000579

S.N.J.B.'s

SHRI H. H. J. B. POLYTECHNIC, CHANDWAD

CLASS TEST I / II (201 - 201)

Course & Semester :		Roll No.:	Date : / /201
Name of Subject :			Marks obtained :
Sign. of Supervisor :		Sign. of Sub. Examiner :	
Q. No. 1	Q. No. 2	Q. No. 3	Total Marks

Supplied By : SNJB

Unit.5 Activity & Multimedia with data bases

* **Intent**

It is simple message object that is used to communicate between android components such as activities, content providers, broadcast receivers, & seevices. It is also used to transfer data between activities

It's are used generally for starting a new activity using startActivity()

Uses

for launching activity
to start new seevice
for broadcasting messages
to Display a list of contacts in listview.

Types

1) Implicit Intent
2) Explicit Intent

1) **Implicit Intent :-**

It is intent where instead of defining the exact components, you define the action that you want to perform for different activities.

① setData() This method is only to specifies a URL.
② setType() This method specifies a MIME Type.
③ setDataAndType() This method specifies both a URL & MIME type

It specifies an action that can invoke any app on the device to be able to perform an action. It is useful when your app cannot perform the action but other apps probably can and you'd like the user to pick which app to use

Syntax

```
Intent i = new Intent
i.setAction (Intent.ACTION_SEND);
```

The different methods used in Intent

1) Action_main
2) Pick
3) chooser
4) dial
5) call
6) send
7) sendTo
8) View

eg:-

```
Intent i=new Intent( Intent.ACTION_VIEW);
i.setData (Uri.parse("http// www.facebook
                .com");
StartActivity(i);
```

① ACTION_MAIN
adds an action to intent filter
<action android:name=" String">

② ACTION_PICK
it is using for picking the image from camera or gallery


```
Intent i= new Intent(Intent.ACTION_PICK);
i.setType(" image/* ");
StaetActivity(i, SELECT_PHOTO);
                ForResult
```

3) ACTION_CHOOSER
It is used for choosing the image from the gallery.

```
Intent i = new Intent (Intent.ACTION_CAMER
A);
    send. setData( uri);
    StartActivity( Intent. createChooser( sele
    ct ," Select an Image from camera"));
```

4) ACTION_DIAL
display the phone dialer with given no. filled in

```
String myno = " tel: 1234-56";
Intent i= new Intent ( Intent.ACTION_DIAL
,Uri. parse(myno));
StartActivity(i);
```

5) ACTION_CALL
placing & immediate phone call

```
String d = "tel: 6512808710";
Intent i = new Intent ( Intent.ACTION_CALL, Uri.
parse(d));
    StartActivity(i);
```

Permission needed

```
<uses-permission android: name="android.per
mission.CALL_PHONE">
```

6) ACTION_SEND

Sending text content from one activity to other

```
Intent i = new Intent();
    i.setAction(Intent.ACTION_SEND);
    i.putExtra(Intent.EXTRA_TEXT," This is
    my text to send");
    i.setType("text/plain");
    startActivity(i);
```

7) ACTION_SENDTO

prepaeing an sms. The text is supplied as an Extra element. The intent excepts such as values to be called 'sms body."

```
Intent i = new Intent(Intent.ACTION_SEN
OTO ,Uri.parse("smsto:8087123321"));
    i.putExtra("sms-body","we aee playing
    cricket");
    startActivity(i);
```

2) Explicit Intent-

It is Intent where you explicitly define the component that needs to be called by the android system

It can use to launch specific app compon ent, such as particular activity or service in youe app.

Syntax:-

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

```
i.putExtra("value1","This value for next Activity");
```

eg :-

```
{ EditText ename;
Button b,
String name;
        onCreate()
        {
ename=(EditText)findViewById(R.id.editText);
 b=(Button)findViewById(R.id.button);
 b.setOnClickListener(new View.OnClickListener()
  {
      public void onClick(View v)
      {
      Intent i = new Intent(MainActivity.this,
       SecondActivity.class);
       i.putExtra("Username", name);
       startActivity(i);
      }
  });
  }
}
```

```
SecondActivity
{ TextView t;
            onCreate()
            {
  t=(TextView)findViewById(R.id.textView);
   Intent i= getIntent();
  String name=(String) i.getSerializableExtra("
      Username");
      t.setText("Welcome"+ name);
            }
}
```

Intent filters

- the components which decide the behaviour of intent.
- We can declare an Intent filter for an activity in manifest file.
- It specify the type of intents that an activity, service, or broadcast receiver, can respond to. It declares the functionality of its parent component.
- The code of intent filter is used inside AndroidManifest.xml file for activity

syntax :-

```
<activity android:name=".MainActivity>
<intent-filter android:icon="@drawable
/icon"                    ①
 android:label="@string/label">    ②
<action android:name="android.intent
 .action.MAIN">
 <category android:name="android.
  intent.category.LAUNCHER"/>
</intent-filter>
</activity>
```

