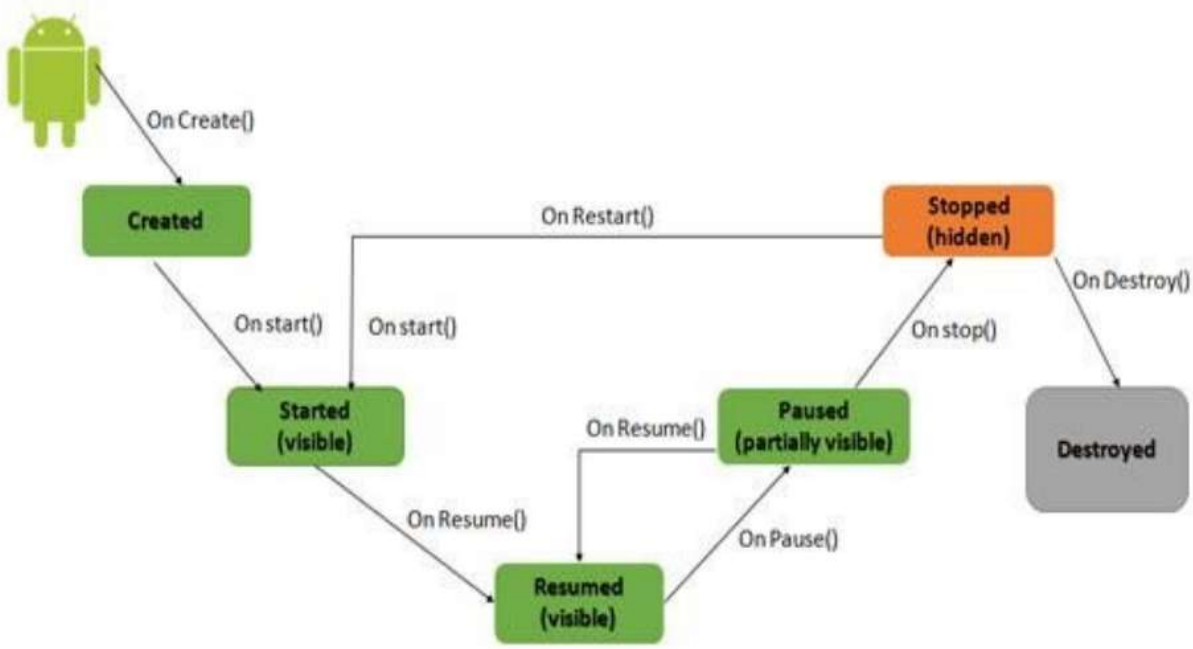
3. Data

- There are two forms in which you can pass the data, using URI(Uniform Resource Identifiers) or MIME type of data.

```
<data android:mimeType="text/plain"/>
```

Activity Lifecycle in Android:

- Activity is one of the building blocks of Android OS.
- In simple words Activity is a screen that user interact with.
- Every Activity in android has lifecycle like created, started, resumed, paused, stopped or destroyed.
- These different states are known as Activity Lifecycle. In other words, we can say Activity is a class pre-written in Java Programming.
- Below Activity Lifecycle Diagram Shows Different States:



- Short description of Activity Lifecycle:
  1. onCreate() – Called when the activity is first created
  2. onStart() – Called just after its creation or by restart method after onStop(). Here Activity start becoming visible to user.
  3. onResume() – Called when Activity is visible to user and user can interact with it.
  4. onPause() – Called when Activity content is not visible because user resume previous activity
  5. onStop() – Called when activity is not visible to user because some other activity takes place of it
  6. onRestart() – Called when user comes on screen or resume the activity which was stopped
  7. onDestroy – Called when Activity is not in background
- Activity is the main component of Android Application, as every screen is an activity so to create any simple app first, we have to start with Activities. Every lifecycle method is quite important to

implement according to requirements, However onCreate(Bundle state) is always needed to implement to show or display some content on screen.

BroadReceiver in Android:

- In android, Broadcast Receiver is a component which will allow android system or other apps to deliver events to the app like sending a low battery message or screen turned off message to the app.
- The apps can also initiate broadcasts to let other apps know that required data available in a device to use it.
- Generally, we use Intents to deliver broadcast events to other apps and Broadcast Receivers use status bar notifications to let user know that broadcast event occurs.
- In android, Broadcast Receiver is implemented as a subclass of BroadcastReceiver and each broadcast is delivered as an Intent object.
- There are following two important steps to make BroadcastReceiver works for the system broadcasted intents:
  1. Creating the Broadcast Receiver
  2. Registering Broadcast Receiver

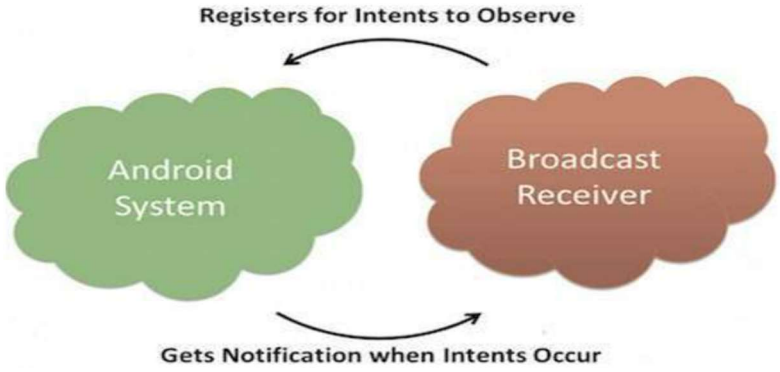
Creating the Broadcast Receiver:

- A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and overriding the onReceive() method where each message is received as a **Intent** object parameter.

```
public class MyReceiver extends BroadcastReceiver
{
  @Override
  public void onReceive(Context context, Intent intent)
  {
        Toast.makeText(context, "Intent Detected.", Toast.LENGTH_LONG).show();
  }
}
```

Registering Broadcast Receiver:

- An application listens for specific broadcast intents by registering a broadcast receiver in AndroidManifest.xml file. Consider we are going to register MyReceiver for system generated event ACTION_BOOT_COMPLETED which is fired by the system once the Android system has completed the boot process.




```
<application
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme" >
  <receiver android:name="MyReceiver">

    <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED">
      </action>
    </intent-filter>

  </receiver>
</application>
```

- Now whenever your Android device gets booted, it will be intercepted by BroadcastReceiver MyReceiver and implemented logic inside onReceive() will be executed.
- There are several system generated events defined as final static fields in the Intent class.
- **android.intent.action.BATTERY_LOW** : It is used to call an event when battery is low on device.
- **android.intent.action.BATTERY_OKAY :** It is used to call an event when battery is OKAY again.
- **android.intent.action.REBOOT :** It call an event when the device rebooted again.
- **android.intent.action.BOOT_COMPLETED** : It raise an event, once boot completed.
- **android.intent.action.POWER_CONNECTED** : It is used to trigger an event when power connected to the device
- **android.intent.action.POWER_DISCONNECTED** : It is used to trigger an event when power got disconnected from the device.

Android Service:

- Android service is a component that is used to perform operations on the background such as playing music, handle network transactions, interacting content providers etc. It doesn't have any UI (user interface).
- The service runs in the background indefinitely even if application is destroyed.
- Service can be bounded by a component to perform interactivity and inter process communication (IPC).

Features of Service:

- Service is an Android Component without an UI.
- It is used to perform long running operations in background. Services run indefinitely unless they are explicitly stopped or destroyed.
- It can be started by any other application component. Components can even infact bind to a service to perform Interprocess - Communication.
- It can still be running even if the application is killed unless it stops itself by calling stopself() or is stopped by an Android component by calling stopService().
- If not stopped it goes on running unless is terminated by Android due to resource shortage.
- The android.app.Service is subclass of ContextWrapper class.

Android platform service:

- The Android platform provides and runs predefined system services and every Android application can use them, given the right permissions.
- These system services are usually exposed via a specific Manager class. Access to them can be gained via the getSystemService() method.
- **Permission:**
- The purpose of a permission is to protect the privacy of an Android user.
- Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet).

- Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request.
- **Add permissions to the manifest:**
- On all versions of Android, to declare that your app needs a permission, put an element in your app manifest, as a child of the top-level element. For example, an app that needs to access the internet would have this line in the manifest:

```
<manifest xmlns:android=http://schemas.android.com/apk/res/android
package="com.vjtech.SampleApp">
    <uses-permission android:name="android.permission.INTERNET"/>
    <application ...>
        ...
    </application>
</manifest>
```

Life Cycle of Android Service:

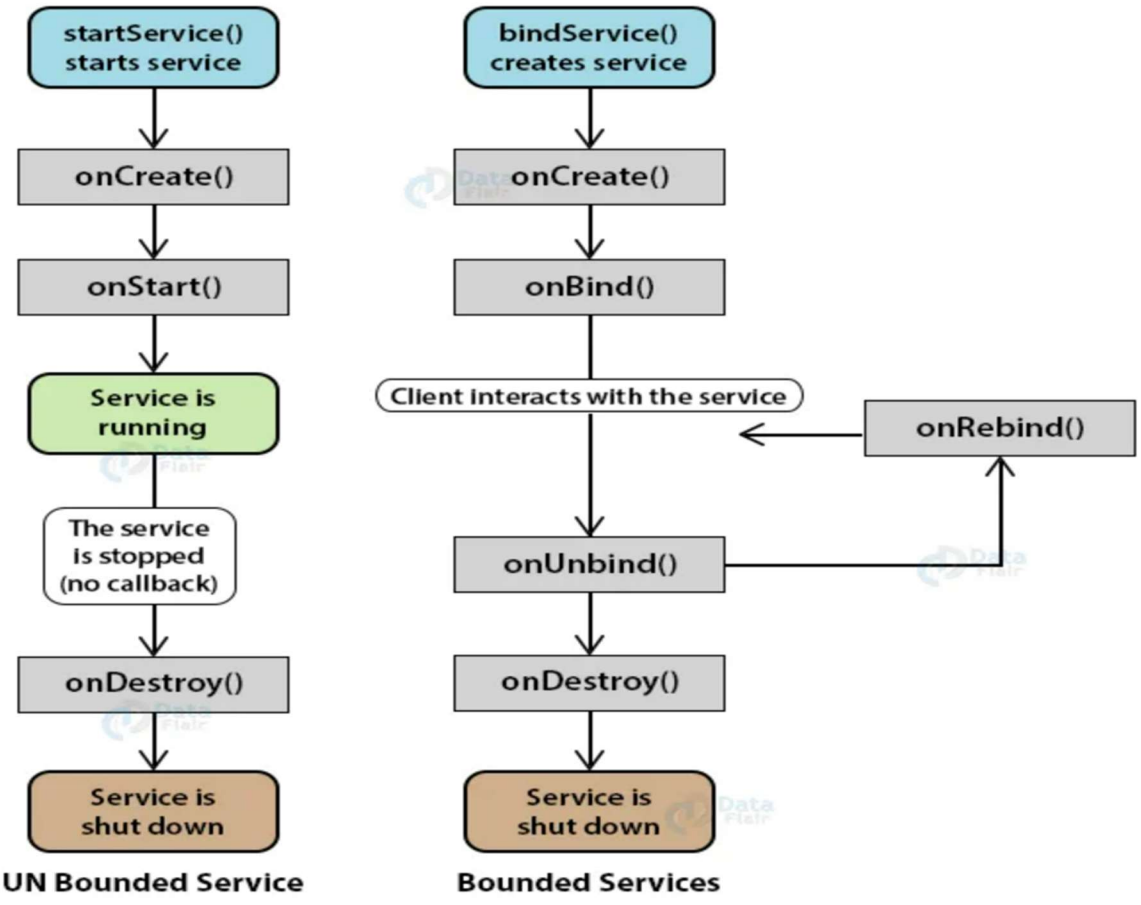
- There can be two forms of a service. The lifecycle of service can follow two different paths: started or bound.
  1. Started
  2. Bound

1) Started Service

- A service is started when component (like activity) calls startService() method, now it runs in the background indefinitely.
- It is stopped by stopService() method. The service can stop itself by calling the stopSelf() method.

2) Bound Service

- A service is bound when another component (e.g. client) calls bindService() method.
- The client can unbind the service by calling the unbindService() method.
- The service cannot be stopped until all clients unbind the service.



onCreate()

- This is the first callback which will be invoked when any component starts the service.
- If the same service is called again while it is still running this method won't be invoked.
- Ideally one time setup and initializing should be done in this callback.

onStart()

- This callback is invoked when service is started by any component by calling startService().
- It basically indicates that the service has started and can now run indefinitely.

onBind()

- This is invoked when any component starts the service by calling onBind.

onUnbind()

- This is invoked when all the clients are disconnected from the service.

onRebind()

- This is invoked when new clients are connected to the service. It is called after

onDestroy()

- This is a final clean up call from the system. This is invoked just before the service is being destroyed.
- Could be very useful to cleanup any resources such as threads, registered listeners, or receivers.

Example of Android Service:

Following is the example of start playing music in the background when we start a service and that music will play continuously until we stop the service in the android application

STEP-1: Create MyService.java file

```
package com.vjtech.androidserviceexample;

import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;
import android.provider.Settings;
import android.widget.Toast;
public class MyService extends Service
{
    private MediaPlayer player;
    public IBinder onBind(Intent intent)
    {
        return  null;
    }
    public void onCreate()
    {
       Toast.makeText(this,"Service was Created",Toast.LENGTH_LONG).show();
    }
    public int onStartCommand(Intent intent, int flags, int startId)
    {
        player=MediaPlayer.create(this,Settings.System.DEFAULT_RINGTONE_URI);
        player.setLooping(true);
        player.start();
        Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
        return START_STICKY;
    }
    public void onDestroy()
    {
        super.onDestroy();
        player.stop();
        Toast.makeText(this, "Service Stopped", Toast.LENGTH_LONG).show();
    }
}
```

