PART-B ESSAY QUESTIONS WITH SOLUTIONS

3.1 INTENTS AND BROADCASTS

3.1.1 Intent

Q11. Write in brief about intents and its uses. Model Paper-II, Q6(a)

Answer :

Intent

Intent can be defined as a messaging object that allows user to request an action from the application component. It can be used as a message-passing mechanism that can be implemented within the application and between applications.

Uses of Intents

The uses of Intents are as follows,

(i) In Android, intents provide interaction among various components within the application and between the applications. This helps a device with a set of independent components to convert into an interconnected system.

(ii) Intents can be used to launch/start new activities or services. These activities can be started either explicitly or implicitly.

(iii) Intents are used to broadcast events or messages to all the components of the system. Applications need to register Broadcast receivers in order to listen and respond to the broadcast intents. These broadcast intents are used to inform about the system events to user such as Internet connection, battery levels etc. In android, applications such as call dialer and messages uses components which listen to specific broadcast intents like incoming call (or) message received and response given to events accordingly.

3.1.2 Using Intents to Launch Activity – Explicitly Starting New Activity, Implicit Intents, Passing Data to Intents, Getting Results from Activities, Native Actions

Q12. Write in brief about launching activities using intents. Model Paper-III, Q6(a)

Answer :

The intents are used to launch activities that allows creation of a workflow for multiple screens. The activity can be created by using startActivity method and passing a specified intent to the method which is shown below,

```
startActivity(User_Intent)
```

Using intents, the activities can be launched (or) started either explicitly or implicitly.

1. **Explicitly Starting a New Activity**

   For answer refer Unit-III, Q13(i).

2. **Implicitly Starting a New Activity**

   For answer refer Unit-III, Q13(ii).

Q13. Write short notes on the following,

(i) Explicitly starting new activity

(ii) Implicit intent.

Answer :

(i) Explicitly Starting New Activity

Generally, applications contain multiple interrelated screens i.e., activities in real which need to be included in application manifest. In order to transition in between them, it is necessary to specify the activity to be opened.

A new Activity class can be started explicitly by creating a new intent to determine current activity's context and new activity class as parameters. Now pass the new intent to startActivity method as shown below,

```
Intent in = new Intent (MyActivity.this, MyNewActivity.class);
startActivity(in);
```

The new activity is created, started and resumed as soon as startActivity is called by moving to top of activity stack. The new activity can be closed or removed from stack by calling finish method or by pressing hardware back button. The startActivity can be used to navigate other activities as well. This process will be repeated each time when startActivity and finish are called.

3.6

(ii) Implicit Intents

The implicit intents allow the anonymous application components to work on the action requests. It means that the system is asked to start an activity in order to perform an action without the knowledge of application or activity to be started. For instance, the users can be allowed to make calls from the application by implementing new dialer or by using an implicit intent which requests an action that is to be performed on phone number. The below code snippet illustrates the launching of activity implicitly.

```
if (x && y)
{
    Intent in = new Intent (Intent.ACTION_DIAL, Uri.parse ("tel : 333 – 1628"));
    startActivity(in);
}
```

This intent gets resolved and a new activity is initiated by Android. This activity facilitates a dial action on telephone number i.e., typically phone dialer. While building a new implicit intent, an action is specified where URI of data can be passed on action to be performed. The extras are added to intent for sending additional data to target activity. It is a mechanism that attaches primitive values to an intent. A new name/value pair (NVP) can be assigned by using overloaded putExtra method on any intent, get[type]Extra method can be used for retrieving NVPs in started activity. This method can also be used to retrieve extras stored in the form of Bundle object.

The Android will use the implicit Intent to resolve the activity into Activity class that is preferable in performing the required action on type of data specified in intent. So, it is possible to create the projects that can utilize the functionality of other applications without the knowledge of the application from which the functionality is taken. In some cases, where the multiple activities perform a particular action, the user is provided with a choice. The intent resolution is specified through analysis of registered Broadcast Receivers.

Q14. Discuss in brief about the following,

(i) Passing data to intents

(ii) Getting results from activities

(iii) Native actions.

