Mobile Application Development

Unit- 5

Activity And Multimedia with databases

Mrs. Chavan P.P.

Intent

Android uses Intent for communicating between the components of an Application and also from one application to another application.

For example: Intent facilitate you to redirect your activity to another activity on occurrence of any event. By calling, startActivity() you can perform this task.

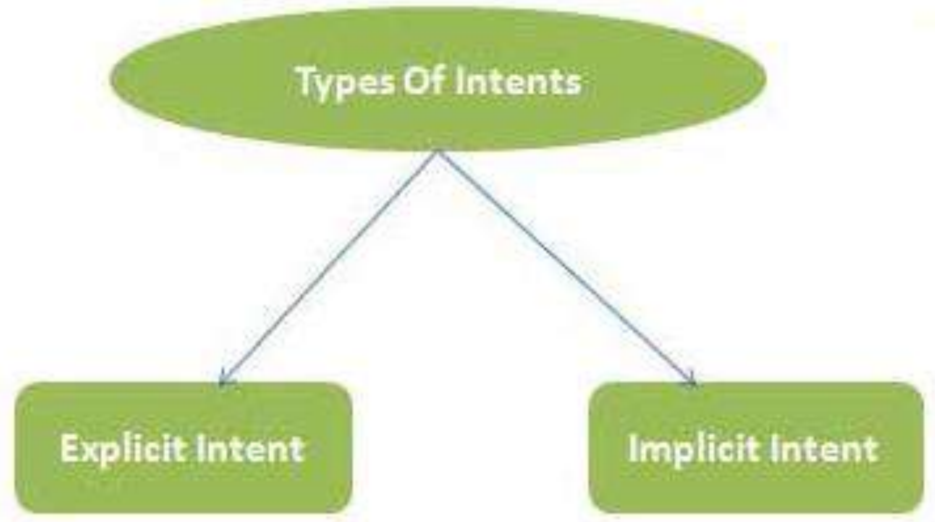
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);

startActivity(intent);

An Intent can be used to:

1. Start an activity
2. Start a service
3. Deliver a broadcast

Types of Intents:



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.

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

Types of Intents:

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.

```
Intent intentObj = new Intent(Intent.ACTION_VIEW);
intentObj.setData(Uri.parse("https://www.google.com"));
startActivity(intentObj);
```

Intent Filters

Intent Filter are the components which decide the behavior of an intent. 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).

Syntax of Intent Filters:

Intent filter is declared inside Manifest file for an 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

A label represents the title of an activity on the toolbar. You can have different Labels for different Activities as per your requirement or choice. The label should be set as a reference to a string resource.

Syntax of Intent Filters:

Intent filter is declared inside Manifest file for an Activity.

```
<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" />
 </intent-filter>
</activity>
```

Elements In Intent Filter:

There are following three elements in an intent filter:

1. Action
2. Data
3. Category

1. Action:

It represent an activities action, what an activity is going to do. Action is a string that specifies the action to perform.It is declared with the name attribute as given below

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

There are few common actions for starting an activity like ACTION_VIEW

ACTION_VIEW: This is used in an Intent with startActivity(). This helps when you redirect to see any website, photos in gallery app or an address to view in a map app.

```
Intent intentObj = new Intent(Intent.ACTION_VIEW);
intentObj.setData(Uri.parse("https://www.google.com"));
startActivity(intentObj);
```

Elements In Intent Filter:

2. Data:

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

The syntax of data attribute is as follows:

```
<data android:scheme="string"
android:host="string"
android:port="string"
android:path="string"
android:pathPattern="string"
android:pathPrefix="string"
android:mimeType="string" />
```

Elements In Intent Filter:

ACTION	DATA	MEANING
Intent.ACTION_CALL	tel:phone_number	Opens phone application and calls phone number
Intent.ACTION_DIAL	tel:phone_number	Opens phone application and dials (but doesn't call) phone_number
Intent.ACTION_DIAL	voicemail:	Opens phone application and dials (but doesn't call) the voice mail number.
Intent.ACTION_VIEW	geo:lat,long	Opens the maps Application centered on (lat, long).
Intent.ACTION_VIEW	geo:0,0?q=address	Opens the maps application centered on the specified address.
Intent.ACTION_VIEW	http://url https://url	Opens the browser application to the specified address.
Intent.ACTION_WEB_SEARCH	plain_text	Opens the browser application and uses Google search for given string

Elements In Intent Filter:

3. 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" />
```

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.

Activity Lifecycle

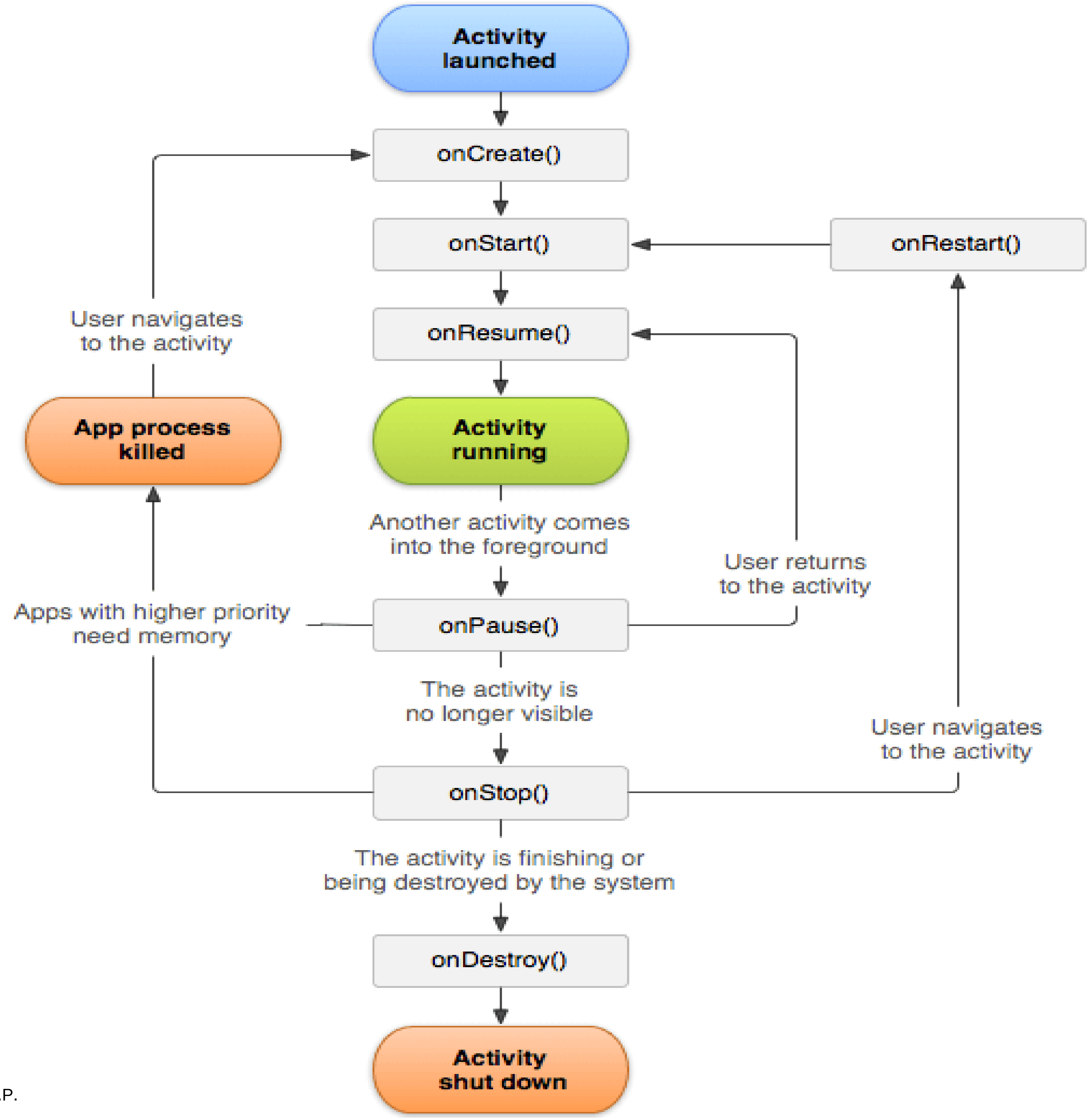
In android, **Activity** represents a single screen with a user interface (UI) of an application and it will acts an entry point for users to interact with an app.

the android apps will contain multiple screens and each screen of our application will be an extension of Activity class. By using activities, we can place all our android application UI components in a single screen.

From the multiple activities in android app, one activity can be marked as a **main activity** and that is the first screen to appear when we launch the application. In android app each activity can start another activity to perform different actions based on our requirements. **Android Activity Lifecycle** is controlled by 7 methods of android.app.Activity class.

an activity goes through a series of states during its lifetime. Android system initiates its program within an **Activity** starting with a call on onCreate() callback method.

Method	Description
onCreate	called when activity is first created.
onStart	called when activity is becoming visible to the user.
onResume	called when activity will start interacting with the user.
onPause	called when activity is not visible to the user.
onStop	called when activity is no longer visible to the user.
onRestart	called after your activity is stopped, prior to start.
onDestroy	called before the activity is destroyed.


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

The Activity lifecycle consists of 7 methods:

onCreate(): It is called when an activity is first created. When a user opens the app then some Activity is created. You have to implement this method in every activity because, inside this method, all the necessary components of your activity will be initialized. Here the initialization of your application's UI is done.

onStart(): This method is called when an activity becomes visible to the user. When all the initialization is done by the *onCreate()* method, then this method is called.

onResume(): It is called just before the user starts interacting with the application. Most of the core functionalities of the app are implemented in this method.

onPause(): It is called when the activity is paused i.e. it is mostly called when you press the back or home button of your Android device. It is an indication that the user is leaving the activity and starting some other activity.

onStop(): It is called when the activity is no longer visible to the user. If you are starting a new activity, or some existing activity is entering into onResume() state, then the current activity will not be visible to the user and is stopped.

onRestart(): It is called when the activity in the stopped state is about to start again. By doing so, the state of the activity from the time it was stopped will be restored.

onDestroy(): It is called when the activity is totally destroyed i.e. when you clear the application stack then onDestroy() will be called and all the states of the activity will be destroyed.