STEP-2: Create activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/btnStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="startService"
        android:layout_marginLeft="130dp"
        android:layout_marginTop="150dp"
        android:text="Start Service"/>
    <Button
        android:id="@+id/btnstop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="stopService"
        android:layout_marginLeft="130dp"
        android:layout_marginTop="20dp"
        android:text="Stop Service"/>
</LinearLayout>
```

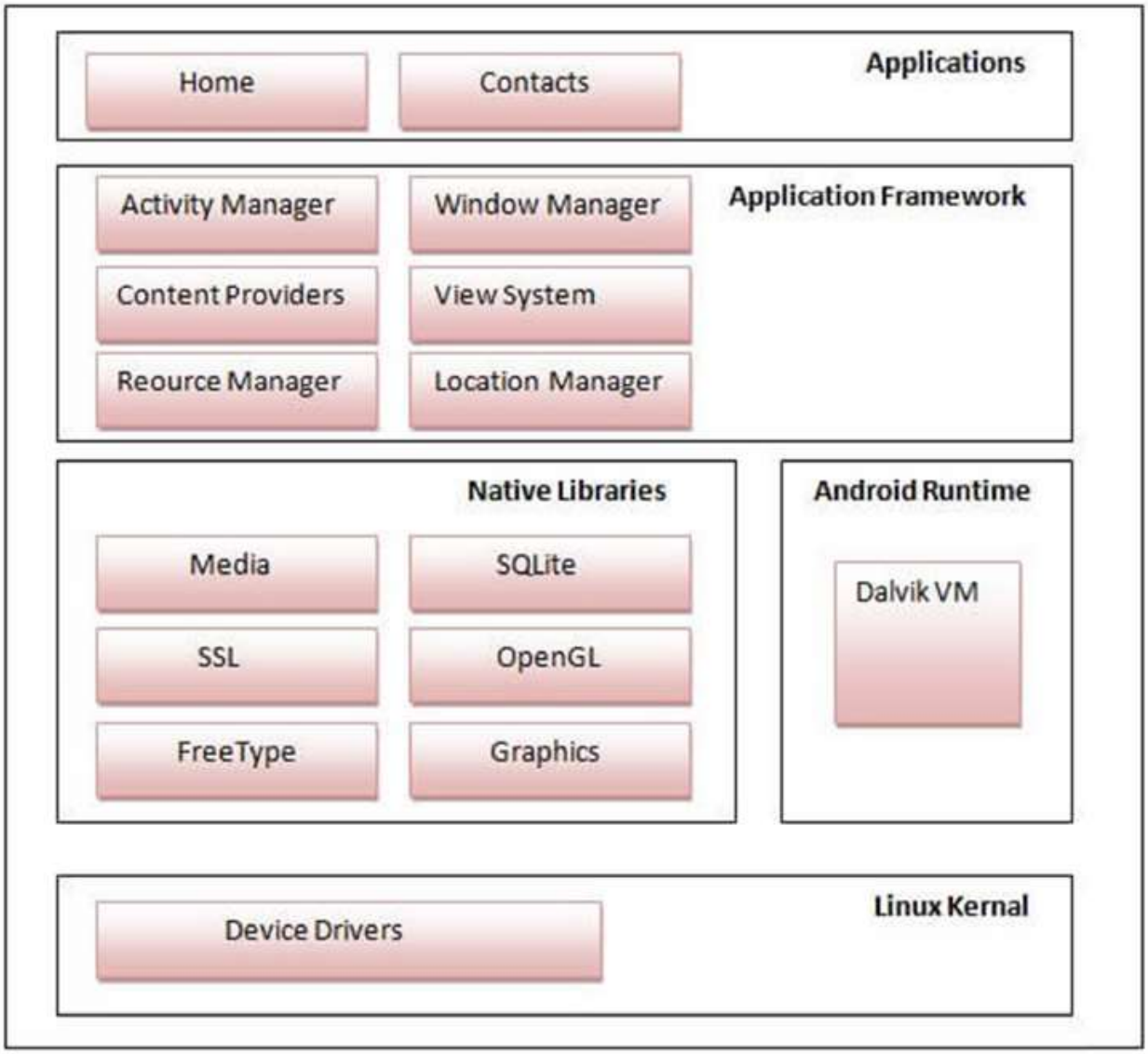
STEP-3: Create MainActivity.java file

```
package com.vjtech.androidserviceexample;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity
{

    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void startService(View view)
    {
        startService(new Intent(this, MyService.class));
    }
    public void stopService(View view)
    {
        stopService(new Intent(this, MyService.class));
    }
}
```

Android System Architecture:

The main components of android architecture are following:

1. Linux kernel
2. native libraries (middleware),
3. Android Runtime
4. Application Framework
5. Applications



1. Linux kernel:

- It is the heart of android architecture.
- It manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc
- Linux kernel is responsible for device drivers, power management, memory management, device management and resource access.

2. Native Libraries:

- The Native Libraries includes various C/C++ core libraries and Java based libraries such as Media, Graphics, OpenGL, SSL etc. to provide a support for android development.
- **Media** library provides support to play and record an audio and video formats.
- **Open GL(graphics library):** This cross-language, cross-platform application program interface (API) is used to produce 2D and 3D computer graphics.
- **Secure Socket Layer (SSL):** These libraries are used to establish a security between a web server and a web browser.
- **Web-Kit** This open-source web browser engine provides all the functionality to display web content and to simplify page loading.
- **SQLite** provides database support.
- **FreeType** provides font support.

3. Android Runtime:

- In android runtime, there are core libraries and DVM (Dalvik Virtual Machine) which is responsible to run android application.
- DVM is like JVM but it is optimized for mobile devices. It consumes less memory and provides fast performance.
- The DVM is a virtual machine to run Android applications. The DVM executes Dalvik bytecode, which is compiled from programs written in the Java language.

4. Application Framework:

- Application Framework provides a lot of classes and interfaces for android application development.
- **Activity Manager:** It manages the activity lifecycle and the activity stack.

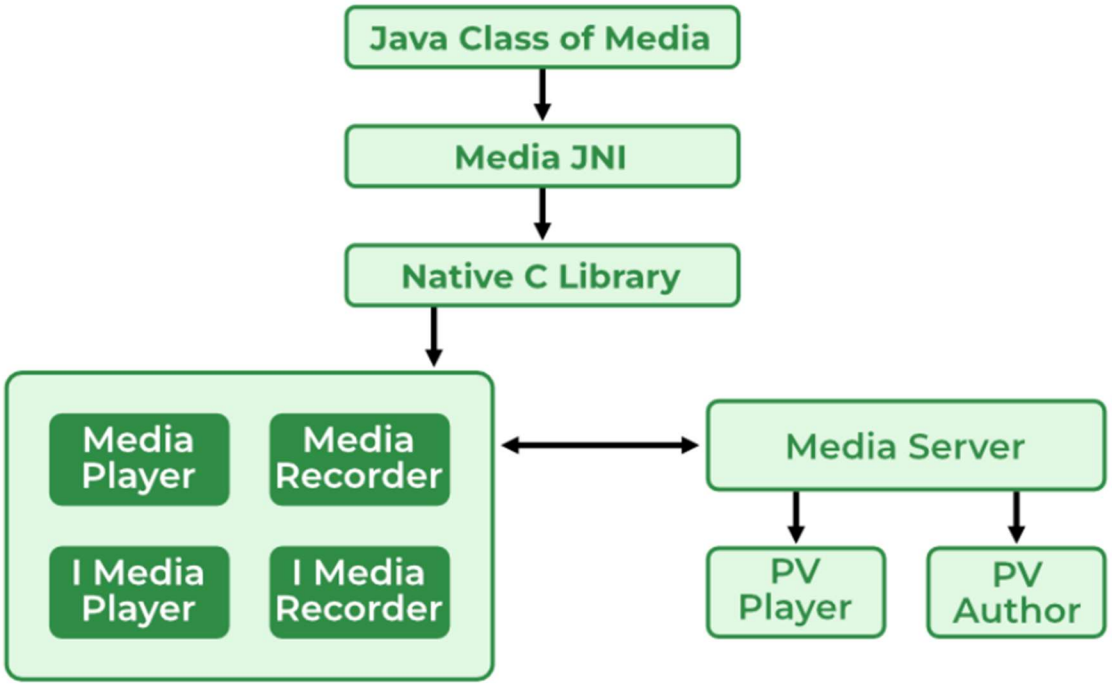
- **Telephony Manager:** It provides access to telephony services as related subscriber information, such as phone numbers.
- **View System:** It builds the user interface by handling the views and layouts.
- **Location manager:** It finds the device's geographic location.
- **Content Providers:** Allows 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.
- **Window manager** is responsible for managing the list of windows, which windows are visible, and how they are displaying on screen.

5. Applications:

- Applications is the top layer of android architecture
- The pre-installed applications like home, contacts, camera, gallery etc and third-party applications downloaded from the play store like chat applications, games etc will be installed on this layer only.
- It runs within the Android run time with the help of the classes and services provided by the application framework.

Android Multimedia Framework:

- Android multimedia framework is designed to provide a reliable interface for java service.
- It is a system that includes multimedia applications, frameworks, an OpenCore engine, hardware devices for audio/video/ input, output devices also several core dynamic libraries such as libmedia, libmediaplayservices, and so on.



- The Android Multimedia framework is a set of APIs for developers which enables them to create a multimedia application on an android platform.
- This framework provides support for audio, video, and images, which provides a range of features such as media playback, recording, editing, streaming, etc.
- Java classes call the Native C library Libmedia through Java JNI (Java Native Interface).
- Libmedia library communicates with Media Server guard process through Android's Binder IPC (inter process communication) mechanism.
- Media Server process creates the corresponding multimedia service according to the Java multimedia applications.
- The whole communication between Libmedia and Media Server forms a Client/Server model.

- In Media Server guard process, it calls OpenCore multimedia engine to realize thespecific multimedia processing functions. And the OpenCore engine refers to the PVPlayer and PVAuthor.

Play Audio and Video:

- We can play and control the audio files in android by the help of MediaPlayer class.
- In android, by using MediaPlayer class we can easily fetch, decode and play both audio and video files with minimal setup.
- The android media framework provides built-in support for playing a variety of common media types, such as audio or video. We have multiple ways to play audio or video but the most important component of media framework is MediaPlayer class.

Android Audio Player:

- MediaPlayer class: The android.media.MediaPlayer class is used to control the audio or video files.
- MediaPlayer class provides a different type of methods to control audio and video files based on requirements.

Method	Description
getCurrentPosition()	It is used to get the current position of the song in milliseconds.
getDuration()	It is used to get the total time duration of the song in milliseconds.
isPlaying()	It returns true / false to indicate whether song playing or not.
pause()	It is used to pause the song playing.
setAudioStreamType()	it is used to specify the audio streaming type.
setDataSource()	It is used to specify the path of audio / video file to play.
setVolume()	It is used to adjust media player volume either up / down.

Method	Description
seekTo(position)	It is used to move song to particular position in milliseconds.
getTrackInfo()	It returns an array of track information.
start()	It is used to start playing the audio/video.
stop()	It is used to stop playing the audio/video.
reset()	It is used to reset the MediaPlayer object.
release()	It is used to releases the resources which are associated with MediaPlayer object.

- **Android Audio Player Example:** Following is the example of implementing an audio player to play a song or audio with multiple playback options using MediaPlayer.

Android Video Player:

- In android, by using VideoView component and MediaController class we can easily implement the video player in android applications to play the videos with multiple playback options, such as play, pause, forward, backward, etc.
- The MediaController class in android will provide playback options for video player, such as play, pause, backward, forward, etc.
- The VideoView class in android will provide the functionality to fetch and play the videos using video player with minimal setup in android applications.
- VideoView class provides a different type of methods to control video files based on requirements.

Method	Description
setMediaController()	It is used to set the media controller to videoview.
setVideoURI()	It is used to set the path of video file.
pause()	It is used to pause the video playing.
stopPlayback()	it is used to stop the video playing.
seekTo(position)	It is used to move video to a particular position in milliseconds.
resume()	It is used to resume the video playing.
start()	It is used to start playing the audio/video.
stopPlayback()	It is used to stop playing the audio/video

- **Android Video Player Example:** Following is the example of implementing a video player to play the video with multiple playback options using VideoView and MediaController objects.

Text To Speech:

- Android allows you convert your text into voice. Not only you can convert it but it also allows you to speak text in variety of different languages.
- Android provides TextToSpeech class for this purpose. In order to use this class, you need to instantiate an object of this class and also specify the initListener.
- In this listener, you have to specify the properties for TextToSpeech object , such as its language etc. Language can be set by calling setLanguage() method.

Android Sensor:

- Android Sensors can be used to monitor the three-dimensional device movement or change in the environment of the device.
- Android provides sensor api to work with different types of sensors. most of the android devices have built-in sensors to measure motion, orientation, and various environmental conditions.
- These sensors will provide raw data with high accuracy and are useful to monitor three-dimensional device movement or positioning or monitor changes in the ambient environment near a device.
- For example, to report changes in the environment a weather application might use a temperature sensor and humidity sensor or a travel application might use the geomagnetic field sensor and accelerometer to report a compass bearing, etc.

Types of Sensors

- Android supports three types of sensors:

1) **Motion Sensors:** These are used to measure acceleration forces and rotational forces along with three axes.

2) **Position Sensors:** These are used to measure the physical position of device.

3) **Environmental Sensors:** These are used to measure the environmental changes such as temperature, humidity etc.

- Android provided a framework called sensor framework to access all the sensors available on device and to get all the raw sensor data.
- The sensor framework provided a wide variety of sensor-related tasks. For example, by using a sensor framework we can perform the following things
  - It lists all the available sensors on the device
  - It determines the capabilities of each sensor, such as its maximum range, manufacturer, power requirements, and resolution.
  - It can acquire raw sensor data and define the minimum rate at which you acquire sensor data.
  - Register and unregister sensor event listeners that monitor sensor changes.

- Android sensor framework provided the following classes and interfaces to access device sensors and acquire raw sensor data.

Class	Description
SensorManager	By using this class, we can create an instance of sensor service and this class provides a various method for accessing and listing sensors, registering and unregistering sensor event listeners and acquiring orientation information.
Sensor	By using this class, we can create an instance of a specific sensor and this class provides various methods that let you determine the sensor's capabilities.
SensorEvent	The system uses this class to create a sensor event object and it provides the raw sensor data, type of sensor that generated the event, accuracy of the data, and the timestamp for the event.
SensorEventListener	We can use this interface to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

- Android Sensor Example: Following is the example of identifying the sensors and list all the available sensors on a device using android sensor framework.

STEP-1: Create activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:paddingLeft="10dp"
   android:paddingRight="10dp"
   tools:context=".MainActivity">
```

```
    <TextView
        android:id="@+id/tv1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="50dp"
        android:text="Sensors list"
        android:textSize="20dp"
        android:textStyle="bold"
        android:gravity="center"/>
</LinearLayout>
```