Model Paper-I, Q6(a)

Answer :

(i) Passing Data to Intents

It is possible to pass data in intent by adding it to intent object using putExtra( ) method. It is passed in the form of key value pair. Here, the value can be of different types like int, float, long, string etc. These are the simple data types that can be passed easily. The only thing which is needed to be done is to pass datatypes to intent with unique key and forward to some other activity. However, it is difficult in case of passing custom objects between the activities. In such case, serialization is used. The data can be sent as shown below,

```
Intent in = new Intent (context, TargetActivity_class);
in.putExtra(key, value)
startActivity(in);
```

Here, a single value is passed in the same way with multiple values attached by using putExtra( ) method. If a string object is sent then it can be fetched as shown below,

```
Intent in = getIntent( );
String s = in.getStringExtra(key);
```

There are some other types of data that can be fetched by using getIntExtra( ), getFloatExtra( ) etc.

(ii) Getting Results from Activities

An activity that is started by using startActivity method does not provide feedback since it is independent of parent. An activity can be started as sub-activity to return results providing feedback to its parent. The sub activities are similar to activities that are opened in a different way. Once the subActivity completes, it generates onActivityResult event handler in calling activity.

Launching Sub-activities

The sub-activities are mainly used in the situations where one activity requires data input from another activity and returns the results.

The sub-activities are launched for returning the results to the parent. It can be done by using StartActivityForResult method. It accepts two parameters such as explicit/implicit intent and request code.

The below code snippet illustrates the launching of sub-activity explicity.

```
private static final int Sub_S1 = 1;        //request code
private void start_Sub1( )
{
    Intent in = new Intent(this, My_Activity class);
    StartActivityForResult(in, Sub_S1);
    The below code snippet illustrates the launching of subActivity implicity,
    private static final int Sub_S2 = 2;        //request code
    private void start_Sub2( )
    {
        Uri uri = Uri.parse("content://Gallery/Photos");
        Intent in = new.Intent(Intent.ACTION_PICK, uri);
        StartActivityForResult(in, Sub_S2);
    }
}
```

When a subActivity is ready to return the result, the method setResult is called before finish method. This method accepts two parameters, that is the result code and result data. The result code can be either RESULT_OK or activity RESULT_CANCELLED. It is possible to define the user response codes when the OK and cancelled constants do not define return results accurately. Consider an example of returning a result from a subactivity as shown below,

```
Button b = (Button) findViewById(R.id.b1);
b.setOnClickListener( new View.OnClickListener( )
{
public void onClick(View v)
{
long s_id = listview.getSelectedItemId( );
Uri  u = Uri.parse("content://h/"+s_id);
Intent res = new Intent(Intent.ACTION_PICK, u);
setResult(RESULT_OK, res);
finish( );
}
});
Button b2 = (Button) findViewById(R.id.b3);
b2.setOnClickListener(new View.OnClickListener( )
{
public void onClick(View v)
{
setResult(RESULT_CANCELLED);
finish( );
}
});
```

(iii) **Native Actions**

The Native Android applications make use of intents from launching the activities and sub-activities. Some of the native actions which are available as static string constants in Intent class are as follows,

(a) **ACTION_ANSWER**

This action is used to open an activity to manage the incoming calls.

(b) **ACTION_CALL**

This action is used to provide a phone dialer and then initiate a call by using number that is specified in Intents data URI.

(c) **ACTION_DIAL**

This action is used to obtain a dialer application along with specified number to dial from Intent's data URI.

(d) **ACTION_DELETE**

This action is used to initiate an activity that allows deletion of specified data at Intents data URI.

(e) **ACTION_EDIT**

This action is used to requests an activity which can edit data at Intent's data URI.

(f) **ACTION_BUG_REPORT**

This action is used to show activity which can report the bug.

(g) **ACTION_CALL_BUTTON**

This action is triggered when hardware "call button" is pressed.

(h) **ACTION_INSERT**

This action is used to open an activity that allows insertion of new items into specified cursor in Intents data URI.