① displayed as icon for activity
② title that appear at top in Toolbar of activity

```
<action>
<category>   show behaviour of intent
<data>
```


S.N.J.B.'s
SHRI H. H. J. B. POLYTECHNIC, CHANDWAD
CLASS TEST I / II (201 - 201)

Course & Semester :	Roll No.:	Date : / /201
Name of Subject :		Marks obtained :
Sign. of Supervisor :	Sign. of Sub. Examiner :	

Q. No. 1	Q. No. 2	Q. No. 3	Total Marks

① **<action>**

```
<action android:name="String">
```

the name of intent action to be accepted if it must be literal string value of an action, not the class constant.

② **<category>**

```
<category android:name="String">
```

It shows behaviour of an Intent. There is a String which contains some additional information about the intent which will be handled by component
- CATEGORY-BROWSABLE
- CATEGORY-LAUNCHER

③ **<data>**

the type of data to be accepted & by using one or more attributes we can specify various aspects of data as two forms in which you can pass the data, using URI (uniform resource identifiers

```
<data android:scheme="string"
 android:host="string"
        port
        path
        pathPattern
        pathPrefix
        mineType
```

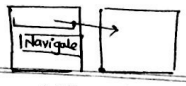
We can used multiple component to send mail

① ACTION_SEND
action which specifies that we are sending some data

② setType
we can use this property to set MIME type of data that we want to send
- message/rfc822
- text/plain
- image/jpg

③ putExtra()
to add extra info to intent
EXTRA_EMAIL
It's an array of email address
EXTRA_SUBJECT
The subjects of the email that we want to add
EXTRA_BODY
The body of email

Eg -
```
onClick(View v)
{
Intent i = Intent(Intent.ACTION_SEND);
i.setType("message/rfc822");
i.putExtra(Intent.EXTRA_EMAIL, new String[] {
eTo.getText().toString()});
i.putExtra(Intent.EXTRA_SUBJECT, eSubject.
getText().toString());
i.putExtra(Intent.EXTRA_BODY, emessage.
getText().toString());
setType("message/rfc");
startActivity(Intent.createChooser(i, "Cho
ose Mail App"));
}
```

Exp
(19) 1.)
```
public void onClick(View v)
{
String url = editText.getText().toString();
Intent i = new Intent.ACTION_VIEW, Uri.
parse(url));
startActivity(i);
}
```

3) factorial of given no

MainActivity.class
```
onclick()
{
int num = Integer.parseInt(editText.getTe
xt().toString());
Intent i = new Intent(MainActivity
.class, Result.class);
int fact = 1;
for(int i=1; i<num; i++)
{
fact = fact * i;
}
i.putExtra("Factorial", "factorial of "+num
+" is "+fact);
startActivity(i);
}
```

Result.class
```
TextView t = (TextView) findViewById(R.id.text
View);
Intent i = getIntent();
String fa = (String) intent.getSerializa
bleExtra("Factorial");
t.setText(fa);
```

2) Phone dialer

```
onClick()
{
Uri u = Uri.parse("tel:" + e.getText().
                    toString());
Intent i = new Intent(Intent.ACTION_
DIAL, u);
try
{  startActivity(i);
}
catch(SecurityException e)
{
Toast.makeText(this, e.message,
Toast.LENGTH_LONG).show();
}
}
```

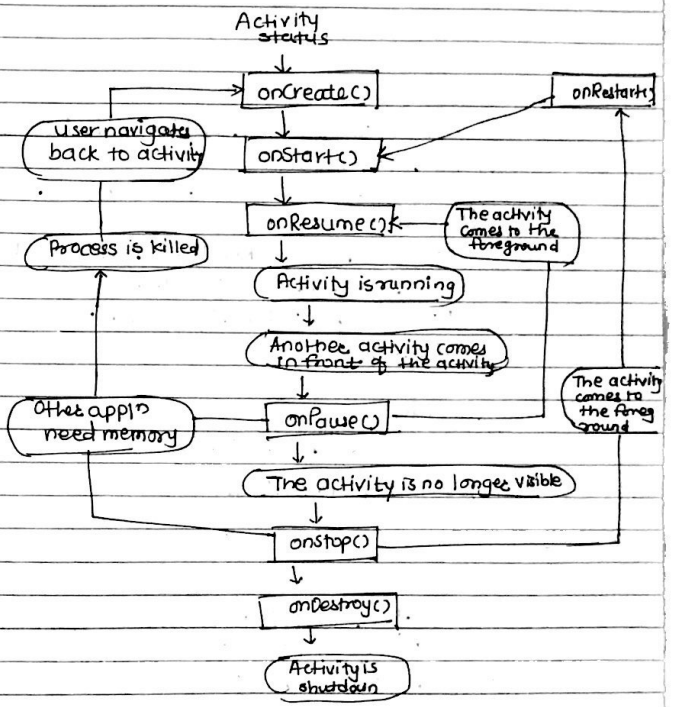

Activity LifeCycle

Activity is class that provides a window to draw the UI of your appln.