```
package com.example.piyush.myapplication;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toast toast = Toast.makeText( getApplicationContext(), "onCreate Called",
             Toast.LENGTH_LONG).show();
   }
   protected void onStart()
   {
       super.onStart();
       Toast toast = Toast.makeText(getApplicationContext(), "onStart Called",
             Toast.LENGTH_LONG).show();
   }
```

```
@Override
   protected void onRestart()
   {
       super.onRestart();
       Toast toast = Toast.makeText(getApplicationContext(), "onRestart Called",
              Toast.LENGTH_LONG).show();
     }
     protected void onPause()
     {
        super.onPause();
        Toast toast = Toast.makeText(getApplicationContext(), "onPause Called",
              Toast.LENGTH_LONG).show();
     }
     protected void onResume()
     {
        super.onResume();
        Toast toast = Toast.makeText(getApplicationContext(), "onResume Called",
              Toast.LENGTH_LONG).show();
     }
```

```
    protected void onStop()
    {
            super.onStop();
            Toast toast = Toast.makeText(getApplicationContext(), "onStop Called",
               Toast.LENGTH_LONG).show();
    }
    protected void onDestroy()
    {
            super.onDestroy();
            Toast toast = Toast.makeText(getApplicationContext(), "onDestroy Called",
                       Toast.LENGTH_LONG).show();
    }
}
```

BroadcastReceiver

A broadcast receiver is an Android component which allows to register and listen for device orientation changes like sms messsage recieved , phone call recieved/pick/cut ,battery status changed, the Wi-Fi came on.

With the help of broadcast receiver you will catch Android operating system specific events and then you can code your application for that event.

Receiving Broadcasts

In android, we can receive broadcasts by registering in two ways.

One way is by **registering a broadcasts** using android application manifest file (AndroidManifest.xml). We need to specify <receiver> element in apps manifest file like as shown below.