(i) **ACTION_SEND**

This action is used to launch an activity that forwards specified data in Intent.

(j) **ACTION_SEARCH**

This action is used to launch a specific search activity.

3.1.3 Using Intent to Dial a Number or to Send SMS

Q15. How intent is used to dial a number?

Model Paper-I, Q6(b)

Answer :

activity_main.xml

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity" >
  <Button
    android:id="@+id/b1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="120dp"
    android:text="Call" />
```

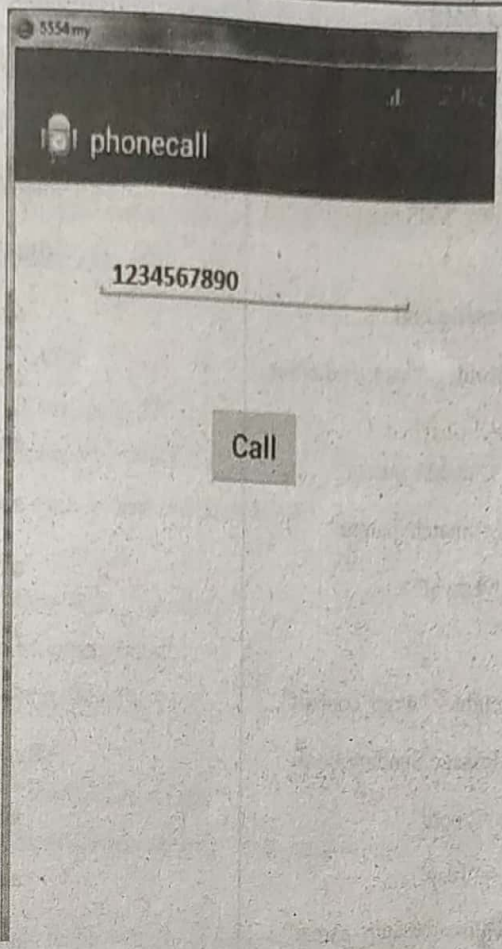
```
<EditText
    android:id="@+id/e1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="35dp"
    android:ems="10" />
</RelativeLayout>
```

Android-Manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:androclass="http://schemas.android.com/apk/res/android"
  package="com.example.phonecall"
  android:versionCode="1"
  android:versionName="1.0" >
  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="16" />
  <uses-permission android:name="android.permission.CALL_PHONE" />
  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name="com.example.phonecall.MainActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

MainActivity.java

```
package com.example.phonecall;
import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
public class MainActivity extends Activity {
  EditText e1;
  Button b1;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //Getting the edittext and button instance
    e1=(EditText)findViewById(R.id.e1);
    b1=(Button)findViewById(R.id.b1);
    //Performing action on button click
    b1.setOnClickListener(new OnClickListener(){
     @Override
      public void onClick(View arg0) {
        String number=e1.getText().toString();
        Intent callIntent = new Intent(Intent.ACTION_CALL);
        callIntent.setData(Uri.parse("tel:"+number));
        startActivity(callIntent);
      }
    });
  }
  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.activity_main, menu);
    return true;
  }
}
```



Q16. How Intent is used to send SMS?

Answer :

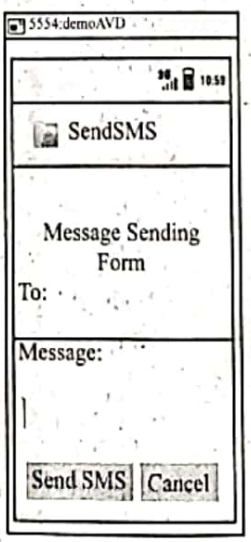
Model Paper-II, Q7

One of the popular communication modes of sending and receiving is the SMS messages. A built-in class called SMS manager in Android provides this facility. SMS messaging can be tested on Android emulator.

Code Written into activity_send_smsapp.xml