The first screen that is launched when you open an app is generally called Main Activity

. You can call any activity from any activity

Activity is one screen of the app's UI.



1) onCreate() - Activity Created
- called when first activity is created.
- only one time called during lifetime
It always followed by onStart()

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

2) onStart() - Activity Started
- called when the activity is becoming visible to user.
- can be called more than once during lifecycle
- followed by onResume() if the activity comes to the foreground
- followed by onStop() if it becomes hidden.

@Override
protected void onStart()
{ super.onStart();
}

3) OnRestart() - Activity Started
- Called after activity has been stopped, immediately before it is started again
- always followed by onStart()

@Override
protected void onRestart()
{ super.onRestart();
}

Course & Semester :	Roll No. :	Date :
Name of Subject :		Marks obtained :
Sign. of Supervisor :	Sign. of Sub. Examiner :	

Q. No. 1	Q. No. 2	Q. No. 3	Total Marks

000006

CO					Max. Marks	Total
Q.1	a/b	c/d			(8)	(20 Marks)
CO						Dated sign Of
Q.2	a/b	c/d			(12)	Course Teacher

4) onResume() Activity Resumed/Running
- called when activity will start interacting with user.
- activity has moved to top of the activity stack
- starts accepting user i/p
- Running state
- always followed by onPause()

@Override
protected void onResume
{ Super.onResume();
}

5) onPause() - Activity pause
- called when system is about to resume a previous activity
- The activity is partly visible but user is leaving the activity.

followed by onResume() if the activity returns back to front
onstop() if becomes invisible to the user.

@Override
protected void onPause()
{ super.onPause();
}

6) onStop() Activity stopped
- called when activity is no longer visible to the user
- new activity is being started, an existing one is brought in front of this one, or this one is being destroyed.
followed by onRestart() if this activity is coming back to interact with the user,
onDestroy() if this activity is going away

@override
protected void onStop()
{ super.onStop();
}

7) onDestroy() - Activity Destroyed
- final call before activity is destroyed
- user navigates back to previous activity or configuration changes
- activity is finishing or system is destroying it to save space
- call isFinishing() method to check
- system may destroy activity without calling this, so use onPause() or onStop() to save data or state

@Override
Protected void onDestroy()
{ super.onDestroy();
}

Broadcast Lifecycle

Broadcast Receivers simply respond to broadcast messages from other appln or from system itself. This messages are sometime called events or intents.

There are following two important steps to make BroadcastReceiver works for the system broadcasted intents.

① Creating the Broadcast Receiver
② Registering Broadcast Receiver

There is one additional steps in case you are going to implement your custom intents then you will have to create & broadcast those intents.

① Creating the Broadcast Receiver

It is implemented as subclass of BroadcastReceiver class & override the onReceive() method where each message is received as Intent object parameter

eg:- public class MyReceiver extends BroadcastReceiv
or { @Override
public void onReceive (Context context, Intent intent)
{
Toast.makeText(context, "IntentDetected", Toast.

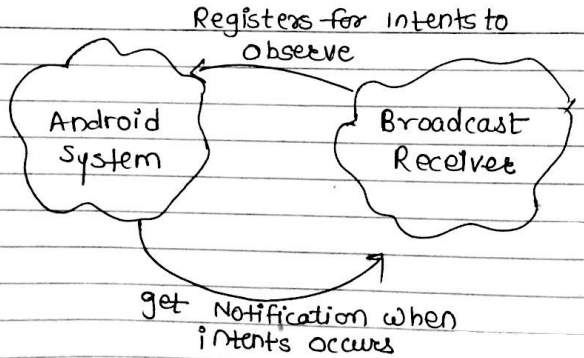
```
        LENGTH_LONG).show();
    }
}
```

2) Registering Broadcast Receiver

An application listens for specific broadcast intents by registering broadcast receiver in AndroidManifest.xml file

eg:-

```
<application>
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/theme">
    <receiver android:name="MyReceiver">
        <intent-filter>
        <action android:name="android.
         intent.action.BOOT_COMPLETED">
        </action>
        </intent-filter>
    </receiver>
</application
```



System Broadcast Intent

There are several system generated events defined as final static fields in Intent class

1) android.intent.action.BATTERY_CHANGED
   sticky broadcast containing the charging state, level & other info about battery

2) android.intent.action.BATTERY_LOW
   Indicates low battery condn on the device

3) android.intent.action.BATTERY_OKAY
   Indicates the battery is now okay after being low

4) andeoid.intent.action.BOOT_COMPLETED
   this is broadcast once, after the system has finised booting