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

The above statement will fire the defined system broadcast event

Another way is to register a receiver dynamically via **Context.registerReceiver()** method.

To register broadcast receiver we need to extend our class using BroadcastReceiver and need to implement a **onReceive(Context, Intent)** method like as shown below.

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

Sending Broadcasts

In android, we can send a broadcasts in apps using three different ways, those are

Method	Description
sendOrderedBroadcast(Intent, String)	This method is used to send broadcasts to one receiver at a time.
sendBroadcast(Intent)	This method is used to send broadcasts to all receivers in an undefined order.
LoadBroadcastManager.sendBroadcast	This method is used to send broadcasts to receivers that are in the same app as the sender.

System Broadcasts

In android, several system events are defined as final static fields in the Intent class. Following are the some of system events available in android applications.

Event	Description
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.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.

Content Providers

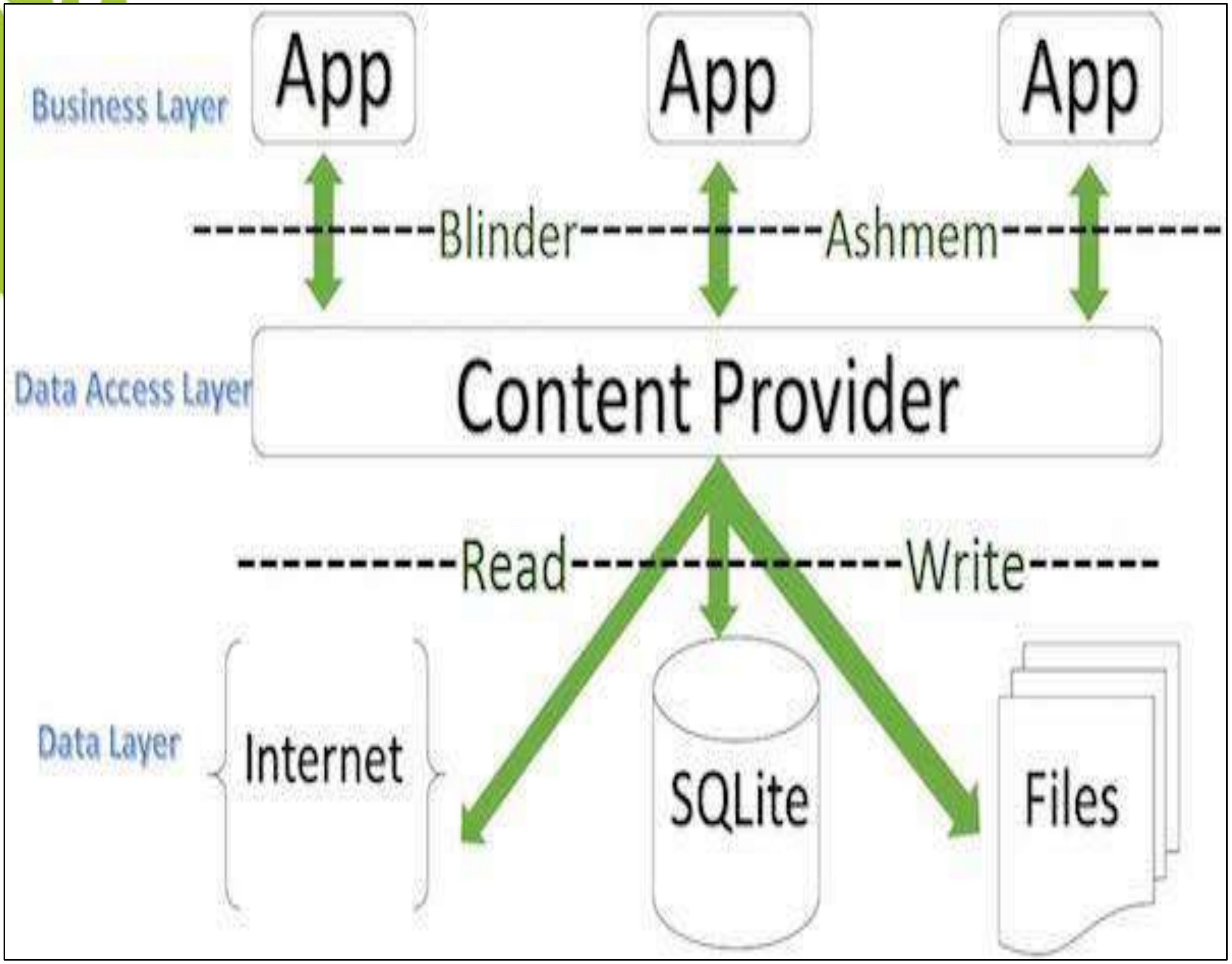
Content Provider will act as a central repository to store the applications data in one place and make that data available for different applications to access whenever it’s required.

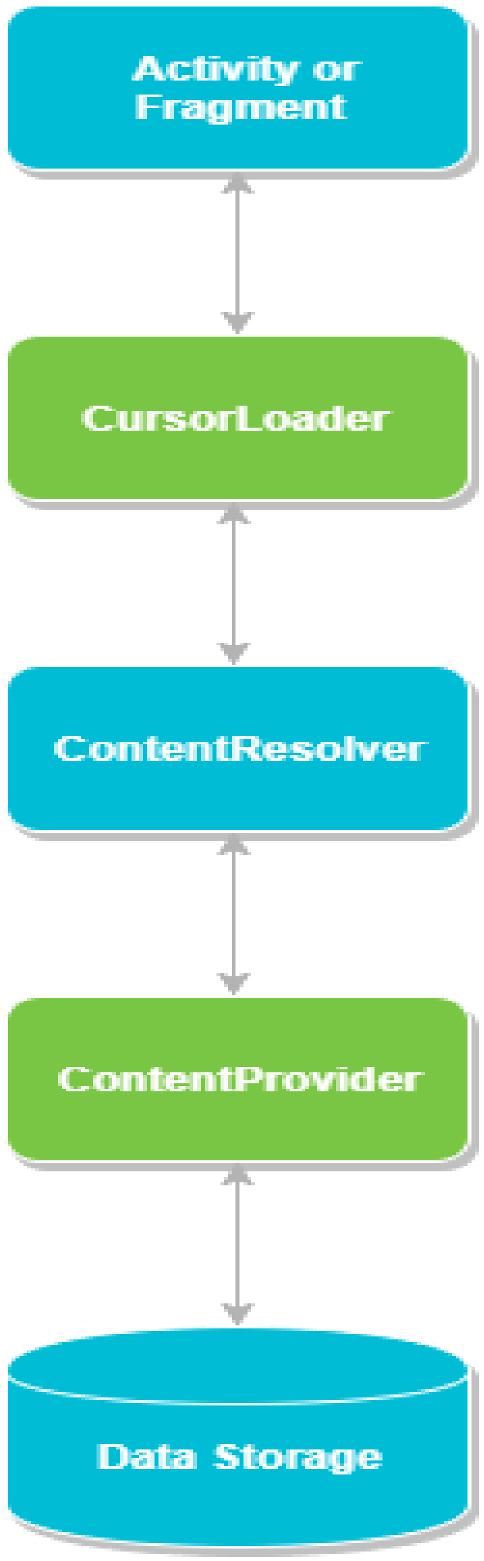
The **Content Provider** is a part of an android application and it will act as more like relational database to store the app data. We can perform multiple operations like insert, update, delete and edit on the data stored in content provider using **insert()** , **update() , delete()** and **query()** methods.

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.