```
<LinearLayout xmlns : android = "http://schemas.
android.com/apk/res/android"
    android : layout_width = "match_parent"
    android : layout_height = "match_parent"
    android : orientation = "vertical">
    <TextView
        android : layout_height = "wrap_content"
        android : text = "Message Sending Form"
        android : textStyle = "bold"
        android : textSize = "18sp"
        android : layout_width = "match_parent"
        android : gravity = "center_horizontal"/>
    <TextView
        android : layout_width = "wrap_content"
        android : layout_height = "wrap_content"
        android : text = "To:" />
    <EditText
        android : id = "@ _ id/recvr_no"
        android : layout_height = "wrap_content"
        android : layout_width = "match_parent" />
    <TextView
        android : layout_width = "wrap_content"
        android : layout_height = "wrap_content"
        android : text = "Message :" />
    <EditText
        android : id = "@ + id/t_msg"
        android : layout_width = "match_parent"
        android : layout_height = "150dp" />
    <RelativeLayout
        android : layout_width = "match_parent"
        android : layout_height = "match_parent"
        android : orientation = "horizontal">
        <Button
            android : id = "@ + id/send_button"
            android : text = "Send SMS"
            android : layout_width = "wrap_content"
            android : layout_height = "wrap_content"
            android : layout_marginLeft = "40dip"
            android : paddingLeft = "30dip"
            android : paddingRight = "30dip" />
        <Button
            android : id = "@ + id/cancel_button"
            android : text = "Cancel"
            android : layout_width = "wrap_content"
            android : layout_height = "wrap_content"
            android : layout_marginLeft = "20dip"
            android : paddingLeft = "30dip"
            android : paddingRight = "30dip" />
    </RelativeLayout>
</LinearLayout>
```

Output



Getting Permission for Sending the SMS Messages

Both the sender and receiver need permissions to send and receive the messages. They are provided in the <manifest> element in the AndroidManifest.xml file.

```
<?xml version = "1.0" encoding = "utr-8"?>
<manifest xmlns : android = "http://schemas.android.com/apk/res/android"
    package = "com.androidunleashed.sendsmsapp"
    android : versionCode = "1"
    android : versionName = "1.0" >
    <uses-sdk android : minSdkVersion = "8"
        android : targetSdkVersion = "15" />
    <uses-permission android : name = "android.permission.SEND_SMS" />
    <application
        android : icon = "@drawable/ic_launcher"
        android : label = "@string/app_name"
        android : theme = "@style/AppTheme">
        <activity
            android : name = "SendSMSAppActivity"
            android : label = "@string/title_activity_send_smsapp" >
            <intent-filter>
                <action android : name = "android.intent.action.MAIN" />
                <category android : name = "android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Writing Java Code