5) android.intent.action.BUG_REPORT
   show activity for reporting a bug

6) android.intent.action.CALL
   perform a call to someone specified by the data

7) android.intent.action.CALL_BUTTON
   The user pressed the call button to go to the dialer or other appropriate UI for placing a call

8) android.intent.action.DATE_CHANGED
   The data has changed

9) android.intent.action.REBOOT
   have the device reboot

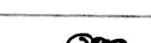
Broadcasting Custom Intents

Custom broadcast intents are broadcast intents that your appln sends out. Use a custom broadcast intent when you want your app to take an action without launching an activity

To create custom broadcast intent, create custom intent action

To deliver custom broadcast to other apps, pass the intent to sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast()

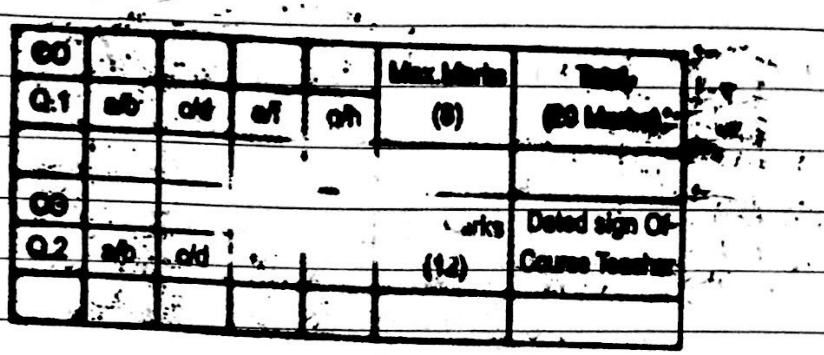
```
public void broadcastIntent (View v)
{
    Intent i = new Intent();
    i.setAction("com.example.CUSTOM_INTENT");
    sendBroadcast(i);
}
```



```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <receiver android:name="MyReceiver">
        <intent_filter>
            <action android:name="com.
            example.CUSTOM_INTENT">
            </action>
        </intent_filter>
    </receiver>
</application
```

Course & Semester :		Roll No.:	Date : / /201
Name of Subject :			Marks obtained :
Sign. of Supervisor :		Sign. of Sub. Examiner :	
Q. No. 1	Q. No. 2	Q. No. 3	Total Marks

000029



Fragments

A fragment is reusable class implementing a portion of an activity.

A fragment typically defines a part of a user interface

Fragments must be embedded in activities; they cannot run independently of activities

"A fragment represents a behaviour or a portion of user interface in an activity"

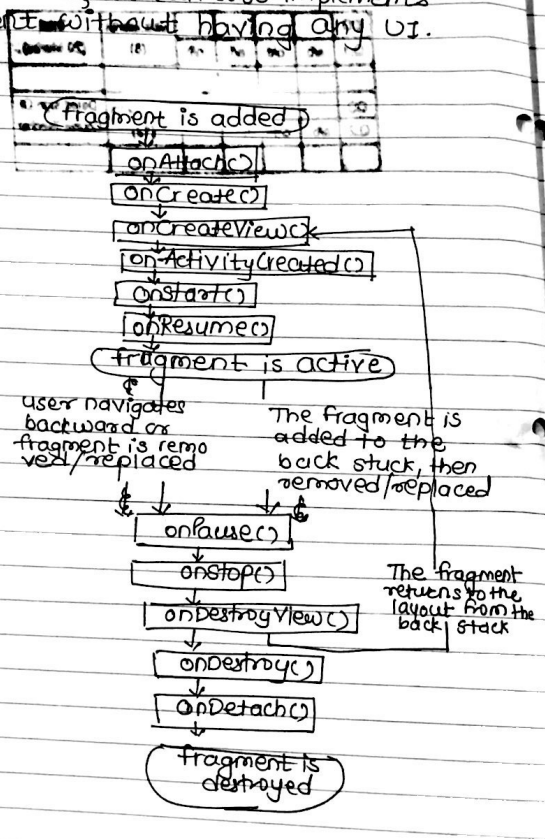
We can add build multiple fragment in a single activity & reuse same fragment in multiple activities.

The fragment has its own lifecycle call_backs & fragments can accepts its own i/p events.

We can add or remove from activity while activity is running. If pause an activity, all the fragments related to activity will also be stopped.

We can insert fragment into activity layout by using <fragment> element & by dividing the layout of an activity into fragments, we can also implements a fragment without having any UI.

Lifecycle



1) onAttach():- It is called when the fragments has been associated with activity
2) onCreate():- It's used to initialize the fragment
3) onCreateView():- to create view hierarchy associated with fragment
4) onActivityCreated():- It is called when the fragment activity has been created & the fragment view hierarchy instantiated
5) onStart():- It is used to make the fragment visible
6) onResume():- It is used to make the fragment visible in activity.
7) onPause():- It is called when fragment is no longer visible & it indicates that the user is leaving the fragment.
8) onStop():- It is called to stop the fragment
9) onDestroyView():- The view hierarchy which associated with the fragment is being removed after executing this method
10) onDestroy():- It is called to perform a final clean up of fragments state.
11) onDetach():- It is called immediately after the fragment disassociated from the activity

Create a fragment class

extends Fragment class, then override key lifecycle methods to insert your app logic.