STEP-2: Create MainActivity.java file

```
package com.vjtech.androidsensorproject;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.widget.TextView;
import java.util.List;
public class MainActivity extends AppCompatActivity {
    SensorManager mgr;
    TextView txtlist;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mgr=(SensorManager) getSystemService(Context.SENSOR_SERVICE);
        txtlist=(TextView) findViewById(R.id.tv1);
        List<Sensor> str =mgr.getSensorList(Sensor.TYPE_ALL);
        StringBuilder buff=new StringBuilder();
        for(Sensor s:str)
        {
            buff.append(s.getName()+"\n");
        }
        txtlist.setText(buff);
    }
}
```

Async Tasks:

- Android AsyncTask is an abstract class provided by Android which gives us the liberty to perform heavy tasks in the background and keep the UI thread light thus making the application more responsive.
- Android application runs on a single thread when launched. Due to this single thread model tasks that take longer time to fetch the response can make the application nonresponsive.
- To avoid this, we use android AsyncTask to perform the heavy tasks in background on a dedicated thread and passing the results back to the UI thread. Hence use of AsyncTask in android application keeps the UI thread responsive at all times.
- The three generic types used in an android AsyncTask class are given below:
  - Params : The type of the parameters sent to the task upon execution
  - Progress : The type of the progress units published during the background computation
  - Result: The type of the result of the background computation

- To start an AsyncTask the following snippet must be present in the MainActivity class:

```
MyTask myTask = new MyTask();
myTask.execute();
```

- In the above snippet we've used a sample classname that extends AsyncTask and execute method is used to start the background thread.

  - The AsyncTask instance must be created and invoked in the UI thread.
  - The methods overridden in the AsyncTask class should never be called. They're called automatically
  - AsyncTask can be called only once. Executing it again will throw an exception.

Audio Capture:

- In android, MediaRecorder class will provide a functionality to record audio or video files.
- The android multimedia framework provides built-in support for capturing and encoding a variety of common audio and video formats.
- We have multiple ways to record audio or video but by using MediaRecorder class we can easily implement audio or video recording.
- In android, to record an audio we need to use device's microphone along with MediaRecorder class.
- In case, if we want to record video, we need to use device's camera along with MediaRecorder class.
- In order to use MediaRecorder class, you will first create an instance of MediaRecorder class. Its syntax is given below.

```
MediaRecorder myAudioRecorder = new MediaRecorder();
```

- Now you will set the source, output and encoding format and output file. Their syntax is given below:

```
myAudioRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
myAudioRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
myAudioRecorder.setAudioEncoder(MediaRecorder.OutputFormat.AMR_NB);
myAudioRecorder.setOutputFile(outputFile);
```

- After specifying the audio source and format and its output file, we can then call the two basic methods prepare and start to start recording the audio.

```
myAudioRecorder.prepare();
myAudioRecorder.start();
```

- Apart from above methods, MediaRecorder class provides a different type of methods to control audio and video recording based on requirements.

Method	Description
setAudioSource()	It is used to specify the source of audio to be recorded.