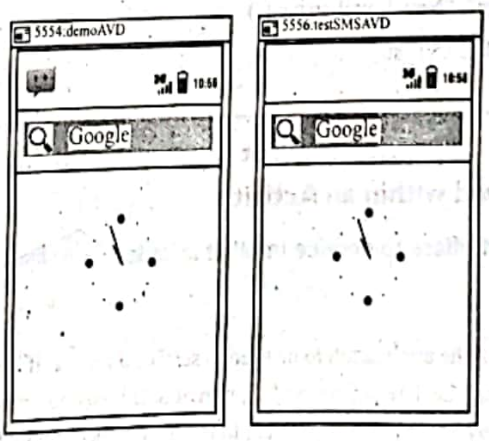
Code to create an activity to send the message is written in Java. It even acknowledges the user about the message delivery.

```
package com.androidunleashed.sendsmsapp;
import android.app.Activity;
import android.os.Bundle;
import android.telephony.smsManager;
import android.widget.Button;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Intent;
import android.content.Context;
import android.content.IntentFilter;
public class SendSMSApp extends Activity
```

```
{
    EditText phoneNumber, msg;
    BroadcastReceiver sentReceiver, deliveredReceiver;
    String SENT = "SMS_SENT";
    String DELIVERED = "SMS_DELIVERED";
    @Override
    public void onCreate(Bundle savedInstanceState) {
    super.onCreate (savedInstanceState);
    setContentView(R.layout.activity_send_smsapp);
    final PendingIntent sentPendIntent = PendingIntent.getBroadcast(this, 0, new Intent (SENT), 0);
    final PendingIntent delivered_pendintnet = PendingIntent.getBroadcast(this, 0, new Intent (DELIVERED), 0);
    sentReceiver = new BroadcastReceiver( ) {
        @Override
        public void onReceive(Context arg0, Intent arg1) {
            switch (getResultCode( )) {
                case Activity.RESULT_OK:
                    Toast.makeText(getBaseContext( ), "SMS sent", Toast.LENGTH_SHORT).show( );
                    break;
                case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                    Toast.makeText(getBaseContext( )), "Generic failure", Toast.LENGTH_SHORT.show( );
                    break;
                case SmsManager.RESULT_ERROR_NO_SERVICE:
                    Toast.makeText(getBaseContext( ), "No service", Toast.LENGTH_SHORT).show( )
                    break;
                case SmsManager.RESULT_ERROR_NULL_PDU:
                    Toast.makeText(getBaseContext( ), "Null PDU", Toast.LENGTH_SHORT). show( );
                    break;
                case SmsManager.RESULT_ERROR_RADIO_OFF:
                    Toast.makeText(getBaseContext( ), "Radio off", Toast.LENGTH_SHORT).show( );
                    break;
            }
        }
    };
    deliveredReceiver = new BroadcastReceiver( ) {
        @override
        public void onReceive(Context arg0, Intent arg1) {
        switch (getResultCode( )) {
            case Activity.RESULT_OK:
                Toast.makeText(getBaseContext( ), "SMS successfully delivered", Toast.LENGTH_SHORT). show( );
                break;
            case Activity.RESULT_CANCELED:
                Toast.makeText(getBaseContext( ), "Failure-SMS not delivered", Toast.LENGTH_SHORT).show( );
                break;
        }
    }
};
```

```
registerReceiver(sentReceiver, new IntentFilter(SENT));
registerReceiver(deliveredReceiver, new IntentFilter(DELIVERED));
Button btn = (Button) this.findViewById(R.id.send_button);
btn.setOnClickListener(new Button.onClickListener( ) {
public void onClick(View v) {
    phoneNumber = (EditText) findviewById(R.id.recvr_no);
    msg = (EditText) findViewById(R.id.txt_msg);
    if(phoneNumber.getText( ).toString( ).trim( ).length( ) >0 && message.getText( ).toString( ).trim( ).length( ) >0) {
        SmsManager sm = SmsManager.getDefault( );
        sm.sendTextMessage(phoneNumber.getText( ).toString( ), null,
        msg.getText( ).toString( ), sentPendIntent, delivered_pendintent);
    }
    else {
        Toast.makeText(SendSMSAppActivity.this, "phone number or text is missing",
                Toast.LENGTH_SHORT).show( );
    }
});
Button cbtn = (Button) this.findViewById(R.id.cancel_button);
cbtn.setOnClickListener(new Button.onClickListener( ) {
    public void onClick(View, v) {
        phoneNumber.setText(" ");
        msg.setText(" ");
    }
});
}
}
```

To run the application, two AVD's are required. One AVD is needed to send the message and other AVD is needed to receive the message.



3.2 BROADCAST RECEIVERS

Q17. Discuss about the broadcast receivers.

Model Paper-II, Q6(b)

Answer :

Broadcast Receiver

Broadcast receiver will receive and respond to the messages broadcasted by system and other applications. The messages like low battery, new mail, completed download, captured photo need response. These messages are broadcasted in the form of intent object. It waits for the response of the receiver. For example, the broadcast receivers can create the status bar notification to alert the user about broadcast.

The broadcast intent can invoke multiple receivers at a time.

Broadcasting an Intent

Broadcasting an Intent involves the following steps.

1. Creation of Intent object
2. Assigning specific action to the Intent object.
3. Attaching data or message to receiver.
4. Finally, Intent is broadcasted.

Some extra data or message can also be added to the Intent optionally.

The methods that are used to broadcast an intent are as follows,

1. **putExtra( )**

   It is used to add data or message to Intent that is to be sent to the broadcast receiver.