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. 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. The UI components such as Activity or Fragment will call a **CursorLoader** to query and get a required data from ContentProvider using ContentResolver.


Business Layer
App
App
App
Blinder
Ashmem
Data Access Layer
Content Provider
Read
Write
Data Layer
Internet
SQLite
Files


Activity or Fragment
CursorLoader
ContentResolver
ContentProvider
Data Storage

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**).

General format of URI

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.tutlane.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

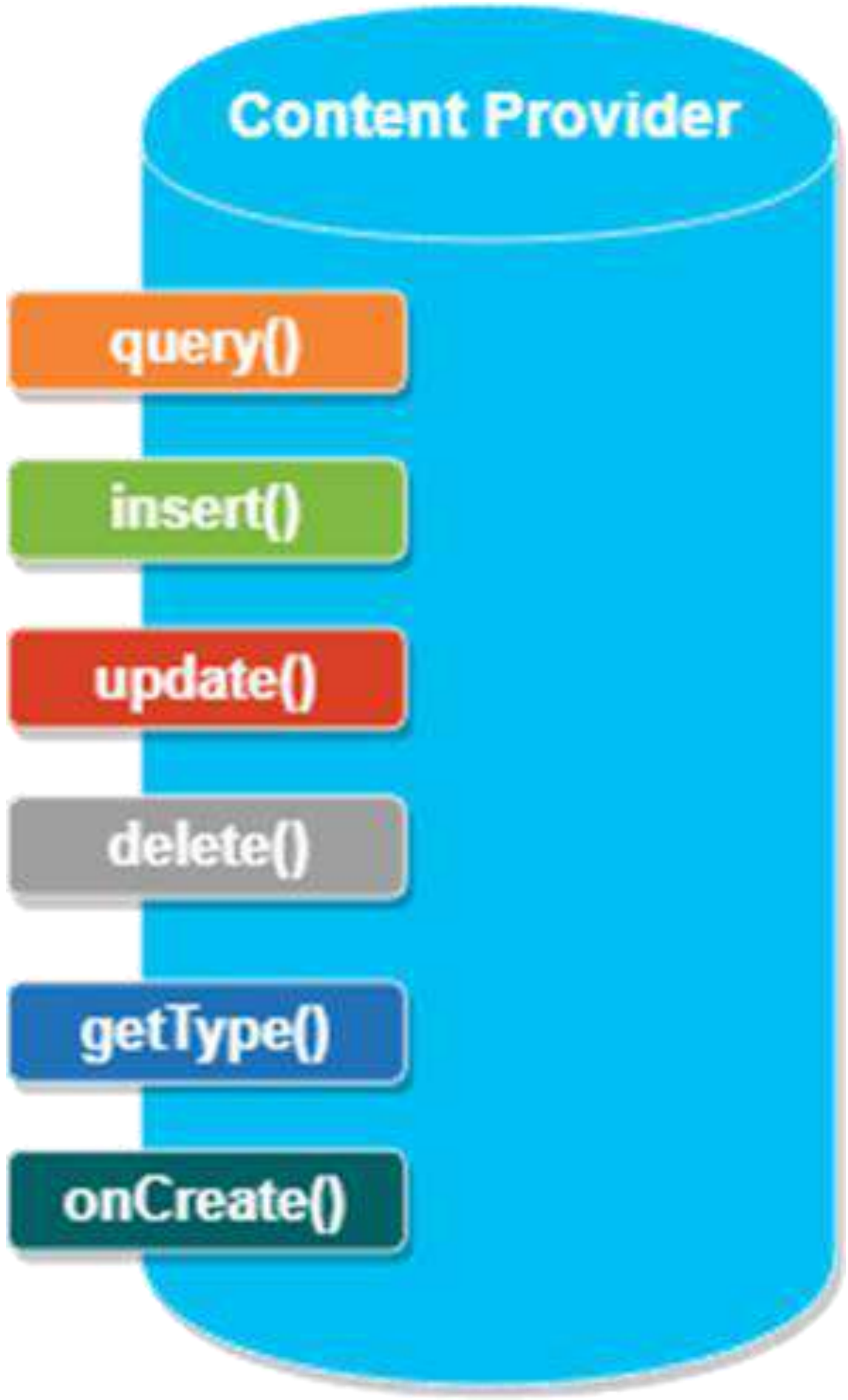
Here the string **content://** is used to represent URI is a content URI, **contacts_info** string is the name of provider’s authority and **users** string is the table’s path.

Creating a Content Provider

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

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

methods 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.

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 has any UI (user interface).

The service runs in the background indefinitely even if application is destroyed.

Moreover, service can be bounded by a component to perform interactivity and inter process communication (IPC).

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.

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.

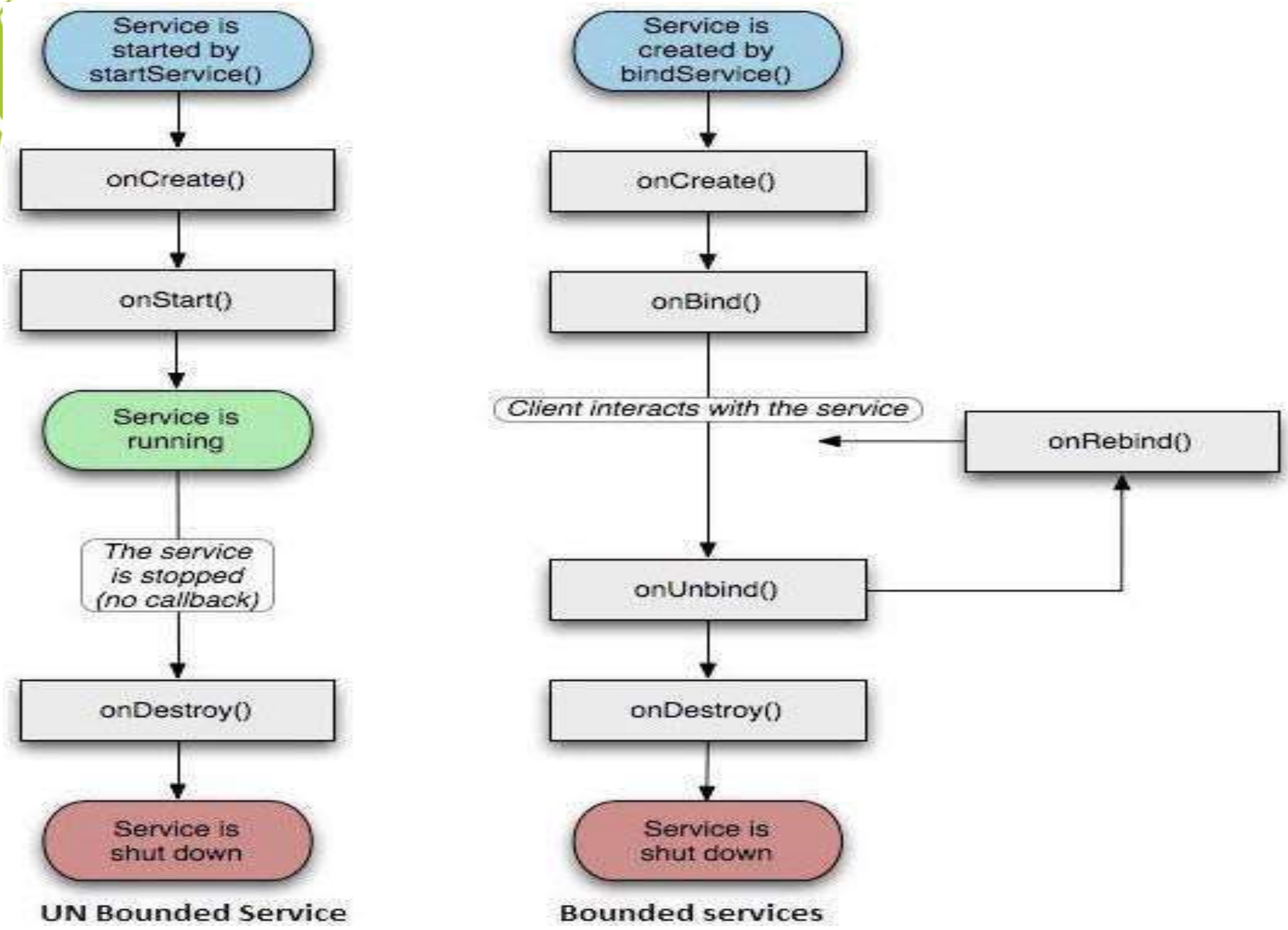
1) Started Service

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.



Service is started by startService()
onCreate()
onStart()
Service is running
The service is stopped (no callback)
onDestroy()
Service is shut down
UN Bounded Service
Service is created by bindService()
onCreate()
onBind()
Client interacts with the service
onRebind()
onUnbind()
onDestroy()
Service is shut down
Bounded services

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.

onStartCommand()

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 indefinetly.