Method	Description
setVideoSource()	It is used to specify the source of the video to be recorded.
setOutputFormat()	It is used to specify the audio / video output format.
setAudioEncoder()	It is used to specify the audio encoder.
setVideoEncoder()	it is used to specify the video encoder.
setOutputFile()	It is used to specify the path of a recorded audio/video files to be stored.
stop()	It is used to stop the recording process.
start()	it is used to start the recording process.
release()	It is used to releases the resources which are associated with MediaRecorder object.

Camera in Android:

- In Android, Camera is a hardware device that allows capturing pictures and videos in your applications. Follow this tutorial to easily understand how to use a camera in your own Android App.
- Android provides the facility to work on camera by 2 ways:
  1. By Camera Intent
  2. By Camera API

1. By Using Camera Application:

- We can capture pictures without using the instance of Camera class. Here you will use an intent action type of MediaStore.ACTION_IMAGE_CAPTURE to launch an existing Camera application on your phone. In Android MediaStore is a type of DataBase which stores pictures and videos in android.

```
Intent cameraIntent = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
```

2. By Using Camera API:

- This class is used for controlling device cameras. It can be used to take pictures when you are building a camera application. Camera API works in following ways:

  1. **Camera Manager:** This is used to get all the cameras available in the device like front camera back camera each having the camera id.
  2. **CameraDevice:** You can get it from Camera Manager class by its id.
  3. **CaptureRequest:** You can create a capture request from camera device to capture images.
  4. **CameraCaptureSession:** To get capture requests from Camera Device create a CameraCaptureSession.
  5. **CameraCaptureSession.CaptureCallback**: This is going to provide the Capture session results

- First, you should declare the Camera requirement in your Manifest file if Camera is compulsory for your application and you don't want your application to be installed on a device that does not support Camera.
- Before you start development on your application you have to make sure that your Manifest has appropriate declarations in it that will allow you to use Camera feature in your application.

```
<uses-permission android:name=”android.permission.CAMERA”/>
```

- **Example:** Simple example of Camera

STEP-1: Create .XML file

```
<?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"
   android:padding="40dp"
   android:orientation="horizontal"
   tools:context=".MainActivity">
   <TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="CAMERA"
      android:id="@+id/text"
      android:textSize="20dp"
      android:gravity="center"/>
   <ImageView
      android:id="@+id/image"
      android:layout_width="350dp"
      android:layout_height="200dp"
      android:layout_below="@+id/text"
      android:layout_marginTop="81dp"
      android:src="@drawable/img"/>
   <Button
      android:id="@+id/photo"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_below="@+id/image"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="100dp"
      android:text="TAKE PHOTO" />
</RelativeLayout>
```

STEP-2: Create .java file:

```
package com.vjtech.androidcameraproject;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.provider.MediaStore;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {
   Button b1;
   ImageView imageView;
   int CAMERA_REQUEST=1;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      b1=findViewById(R.id.photo);
      imageView=findViewById(R.id.image);
      b1.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            Intent i=new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
            startActivityForResult(i,CAMERA_REQUEST);
         }
      });
   }
   @Override
   protected void onActivityResult(int requestCode, int resultCode, Intent data) {
      super.onActivityResult(requestCode, resultCode, data);
      if (requestCode==CAMERA_REQUEST)
      {
         Bitmap image= (Bitmap) data.getExtras().get("data");
         imageView.setImageBitmap(image);
      }
   }
}
```

STEP-3: Manifest File

```
<uses-permission android:name="android.permission.CAMERA"
   android:required="true"
   android:requiredFeature="true"/>
```

Android Bluetooth:

- In android, Bluetooth is a communication network protocol, which allows devices to connect wirelessly to exchange the data with other Bluetooth devices
- In android applications by using Bluetooth API's we can implement Bluetooth functionalities, such as searching for the available Bluetooth devices, connecting with the devices and managing the data transfer between devices within the range.
- By using android Bluetooth APIs in android applications, we can perform the following functionalities.
  - Scan for the available Bluetooth devices within the range
  - Use local Bluetooth adapter for paired Bluetooth devices
  - Connect to other devices through service discovery
  - Transfer data to and from other devices
  - Manage multiple connections
- **Android Bluetooth API:**
- The android.bluetooth package provides a lot of interfaces classes to work with Bluetooth such as:
  - BluetoothAdapter
  - BluetoothDevice
  - BluetoothSocket
  - BluetoothServerSocket
  - BluetoothClass
  - BluetoothProfile
  - BluetoothProfile.ServiceListener
  - BluetoothHeadset
  - BluetoothA2dp
  - BluetoothHealth
  - BluetoothHealthCallback
  - BluetoothHealthAppConfiguration
- **Android BluetoothAdapter Class**
- In android, we can perform Bluetooth related activities by using BluetoothAdapter class in our applications.
- By using BluetoothAdapter object, we can interact with device's Bluetooth adapter to perform Bluetooth related operations. In case, if device does not contain any Bluetooth adapter, then it will return null.
- BluetoothAdapter class provides many constants. Some of them are as follows:
  - String ACTION_REQUEST_ENABLE
  - String ACTION_REQUEST_DISCOVERABLE
  - String ACTION_DISCOVERY_STARTED

- String ACTION_DISCOVERY_FINISHED

- Methods of BluetoothAdapter class:
    1. **static synchronized BluetoothAdapter getDefaultAdapter()** returns the instance of BluetoothAdapter.
    2. **boolean enable()** enables the bluetooth adapter if it is disabled
    3. **boolean isEnabled()** returns true if the bluetooth adapter is enabled.
    4. **boolean disable()** disables the bluetooth adapter if it is enabled
    5. **String getName()** returns the name of the bluetooth adapter
    6. **boolean setName(String name)** changes the bluetooth name.
    7. **int getState()** returns the current state of the local bluetooth adapter
    8. **Set <BluetoothDevice> getBondedDevices()** returns a set of paired (bonded) BluetoothDevice objects.
    9. **boolean startDiscovery()** starts the discovery process

- **Example:**

STEP-1 Activity_main.xml file

```
<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"/>

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bluetooth Connectivity"
    android:textStyle="bold"
    android:textSize="30dp"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="30dp"
    android:textColor="#f00"
    android:padding="20dp"
    android:background="#FD9696"/>
  <Button
    android:id="@+id/b1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:text="ON"
    android:onClick="On"
    android:textSize="30dp"
    android:textStyle="bold"
    android:layout_marginTop="10dp" />
```

```
    <Button
        android:id="@+id/b2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OFF"
        android:textStyle="bold"
        android:onClick="Off"
        android:textSize="30dp"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="20dp"/>
    <Button
        android:id="@+id/b3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="GET Visible"
        android:textSize="30dp"
        android:onClick="getVisible"
        android:textStyle="bold"
        android:layout_gravity="center_horizontal"
        android:layout_margin="20dp"/>
    <Button
        android:id="@+id/b4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="LIST DEVICES"
        android:textStyle="bold"
        android:onClick="device"
        android:textSize="30dp"
        android:layout_gravity="center_horizontal"/>
    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/list"
        android:layout_marginTop="20dp"/>
</LinearLayout>
```

STEP-2 MainActivity.java file

```
public class MainActivity extends AppCompatActivity
{
   BluetoothAdapter BA;
   Button button1, button2, button3, button4;
   ListView list1;
   private Set<BluetoothDevice>pairedDevices;
   protected void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
```