When creating fragment must use the onCreateView() callback to define layout.

eg:-

```
import android.os.Bundle;
        .support.v4.app.Fragment;
        .view.ViewGroup;
        .view.LayoutInflater;
public class ArticleFragment extends
 Fragment
{
   @Override
public View onCreateView(LayoutInflater
inflater, ViewGroup container, Bundle saveIn
stanceState)
      {
      return inflater.inflate(R.layout.
       article_view, container, false);
      }
}
```

res/layout-large/news_articles.xml

```
<LinearLayout
    android:orientation="horizontal"
          :layout-width="fill_parent"
          -height="
<fragment
android:name=".com.example.android
    .fragments.ArticleFragment"
android.id="@+id/article_fragment"
android:layout_weight="2"
          - width="0dp"
          - height="match_parent"/>
</LinearLayout>
```

Then apply the layout to your activity:-

```
public class MainActivity extends FragmentA
ctivity
{ @Override
public void onCreate(Bundle savedInstanc
        eState)
    {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.news_arti
      cles);
    }
}
```

Note :-
If you're using v7 appcompat library your activity should extends AppCompact Activity which is subclass of FragmentActivity

Types

1) Single Frame fragment
It is design for small screen devices such as hand held devices (mobiles). & it should be above android 3.0 version.

2) List fragments
Static library support version of the

framework's ListFragment.

3) Fragment Transition

used to move one to another fragment

S.N.J.B.'s

SHRI H. H. J. B. POLYTECHNIC, CHANDWAD

CLASS TEST I / II (201 - 201)

Course & Semester :	Roll No.:	Date : / /201
Name of Subject :		Marks obtained :
Sign. of Supervisor :	Sign. of Sub. Examiner :	

Q. No. 1	Q. No. 2	Q. No. 3	Total Marks

000028

CO					Max.Marks (8)	Total (20 Marks)
Q.1	a/b	c/d	e*			
						Dated sign Of Course Teacher
CO						
Q.2	a/b					

Content Provider

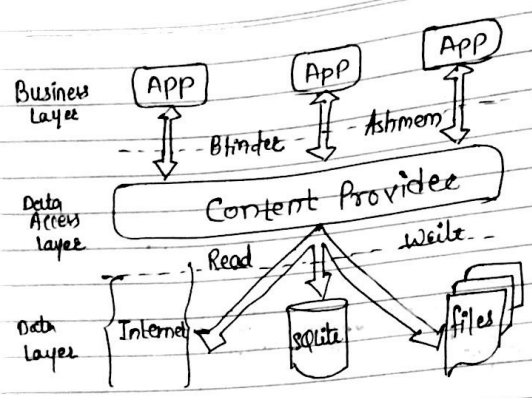
It will act as central responsitory to store the applications data in one place and make that data available for different applications to access whenever its. required.

We can configuce content providers to allow other applications ~~to~~ seaurely access & modify owe app data based on ouc requisement

The Content provider is part of andeoid appln. & it will act as more like relational database to store the app data.

We can perform multiple operations like insert, update, delete & edit on the data stored in content provider using insert(), update(), delete(), & query() methods. In most cases this data is stored in SQlite db.

We can manage data such as audio, video images & personal contact info.



ContentProvider

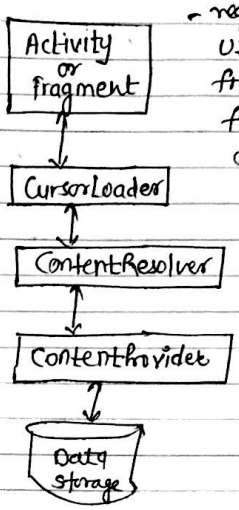
Access Data from Content Provider

We need to use ContentResolver object in our appln to communicate with the provider as client.

The ContentResolver object will communicate with the provider object (ContentProvider) which is implemented by instance of class

In android, to send request from UI to ContentResolver we have another object called CursorLoader which is used to run the query asynchronously in background. In android appln the UI components such as Activity or fragments will call a Cursorloader to query & get required data from ContentProvider using ContentResolver

The ContentProvider object will receive a data request from client, performs the requested actions (create, update, delete, retrieve) & return the result.