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()

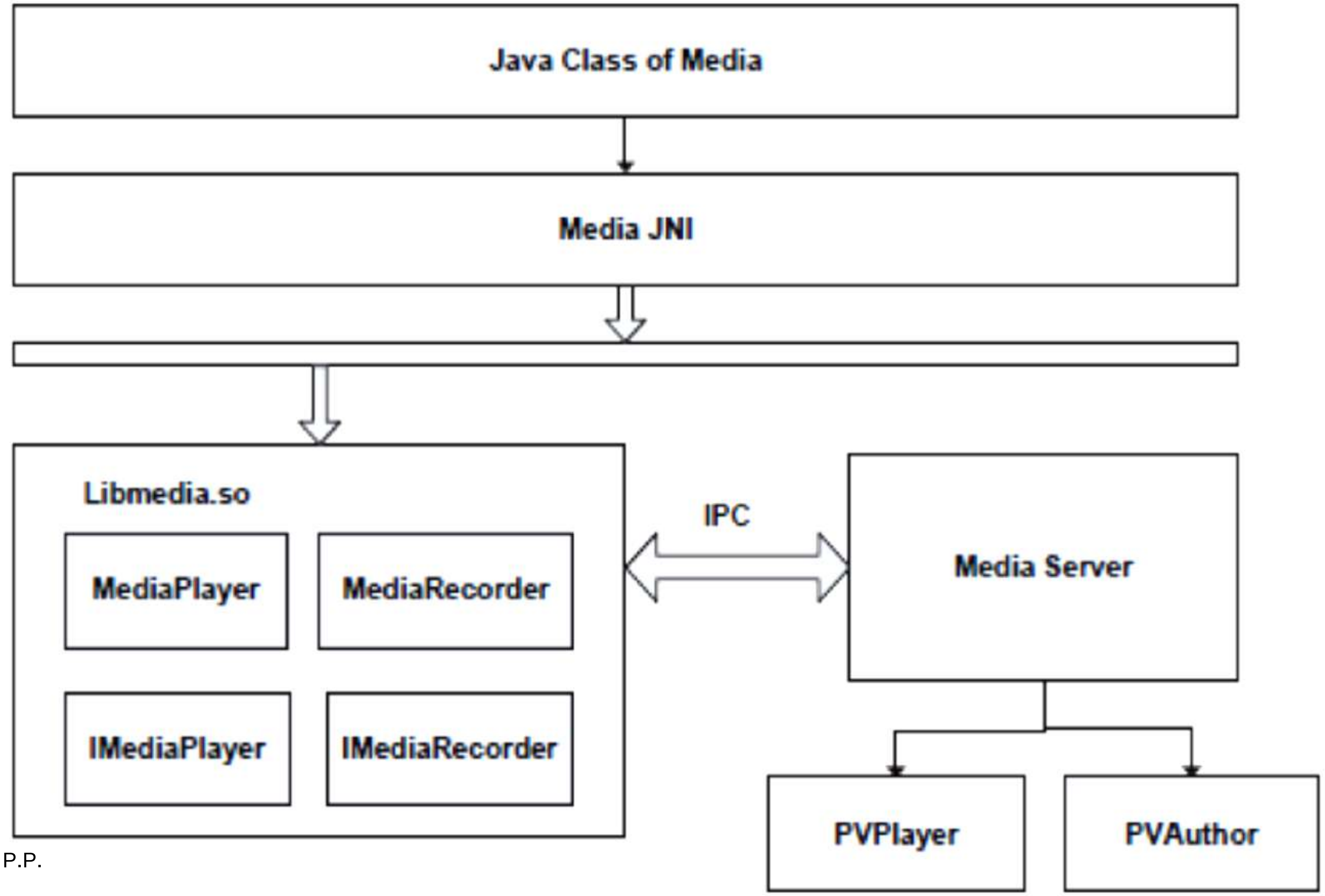
This is invoked when new clients are connected to the service. It is called after onUnbind

onDestroy()

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

Android Multimedia framework

The android multimedia system includes multimedia applications, multimedia framework, OpenCore engine and hardware abstract for audio/video input/output devices. And the goal of the android multimedia framework is to provide a consistent interface for Java services. The multimedia framework consists of several core dynamic libraries such as libmediajni, libmedia, libmediaplayservice and so on. A general multimedia framework architecture is shown in Figure



MediaPlayer class

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

Methods of MediaPlayer class

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

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

Android Video Player Example

By the help of **MediaController** and **VideoView** classes, we can play the video files in android.

MediaController class

The **android.widget.MediaController** is a view that contains media controls like play/pause, previous, next, fast-forward, rewind etc.

VideoView class

The **android.widget.VideoView** class provides methods to play and control the video player. The commonly used methods of VideoView class are as follows:

Method	**Description**
public void setMediaController(MediaController controller)	Sets the media controller to the video view.
public void setVideoURI (Uri uri)	Sets the URI of the video file.
public void start()	Starts the video view.
public void stopPlayback()	Stops the playback.
public void pause()	Pauses the playback.
public void suspend()	Suspends the playback.
public void resume()	Resumes the playback.
public void seekTo(int millis)	Seeks to specified time in miliseconds.

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**.

e.g.