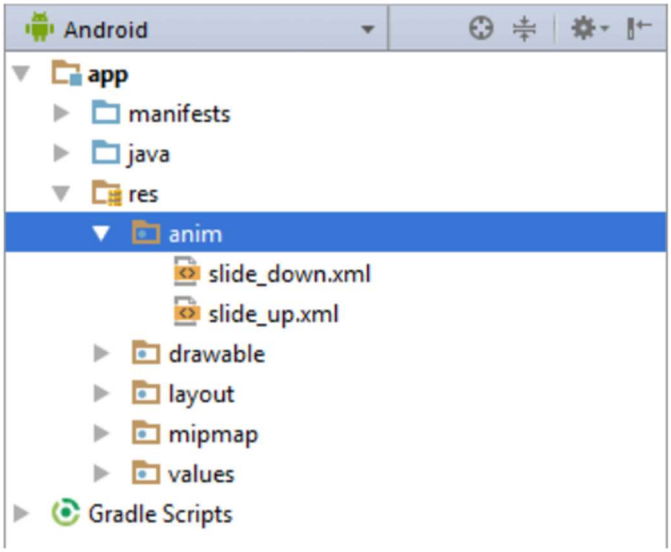
```
    setContentView(R.layout.activity_main);
    BA = BluetoothAdapter.getDefaultAdapter();
    button1 = findViewById(R.id.b1);
    button2 = findViewById(R.id.b2);
    button3 = findViewById(R.id.b3);
    button4 = findViewById(R.id.b4);
    list1 = findViewById(R.id.list);
  }
  public void On(View v)
  {
    if (!BA.isEnabled())
    {
      Intent turnon = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
      startActivityForResult(turnon, 0);
      Toast.makeText(getApplicationContext(),"Turn on",Toast.LENGTH_SHORT).show();
    }
    else
    {
      Toast.makeText(getApplicationContext(),"Bluetooth is Already Turned on",Toast.LENGTH_SHORT).show();
    }
  }
  public void Off(View v)
  {
    BA.disable();
    Toast.makeText(getApplicationContext(),"Turned off", Toast.LENGTH_SHORT).show();
  }
  public void getVisible(View v)
  {
    Intent getVisible = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
    startActivityForResult(getVisible, 0);
  }
  public void device(View v)
  {
    pairedDevices = BA.getBondedDevices();
    ArrayList list = new ArrayList();
    for (BluetoothDevice bt : pairedDevices)
      list.add((bt.getName()));
    final ArrayAdapter adapter=new ArrayAdapter(this,android.R.layout.simple_dropdown_item_1line,list);
    list1.setAdapter(adapter);

    Toast.makeText(getApplicationContext(),"Showing paired Device",Toast.LENGTH_SHORT).show();
  }
}
```

STEP-3 AndroidManifest.xml file

```
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.INTERNET"/>
```

Android Animation

- In android, Animations are used to change the appearance and behavior of the objects over a particular interval of time. The animations will provide a better look and feel high-quality user interface for our applications.
- Generally, the animations are useful when we want to notify users about the changes happening in our app, such as new content loaded or new actions available, etc.
- We have a different type of animations available in android, here we will discuss the most commonly used android animations such as zoom in / zoom out, fade in / fade out, slide up / slide down and rotate clockwise or anti-clockwise, etc. with examples.
- Create XML File to Define Animation: We need to create an XML file that defines the type of animation to perform in a new folder anim under res directory (res à anim à animation.xml) with the required properties. In case, anim folder not exists in res directory, create a new one.
- Following is the example of creating XML files under anim folder to define slide up / down animation properties.



- The XML files will contain the code like as shown below based on the type of animation.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator">
    <scale
        android:duration="500"
        android:fromXScale="1.0"
        android:fromYScale="0.0"
        android:toXScale="1.0"
        android:toYScale="1.0" />
</set>
```

- In case if we want to use different type of animations such as fade in / out, zoom in / out, etc. we need to create a new xml file in anim folder with required properties.
- The following are some of the important animation attributes that will help us to change the behavior of animation in our application.

Attributes	Description
android:duration	It is used to define the duration of the animation to complete.
android:startOffset	It is used to define the waiting time before the animation starts.
android:interpolator	It is used to define the rate of change in animation.
android:repeatMode	It is useful when we want to repeat our animation.
android:repeatCount	It is used to define the number of times the animation repeats. In case if we set **infinite**, the animation will repeat infinite times.
android:fillAfter	It is used to define whether to apply animation transformation after the animation completes or not.

- **Android Load and Start the Animation:**
- In android, we can perform animations by using AnimationUtils component methods such as loadAnimation(). Following is the code snippet of loading and starting an animation using loadAnimation() and startAnimation() methods.

```
ImageView img = (ImageView)findViewById(R.id.imgvw);
Animation aniSlide =
AnimationUtils.loadAnimation(getApplicationContext(),R.anim.slide_up);
img.startAnimation(aniSlide);
```

- If you observe above code snippet, we are adding an animation to the image using loadAnimation() method. The second parameter in loadAnimation() method is the name of our animation xml file.

- Here we used another method startAnimation() to apply the defined animation to imageview object.

Different Types of Android Animations:

- In android, we have different types of animations such as Fade In / Fade Out, Zoom In / Zoom Out, Slide Up / Slide Down, Rotate in Clockwise or Anti-Clockwise, etc.

1. Android Fade In / Out Animation:

- To use Fade In or Fade Out animations in our android applications, we need to define a new XML file with **<alpha>** tag like as shown below.
- For Fade In animation, we need to increase the alpha value from 0 to 1 like as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android" android:interpolator="
@android:anim/linear_interpolator">
  <alpha
      android:duration="2000"
      android:fromAlpha="0.1"
      android:toAlpha="1.0">
  </alpha>
</set>
```

- For Fade Out animation, we need to decrease the alpha value from 1 to 0 like as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
  android:interpolator="@android:anim/linear_interpolator">
  <alpha
      android:duration="2000"
      android:fromAlpha="1.0"
      android:toAlpha="0.1" >
  </alpha>
</set>
```

2. Android Slide Up / Down Animation:

- To use Slide Up or Slide Down animations in our android applications, we need to define a new XML file with **<scale>** tag like as shown below.
- For Slide Up animation, we need to set android:fromYScale="1.0" and android:toYScale="0.0" like as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
android:interpolator="@android:anim/linear_interpolator">
   <scale
      android:duration="500"
      android:fromXScale="1.0"
      android:fromYScale="1.0"
      android:toXScale="1.0"
      android:toYScale="0.0" />
</set>
```

- For Slide Down animation, we need to set android:fromYScale="0.0" and android:toYScale="1.0" like as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android" android:interpolator="
@android:anim/linear_interpolator">
   <scale
      android:duration="500"
      android:fromXScale="1.0"
      android:fromYScale="0.0"
      android:toXScale="1.0"
      android:toYScale="1.0" />
</set>
```

3. Android Zoom In / Out Animation

- To use Zoom In or Zoom Out animations in our android applications, we need to define a new XML file with <scale> tag like as shown below.
- For Zoom In animation, we need to set android:pivotX="50%" and android:pivotY="50%" to perform the zoom from the centre of the element.
- Also, we need to use fromXScale, fromYScale attributes to define the scaling of an object and we need keep these values lesser than toXScale, toYScale like as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
   <scale
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:duration="1000"
      android:fromXScale="2"
      android:fromYScale="2"
      android:pivotX="50%"
      android:pivotY="50%"
      android:toXScale="4"
      android:toYScale="4" >
   </scale>
</set>
```

- In android, Zoom Out animation is same as Zoom In animation but fromXScale, fromYScale attribute values must be greater than toXScale, toYScale like as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
   <scale
      android:duration="2500"
      android:fromXScale="1.0"
      android:fromYScale="1.0"
      android:pivotX="50%"
      android:pivotY="50%"
      android:toXScale=".2"
      android:toYScale=".2" />
</set>
```

4. Android Rotate Clockwise / Anti Clockwise Animation

- To use Rotate animation in our android applications, we need to define a new XML file with <rotate> tag like as shown below.
- To Rotate animation in Clockwise, we need to set android:fromDegrees and android:toDegrees property values and these will define a rotation angles like as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android" android:interpolator="
@android:anim/cycle_interpolator">
   <rotate android:fromDegrees="0"
      android:toDegrees="360"
      android:pivotX="50%"
      android:pivotY="50%"
      android:duration="5000" />
</set>
```

- To Rotate animation in Anti Clockwise, we need to set android:fromDegrees and android:toDegrees property values and these will define a rotation angles like as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android" android:interpolator="
@android:anim/cycle_interpolator">
   <rotate android:fromDegrees="360"
      android:toDegrees="0"
      android:pivotX="50%"
      android:pivotY="50%"
      android:duration="5000" />
</set>
```

SQLite Database:

- SQLite is an open-source lightweight relational database management system (RDBMS) to perform database operations, such as storing, updating, retrieving data from the database.
- Android SQLite is the mostly preferred way to store data for android applications.
- By default, Android comes with built-in SQLite Database support so we don't need to do any configurations.
- Just like we save the files on the device's internal storage, Android stores our database in a private disk space that's associated with our application and the data is secure, because by default this area is not accessible to other applications.
- he SQLiteDatabase namespace defines the functionality to connect and manage a database. It provides functionality to create, delete, manage and display database content.
- The package android.database.sqlite contains all the required APIs to use an SQLite database in our android applications.
- Simple steps to create a database and handle are as follows.
  1. Create "SQLiteDatabase" object.
  2. Open or Create a database and create a connection.
  3. Perform insert, update or delete operation.
  4. Create a Cursor to display data from the table of the database.
  5. Close the database connectivity.

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

Step 2: object Open or Create a database and create a connection.