- requesting operation from UI using activity or fragment to get data from ContentProvider object

A content provider is implemented as a subclass of ContentProvider class must implement a standard set of API's that enable other applications to perform transactions

```
public class MyApplication extends
ContentProvider
{

}
```

eg:- content://contacts.info/users
content is used to represent URI is content URI
string is the table path
string is the name of provider's authority

Content URIs

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

syntax content://authority/path

content://
The string content:// is always present in URI & it is used to represent the given URI is content URI

authority
It represent the name of content provider,
eg - phone, contacts etc

path
It represents the table's path

or content://<authority>/<data_type>/<id>

datatype
specifies the type of data that this particular providers provides

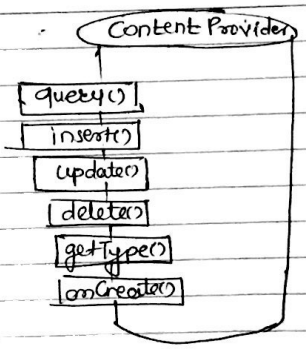
id
specific record requested.

The ContentResolver object use the URI's authority to find the appropriate provider & send the query object to the correct provider. After ContentProvider uses the path of content URI to choose the right table to access.

Creating a Content Provider

- We need to create a content provider class that extends the ContentProvider base class
- We need to define our content provider URI to access the content.
- The ContentProvider class define six abstract method
  - insert()
  - update()
  - delete()
  - query()
  - getType()

  which we need to implement all these methods as part of our subclass
- We need to register our content provider in AndroidManifest.xml using <provider> tag



query() - It is receives a request from the client. By using arguments it will get data from requested table & return the data as a cursor object

Insert () :— This method will insert a new row into our content provider & it will return the content URI for newly inserted row.

Update () :— This method will update an existing rows in our content provider & it return the no. of rows updated.

delete () :— This method will delete the rows in our content provider & it return the no. of rows deleted.

get Type () :— This method will return the MIME type of data to give content URI

On create () :— This method will intialize our provider. The android system will call this method immediately after it creates our provider

Name of Subject :		Marks obtained :	
Sign. of Supervisor :		Sign. of Sub. Examiner :	
Q. No. 1	Q. No. 2	Q. No. 3	Total Marks

Supplied By : SNJB

Services

- It is component that is used to perform operations on the background such as playing music, handle n/w transactions, interacting content provider etc
- It doesn't has any UI.
- The service runs in the background indefinitely even if appln is destroyed.
- The service can be bounded by component to perform interactivity & inter process comm.
- The android.app.Service is subclass of contextWrapper class

Note Android service is not a thread or seperate process.

Lifecycle of Android service

There are two form of a service. The lifecycle of service can follow two diff paths : Started or bound

1) Started (Unbound)

2) bound

1) Started Service

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

2) Bound service

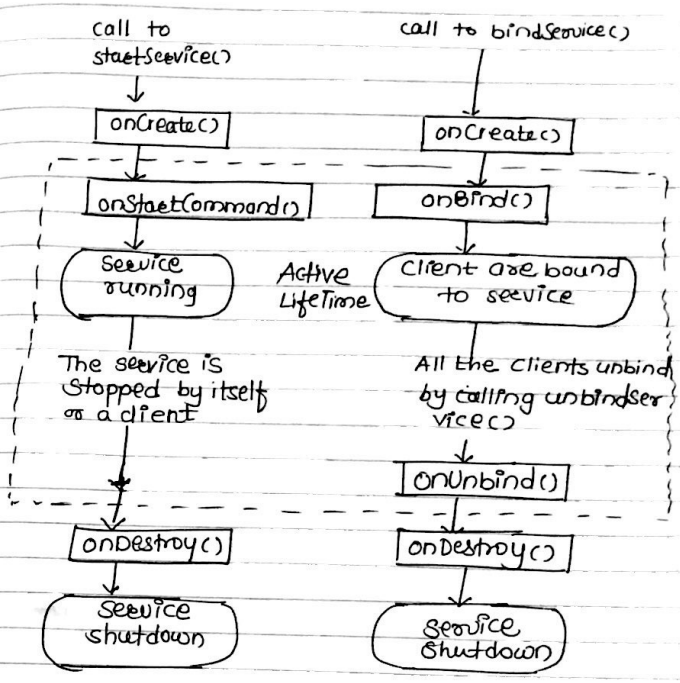
A service is bound when another component (eg. client) calls bindService() method. The client can unbind the service by calling the unbindService() method.

The service can't be stopped until all clients unbind the service

Declared the service in Manifest:-

```
<manifest...>
  ...
  <appln....>
    <service
    android:name="ExampleService"
     android:exported="false"/>
    ...
  </appln>
</manifest>
```

Lifecycle



1) onStartCommand()

The system calls this method when another component request that service be started, by calling startService(), If you implementing this method, it is your responsibility to stop the service when its work is done, by calling stopSelf() or stopService()

2) onBind()

The system calls this method when another component want to bind with the service by bindService() If you implement this method, you must provide an interface that client use to communicate with the service, by returning an IBinder object. You must always implement this method. but if you don't want to allow binding, then you should return null.