```
TextToSpeech tts = new (context, OnInitListener);
```

In this listener, you have to specify the properties for TextToSpeech object , such as its language ,pitch e.t.c.

Language can be set by calling **setLanguage()** method. Its syntax is

```
tts.setLanguage(Locale.UK);
```

The list of some of the locales available are given below –

Sr.No.	Locale
1	US
2	CANADA_FRENCH
3	GERMANY
4	ITALY
5	JAPAN
6	CHINA

Once you have set the language, you can call **speak** method of the class to speak the text. Its syntax is given below −

```
tts.speak(text, TextToSpeech.QUEUE_FLUSH, null, null);
```

- The first parameter is the **text that would be spoken**.
- The second parameter defines that the **previous input** is flushed so as to begin a clean slate. Alternatively, we can use QUEUE_ADD to add the current text to the speech.
- The third parameter is the **bundle** that is passed.
- Fourth is the String: An **unique identifier for this request**.

some other methods available in the TextToSpeech class are :

Sr.No	Method & description
1	**addSpeech(String text, String filename)** This method adds a mapping between a string of text and a sound file.
2	**getLanguage()** This method returns a Locale instance describing the language.
3	**isSpeaking()** This method checks whether the TextToSpeech engine is busy speaking.
4	**setPitch(float pitch)** This method sets the speech pitch for the TextToSpeech engine.
5	**setSpeechRate(float speechRate)** This method sets the speech rate.
6	**shutdown()** This method releases the resources used by the TextToSpeech engine.
7	**stop()** This method stop the speak.

Async Task

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.

The basic methods used in an android AsyncTask class are:

- **doInBackground()** : contains the code which needs to be executed in background. In this method we can send results multiple times to the UI thread by publishProgress() method.
- **onPreExecute()** : contains the code which is executed before the background processing starts
- **onPostExecute()** : This method is called after doInBackground method completes processing. Result from doInBackground is passed to this method
- **onProgressUpdate()** : This method receives progress updates from doInBackground method, which is published via publishProgress() method, and this method can use this progress update to update the UI thread.

- **Audio Capture**

Android has a built in microphone through which you can capture audio and store it. Android provides MediaRecorder class to record audio or video. In order to use MediaRecorder class ,you will first create an instance of MediaRecorder class.

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

To start recording the audio call the two basic methods prepare() and start() are used.

Sr.No	Method & description
1	**setAudioSource()** This method specifies the source of audio to be recorded
2	**setVideoSource()** This method specifies the source of video to be recorded
3	**setOutputFormat()** This method specifies the audio format in which audio to be stored
4	**setAudioEncoder()** This method specifies the audio encoder to be used
5	**setOutputFile()** This method configures the path to the file into which the recorded audio is to be stored
6	**stop()** This method stops the recording process.
7	**release()** This method should be called when the recorder instance is needed.

Camera In Android

In Android, Camera is a hardware device that allows capturing pictures and videos in your applications.

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

1. By Camera Intent
2. By Camera API

1 Using Camera By Using Camera Application

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);
```

AndroidManifest.xml

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

2. Using Camera 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 request's from Camera Device create a CameraCaptureSession.

5.**CameraCaptureSession.CaptureCallback**: This is going to provide the Capture session results.

Bluetooth

Bluetooth is a communication network protocol, which allows devices to connect wirelessly to exchange the data with other Bluetooth devices.
By using android Bluetooth API’s in android applications, we can perform the following functionalities.

- Scan for the available Bluetooth devices within the range
- Connect to other devices through service discovery
- Transfer data to and from other devices
- Manage multiple connections

Android Set Bluetooth Permissions
To use Bluetooth features in our android applications, we must need to add multiple permissions, such
as **BLUETOOTH** and **ACCESS_COARSE_LOCATION** or **ACCESS_FINE_LOCATION** in our manifest file.

Permission	Description
BLUETOOTH	We need this permission to perform any Bluetooth communication, such as **requesting a connection, accepting a connection, and transferring data.**
LOCATION	We need this permission because the Bluetooth scans can be used to **gather the information about the location of user**.