```
db=openOrCreateDatabase(String path, int mode, SQLiteDatabase.CursorFactoryfact ory);
```

- 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);
```

Where:

- int mode : operating mode. Use 0 or "MODE_PRIVATE" for the default operation, or "CREATE_IF_NECESSARY" if you like to give an option that "if a database is not there, create it"
- CursorFactory factory : An optional factory class that is called to instantiate a cursor when query is called.

Step 3: Execute DDL command

```
db.execSQL(String sql)throws SQLException
```

- This command is used to execute a single SQL statement that doesn't return any data means other than SELECT or any other.

```
db.execSQL("Create Table Temp (id Integer, name Text)");
```

- In the above example, it takes "CREATE TABLE" statement of SQL. This will create a table of "Integer" & "Text" fields.
- Try and Catch block is required while performing this operation. An exception that indicates there was an error with SQL parsing or execution.

Step 4: Create an object of "ContentValues" and initiate it

```
ContentValues values=new ContentValues();
```

- This class is used to store a set of values. We can also say, it will map ColumnName and relevant ColumnValue

```
values.put("id", eid.getText().toString());
values.put("name", ename.getText().toString());
```

Step 5: Perform Insert Statement.

```
insert(String table, String nullColumnHack, ContentValues values)
```

Where:

- **String table :** Name of table related to the database.
- **String nullColumnHack:** If not set to null, the nullColumnHack parameter provides the name of nullable column name to explicitly insert a NULL into in the case here your values are empty.
- **ContentValues values:** This map contains the initial column values for the row

This method returns a long. The row ID of the newly inserted row, or -1 if an error occurred.
Example: `db.insert("temp", null, values);`

Step 6: Create Cursor

- This interface provides random read-write access to the result set returned by a database query

```
Cursor c=db.rawQuery(String sql, String[] selectionArgs)
```

- **String sql** : The SQL query
- **String []selectionArgs**: You may include ? s in where clause in the query, which will be replaced by the values from selectionArgs. The values will be bound as Strings
- Example, Cursor c=db.rawQuery("SELECT * FROM temp", null);
- Methods:
  1. **moveToFirst** : Moves cursor pointer at a first position of a result set
  2. **moveToNext** : Moves cursor pointer next to the current position.
  3. **isAfterLast** : Returns false, if the cursor pointer is not at last position of a result set
- **Example**

```
c.moveToFirst();
while(!c.isAfterLast())
{
    //statements
    c.moveToNext();
}
```

Step 7: Close Cursor and Close Database connectivity

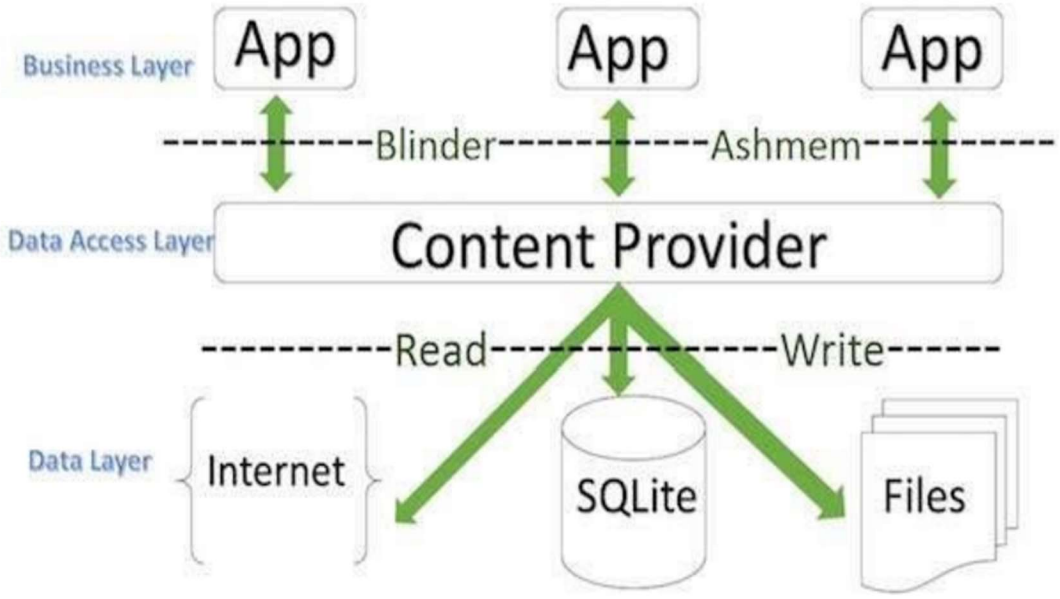
- It is very important to release our connections before closing our activity. It is advisable to release the Database connectivity in "onStop" method. And Cursor connectivity after use it.

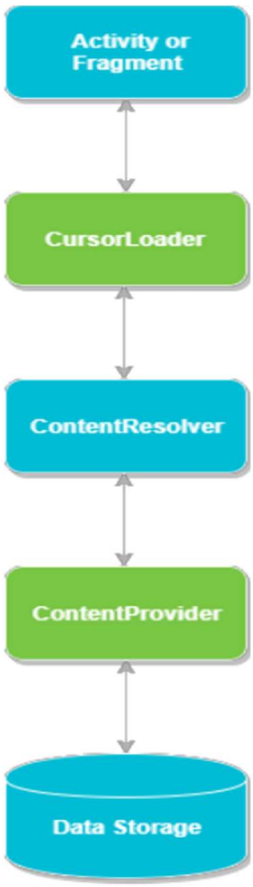
Content Providers:

- In android, Content Provider will act as a central repository to store the applications data in one place and make that data available for different applications to access whenever it's required
- In android, we can configure Content Providers to allow other applications securely access and modify our app data based on our requirements.
- Generally, the Content Provider is a part of an android application and it will act as more like relational database to store the app data. We can perform a multiple operations like insert, update, delete and edit on the data stored in content provider using insert(), update(), delete() and query() methods.
- In android, we can use content provider whenever we want to share our app data with other apps and it allow us to make a modifications to our application data without effecting other applications which depends on our app
- In android, content provider is having different ways to store app data. The app data can be stored in a SQLite database or in files or even over a network based on our requirements. By using content providers we can manage data such as audio, video, images and personal contact information.
- We have a different type of access permissions available in content provider to share the data. We can set a restrict access permissions in content provider to restrict data access limited to only our application and we can configure different permissions to read or write a data.

Access Data from Content Provider:

- To access a data from content provider, we need to use ContentResolver object in our application to communicate with the provider as a client. The ContentResolver object will communicate with the provider object (ContentProvider) which is implemented by instance of class.
- Generally, in android to send a request from UI to ContentResolver we have another object called CursorLoader which is used to run the query asynchronously in background. In android application the UI components such as Activity or Fragment will call a CursorLoader to query and get a required data from ContentProvider using ContentResolver.
- The ContentProvider object will receive a data requests from client, performs the requested actions (create, update, delete, retrieve) and return the result.
- Following is the pictorial representation of requesting an operation fromUI using Activity or Fragment to get the data from ContentProvider object.





- This is how the interaction will happen between android application UI and content providers to perform required actions to get a data.