3) onUnbind()

The system calls this method when all the clients have disconnected from a particular interface published by the service

4) onCreate()

The system calls this method when the service is first created using onStartCommand() or onBind(). This call is required to perform onetime set up

5) onDestroy()

The system calls this method when the service is no longer used & is being destroyed. Your service should implement this to clean up any resources such as thread, registered listeners, receivers etc

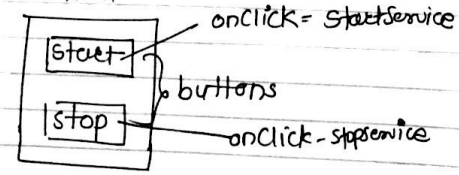
The following skeleton service demonstrates each of the life cycle methods

```
import android.app.Service;
       .OS.IBinder;
       .content.Intent;
       .Os.Bundle;

public class Demo extends Service
{
  int mStartMode; // indicates how to behave if the service is kill.
  IBinder mBinder; // interface for clients that bind
  boolean mAllowRebind; // indicates whether onRebind should be used
  public void onCreate() // called when service is being created
  {
  }
  // The service is starting due to call to startService()
  public int onStartCommand(Intent intent, int flags, int startId
  {
    return mStartMode;
  }
  // A client is binding to the service with bindService()
  public IBinder onBind(Intent intent)
  {
    return mBinder;
  }
  // call when all clients have unbound with unbindService()
  public boolean onUnbind(Intent intent)
  {
    return mAllowRebind;
  }
  // call when service is no longer used & is being destroyed
  public void onDestroy()
  {
  }
}
```

MainActivity

```
impoet android.content.*;
                .os.*;
                .app.*;
                .util.*;
                .view.*;
                .support.v7.app.AppCompatActivity;

public class MainActivity extends Activity
{
    public void onCreate(Bundle savedInstancestate){
    {
      super.onCreate(savedInstanxcestate);
      setContentView(R.layout.activity_main);
    }
    public void startSeevice (View view)
    {
      startService (new Intent (getBaseContext(), Myseevice.class);
    }
    public void stopSeevice (View view)
    {
      stopSeevice(new Intent(getBaseContext(), myseevice.class);
    }
}
```

activity-main.xml



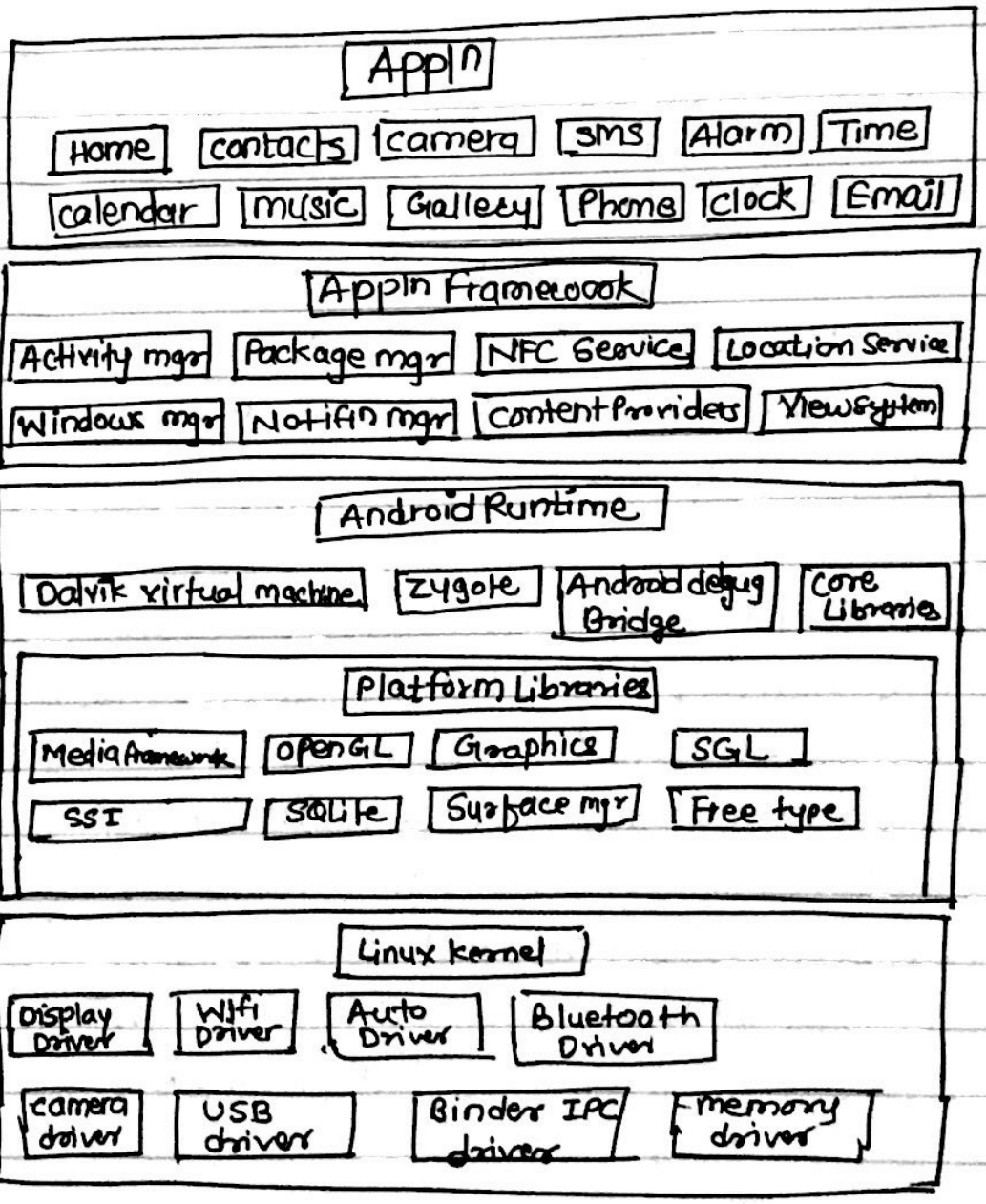
MyService.java

```
impoet android.app.Service,
                .content.Intent;
                .os.IBinder;
                .support.annotation.Nullable;
                .widget.Toast;
public class MyService extends Service
{   @Nullable
    @Oveeeide
    public IBinder onBind (Intent intent)
    {
      retuen null;
    }
    @Oveeeide
    public int onStaetCommand(Intent intent, int flags, int StaetId)
    {
      Toast.makeText (this," service staeted", Toast.LENGTH_LONG).show();
      retuen START_STICKY
    }
    @Oveeeide
    public void onDestroy()
    {
        super.onDestroy();
      Toast.makeText(this, "seevice Destroyed", Toast.LENGTH_LONG).show();
    }
}
```