In case, if we want to discover the available Bluetooth devices or manipulate Bluetooth settings from our app, we need to define **BLUETOOTH_ADMIN** permission.

Following is the example of defining the Bluetooth permissions in android manifest file.

```
<manifest ... >
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
...
</manifest>
```

BluetoothAdapter class

By the help of BluetoothAdapter class, we can perform **fundamental tasks** such as initiate device discovery, query a list of paired (bonded) devices, create a BluetoothServerSocket instance to listen for connection requests etc.

Constants of BluetoothAdapter class

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

static synchronized BluetoothAdapter getDefaultAdapter() returns the instance of BluetoothAdapter.

boolean enable() enables the bluetooth adapter if it is disabled.

boolean isEnabled() returns true if the bluetooth adapter is enabled.

boolean disable() disables the bluetooth adapter if it is enabled.

String getName() returns the name of the bluetooth adapter.

boolean setName(String name) changes the bluetooth name.

int getState() returns the current state of the local bluetooth adapter.

Set<BluetoothDevice> getBondedDevices() returns a set of paired (bonded) BluetoothDevice objects.

boolean startDiscovery() starts the discovery process.

Android Animation

Android Animation is used to give the UI a rich look and feel. Animations in android apps can be performed through XML or android code. In this android animation tutorial we'll go with XML codes for adding animations into our application.

Animation in android apps is the process of creating motion and shape change. The basic ways of animation are:

Fade In Animation

- Fade Out Animation
- Cross Fading Animation
- Blink Animation
- Zoom In Animation
- Zoom Out Animation
- Rotate Animation
- Move Animation
- Slide Up Animation
- Slide Down Animation
- Bounce Animation
- Sequential Animation
- Together Animation

android:interpolator : It is the rate of change in animation. We can define our own interpolators using the time as the constraint.

android:duration : Duration of the animation in which the animation should complete. This is generally the ideal duration to show the transition on the screen.The start and end of the animation are set using: android:fromTRANSFORMATION android:toTRANSFORMATION

TRANSFORMATION : is the transformation that we want to specify.

android:fillAfter : property specifies whether the view should be visible or hidden at the end of the animation. If it sets to false, the element changes to its previous state after the animation

android:startOffset : It is the waiting time before an animation starts. This property is mainly used to perform multiple animations in a sequential manner

android:repeatMode : This is useful when you want the animation to be repeat

android:repeatCount : This defines number of repetitions on animation. If we set this value to infinite then animation will repeat infinite times

Android SQLite Tutorial

SQLite is an **open-source relational database** i.e. used to perform database operations on android devices such as storing, manipulating or retrieving persistent data from the database.

It is a very lightweight database which comes with Android OS, that is embedded in android by default. So, there is no need to perform any database setup or administration task.

It combines a clean SQL interface with a very small memory footprint and decent speed.

Methods of SQLiteOpenHelper class

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

Method	Description
public abstract void onCreate(SQLiteDatabase db)	called only once when database is created for the first time.
public abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)	called when database needs to be upgraded.
public synchronized void close ()	closes the database object.
public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion)	called when database needs to be downgraded.

SQLiteDatabase class

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

Methods of SQLiteDatabase class

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

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

Opening and Closing Android SQLite Database Connection

Before performing any database operations like insert, update, delete records in a table, first open the database connection by calling **getWritableDatabase()** method as shown below:

```
public DBManager open() throws SQLException
{
        dbHelper = new DatabaseHelper(context);
        database = dbHelper.getWritableDatabase();
        return this;
}
```

Closing the database

To close a database connection the following method is invoked.

```
public void close()
{
        dbHelper.close();
}
```

Inserting new Record into Android SQLite database table

The following code snippet shows how to insert a new record in the android SQLite database.

```
public void insert(String name, String desc)
{
        ContentValues contentValue = new ContentValues();
        contentValue.put(DatabaseHelper.SUBJECT, name);
        contentValue.put(DatabaseHelper.DESC, desc);
        database.insert(DatabaseHelper.TABLE_NAME, null, contentValue);
}
```

Content Values creates an empty set of values using the given initial size.

Updating Record in Android SQLite database table

The following snippet shows how to update a single record.

```
public int update(long _id, String name, String desc)
{
        ContentValues contentValues = new ContentValues();
        contentValues.put(DatabaseHelper.SUBJECT, name);
        contentValues.put(DatabaseHelper.DESC, desc); int i =
        database.update(DatabaseHelper.TABLE_NAME, contentValues,
        DatabaseHelper._ID + " = " + _id, null);
        return i;
}
```

Android SQLite – Deleting a Record

We just need to pass the id of the record to be deleted as shown below.

```
public void delete(long _id)
{
database.delete(DatabaseHelper.TABLE_NAME, DatabaseHelper._ID + "="
+ _id, null);
}
```

Android SQLite Cursor

A Cursor represents the entire result set of the query. Once the query is fetched a call to **cursor.moveToFirst()** is made. Calling moveToFirst() does two things:

- It allows us to test whether the query returned an empty set (by testing the return value)
- It moves the cursor to the first result (when the set is not empty)

The following code is used to fetch all records.

```
public Cursor fetch()
{ String[] columns = new String[] { DatabaseHelper._ID,
DatabaseHelper.SUBJECT, DatabaseHelper.DESC };
Cursor cursor = database.query(DatabaseHelper.TABLE_NAME, columns,
null, null, null, null, null);
if (cursor != null)
{
cursor.moveToFirst();
}
return cursor;
}
```

Good Luck!