Content URIs

- In android, Content URI is an URI which is used to query a content provider to get the required data. The Content URIs will contain the name of entire provider (authority) and the name that points to a table (path)
- Generally the format of URI in android applications will be like as shown below content://authority/path
- Following are the details about various parts of an URI in android application.
- **content:// -** The string content:// is always present in the URI and it is used to represent the given URI is a content URI.
- **authority -** It represents the name of content provider, for example phone, contacts, etc. and we need to use fully qualified name for third party content providers like com.vjtech.contactprovider
- **path** - It represents the table's path.
- The ContentResolver object use the URI's authority to find the appropriate provider and send the query objects to the correct provider. After that ContentProvider uses the path of content URI to choose the right table to access.
- Following is the example of simple example of URI in android applications. content://contacts_info/users
- Here the string content:// is used to represent URI is a content URI, contacts_infostring is the name of provider's authority and users string is the table's path.

Creating a Content Provider

- To create a content provider in android applications we should follow below steps.

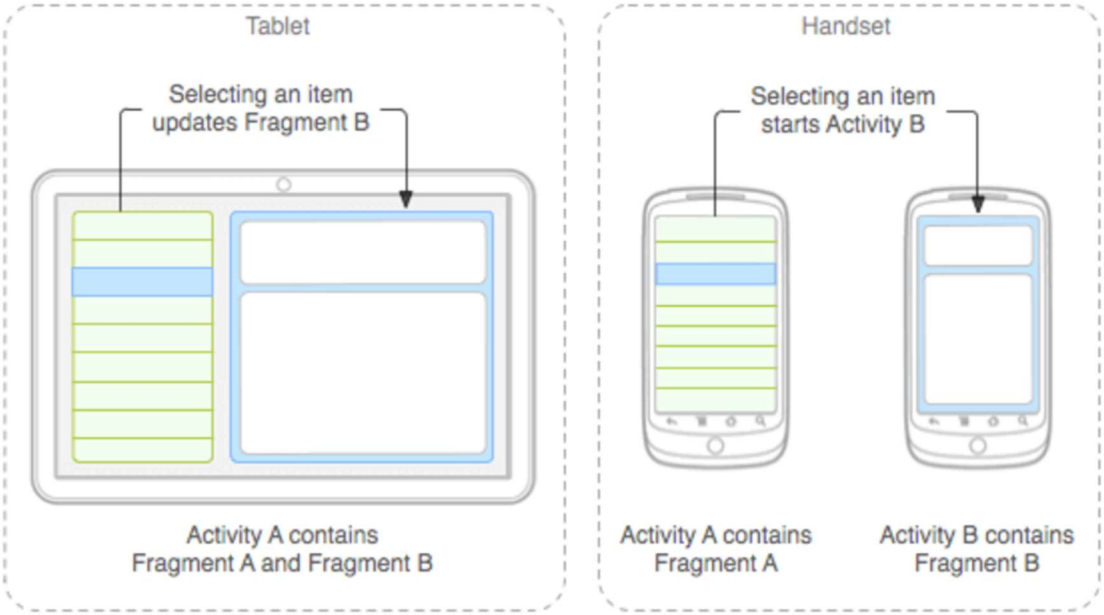
1. We need to create a content provider class that extends the ContentProvider base class.
2. We need to define our content provider URI to access the content.
3. The ContentProvider class defines a six abstract methods (insert(), update(), delete(), query(), getType()) which we need to implement all these methods asa part of our subclass.
4. We need to register our content provider in AndroidManifest.xml using <provider> tag

- Following are the list of methods which need to implement as a part of ContentProvider class.

- **query()** - It receives a request from the client. By using arguments it will get a data from requested table and return the data as a Cursor object.
- **insert()** - This method will insert a new row into our content provider and it will return the content URI for newly inserted row.
- **update()** - This method will update an existing rows in our content provider and it return the number of rows updated.
- **delete()** - This method will delete the rows in our content provider and it return the number of rows deleted.
- **getType()** - This method will return the MIME type of data to given content URI
- **onCreate()** - This method will initialize our provider. The android system will call this method immediately after it creates our provider.

Fragment in Android:

- In Android, Fragments are the modular section of activity design and these are used to represent the behavior of user interface (UI) in an activity.
- By using fragments, we can create flexible UI designs that can be adjusted based on the device screen size such as tablets, smartphones.
- We can build multi-pane UI by combining multiple fragments in a single activity and we can reuse the same fragment in multiple activities. The fragment has its own lifecycle call-backs and accepts its own input events.
- We can add or remove fragments in an activity while the activity is running. In android, the fragment will act as a sub-activity and we can reuse it in multiple activities.
- The fragment must be included in an activity due to that the fragment lifecycle will always be affected by the host activity life cycle.
- In case if we pause an activity, all the fragments related to an activity will also be stopped.
- In android, we can insert the fragment into activity layout by using <fragment> element and by dividing the layout of activity into fragments, we can modify the appearance of an app design at runtime. We can also implement a fragment without having any user interface (UI).
- It's an optional to use fragments into activity but by doing this it will improve the flexibility of our app UI and make it easier to adjust our app design based on the device size.
- Following is the example of defining multiple fragments in single activity for the tablet design to display the details of an item which we selected in the app, but separated for mobile design.



- If you observe above example for Tablet we defined an Activity A with two fragments such as one is to show the list of items and second one is to show the details of item which we selected in first fragment.
- For Handset device, there is no enough space to show both the fragments in single activity, so the Activity A includes first fragment to show the list of items and the Activity B which includes another fragment to display the details of an item which is selected in Activity A.

Sens SMS program in Android:

STEP-1: Create .xml file

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <TextView
        android:layout_width="250dp"
        android:gravity="center"
        android:layout_height="wrap_content"
        android:text="Messages"
        android:textSize="25dp"
        android:textColor="@color/white"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="15dp"
        android:background="#2196F3"
        />
    <TextView
        android:layout_width="250dp"
        android:gravity="center"
        android:layout_height="wrap_content"
        android:text="Phone No:"
        android:textSize="25dp"
        android:textColor="@color/white"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="100dp"
        android:background="@color/teal_200"
        />
    <EditText
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:id="@+id/phone_no"
        android:layout_marginTop="20dp"
        android:ems="10"
        android:inputType="number"
        android:layout_gravity="center_horizontal"
        android:backgroundTint="@color/purple_500"/>
    <TextView
        android:layout_width="250dp"
        android:gravity="center"
```

```
        android:layout_height="wrap_content"
        android:text="Message:"
        android:textSize="25dp"
        android:textColor="@color/white"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="50dp"
        android:background="@color/teal_200"
        />
    <EditText
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:id="@+id/msg"
        android:layout_marginTop="20dp"
        android:inputType="text"
        android:ems="50"
        android:layout_gravity="center_horizontal"
        android:backgroundTint="@color/purple_500"/>
    <com.google.android.material.button.MaterialButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/sent"
        android:text="SENT"
        android:textSize="20dp"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="35dp"/>
</LinearLayout>
```

STEP-2: Create .java file

```
package com.vjtech.sendsms;


import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;

import android.os.Bundle;

import android.telephony.SmsManager;

import android.view.View;

import android.widget.EditText;

import android.widget.Toast;

import com.google.android.material.button.MaterialButton;

public class MainActivity extends AppCompatActivity {

    MaterialButton sent,recive;

    EditText phone,msg;
```

```
  SmsManager manager;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    sent=findViewById(R.id.sent);
   // recive=findViewById(R.id.recive);
    phone=findViewById(R.id.phone_no);
    msg=findViewById(R.id.msg);

    sent.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        try{
          manager=SmsManager.getDefault();
          manager.sendTextMessage(phone.getText().toString(),null,msg.getText().toString(),null,null);
          Toast.makeText(MainActivity.this, "Sms Send Successfully ", Toast.LENGTH_SHORT).show();
          phone.setText(null);
          msg.setText(null);
        }
        catch (Exception e){
          Toast.makeText(MainActivity.this, "Sms Send Not Successfully ", Toast.LENGTH_SHORT).show();
        }
      }
    });
  }
}
```

STEP-3: AndroidManifest.xml file

```
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
```