AndroidManifest.xml.

```
    </Activity>
    <service android:name=".MyService"/>
  </application>
</manifest>
```

Android System Architecture



Android Architecture

It is s/w stack of components to support a mobile device needs.
Android s/w stack contains a Linux kernel, collection of C/C++ libraries which is exposed through an appln framework services, runtime & appln

following are main component of android architecture. 1) Appln
2) Android framework.
3) Android Runtime.
4) Platform Libraries.
5) Linux kernel.

In these component Linux kernel is the main component to provides its o.s fn to mobile DVM which is responsible for running a mobile appln.

1) Appln

The top layer of android architecture is appln. The native & 3rd party appln like contacts, email, music, gallery, clock, games etc. whatever we will built those will be installed on this layer only.

It runs within the Android run time using the classes & services made available from the appln framework.

2) Appln framework

The application framework provides the classes used to create an android applications. It also provides a generic abstraction for hardware access & manages the user interface and application resources.

It basically provides the services through which we can create the particular class & make that class helpful for the Applications creation.
- The application framework includes services like telephony service, location services. notification manager, NFC service, view system, etc. which we can use for application development as per our requirements.

3) Android runtime :—

Android runtime environment is an important part of Android rather than an internal part & it contains a components like core libraries & the Dalvik virtual machine. The Android run time is the engine that powers our applications along with the libraries & it forms the basis for the appln framework.

Dalvik virtual Machine (DVM) is a register based virtual machine like Java virtual Machine (JVM). It is specially designed & optimized for android to ensure that a device can run multiple instances efficiently. It relies on the linux kernel for threading & low-level memory management.

The core libraries in android runtime will enable us to implement an android appln using standard JAVA programming language

4) Platform libraries :—

The platform libraries includes various c/c++ core libraries & Java based libraries such as SSL, libc, Graphics, SQLite, Webkit, media, Surface

Manager, OpenGL, etc to provide a support for android development

- following are the summary details of some core android libraries available for android development
  - Media library for playing & recording an audio & video formats.
  - The surface manager library to provide a display management
  - SGL & openGL Graphics libraries for 2D & 3D graphics.
  - SQLite is for database support and FreeType for font support
  - Webkit for web-browser support & SSL for Internet security

5) Linux kernel:—

Linux kernel is a bottom layer and heart of the android architecture. It manage all the drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc. which are mainly required for the android device during the runtime

The Linux kernel will provides an abstraction layer between the device hardware & the remainder of the stack. It is responsible for memory management, power management, device management, resources access, etc

6) Multimedia framework:—

Android includes, a media playback engine at the native level that has built in software based codecs for popular media formats.

- Audio & video playback features include integration with openMAX codecs, session management time synchronized rendering, transport control and DRM.
- Stagefright also supports integration with custom hardware codecs provided by you. To set a hardware path to encode & decode media, you must implement a hardware-based codecs as an openMAX IL (integration layer) component