   Syntax

   putExtra(string_name, string_val)

2. **setAction( )**

   It is used to set the action that is to be performed on data or message to be sent with Intent.

   Syntax

   setAction(String action)

3. **sendBroadcast( )**

   It is used to send the broadcast Intent to the registered Intent receivers.

   Syntax

   void sendBroadcast(intent_to_broadcast)

Example

```
Public static String BROADCAST_str = "com.androidunleashed.testingbroadcast";
Intent broadcastInt = new Intent( );
broadcastInt.putExtra("message", "New Email arrived");
broadcastInt.setAction(BROADCAST_str);
sendBroadcast(broadcastInt);
```

3.2.1 Using Intent Filters to Service Implicit Intents, Resolving Intent Filters, Finding and Using Intents Received within an Activity

Q18. Explain about using intent filters to service implicit intents.

Answer :

Model Paper-III, Q7(a)

The Android will be unaware of the application to be used to service a request if it is an intent to perform an action on set of data. The application components can declare actions and supported data by using Intent Filters. An activity or service can be registered as potential Intent handler by adding intent_filter tag to manifest node through below defined tags.

1. **action**

This tag makes use of the android : name attribute to specify the action name to be serviced. The name of action must be unique and describe the action. Every Intent filter needs to have atleast one action tag.

2. **category**

It makes use of android : name attribute to specify the type of situations in which the action can be serviced. The Intent filter tag can include multiple category tags. It is possible to specify own categories or even use standard values which are provided by Android.

(i) **ALTERNATIVE**

It determines that the action must be available as alternative to default action which is performed on item of such type.

(ii) **SELECTED_ALTERNATIVE**

It is used when a group of possibilities are needed.

(iii) **HOME**

It acts as an alternative to native home screen.

(iv) **DEFAULT**

It sets components default action for data type in Intent Filter.

(v) **LAUNCHER**

It is used to launch an activity listed in application launcher.

(vi) **BROWSABLE**

It specifies an action which is available from the browser. An Intent in browser when fired will hold the browsable category. It is mandatory for the users to make the application respond to the actions that are triggered in browser.

3. **Data**

The data tag allows to specify the data types on which the component can act. There are various data tags that can be included if it is required. The following attributes can be combined to specify the supported data of component.

- **android : host**

  It specifies a valid hostname.

- **android : path**

  It specifies path values for URI.

- **android : port**

  It specifies valid port for particular host.

- **android : mimeType**

  It specifies the type of data that can be handled by component.

- **android : scheme**

  It specifies a scheme part of URI.

The below code depicts Intent Filter for an Activity which can perform SHOW_DAMAGE action as primary or alternative action depending upon mimetype.

```
<intent-filter>
<action android : name = "com.paid.earthquake.intent.action.SHOW_DAMAGE"/>
<category android : name = "android.intent.category.DEFAULT"/>
<category android : name = "android.intent.category.SELECTED_ALTERNATIVE"/>
<data android : mimeType = "vnd.earthquake.cursor.item/*"/>
</intent-filter>
```

Q19. Write short notes on the following,

(i) Resolving Intent filters

(ii) Finding and using Intents received within an activity.

Answer : Model Paper-III, Q7(b)

(i) Resolving Intent Filters

Intent resolution is the process of determining the Activity to be started when implicit Intent is passed to startActivity. The purpose of it is to find a possible and suitable Intent Filter through the following process,

1. Android combines all the intent filters which are available from installed packages.
2. The Intent filters that are not matchable with any action or category of resolving intent are eliminated from list.
   - ❖ If the Intent filter has specified an action then the action matches are made. If any of the actions does not match with the specified action, then the Intent filters fail the action match check.
   - ❖ The Intent Filters need to carry all the categories defined in resolving Intent. However, they can also handle the extra categories which are not present in Intent.
3. All the parts of Intent's data URI are compared with Intent Filter data tag. The values such as scheme, host/authority, path or MIME type specified by Intent filter are compared with Intent's URI. The Intent Filter will be eliminated from list if there is a mismatch. If no data is specified in Intent filter then it matches with all the Intent data values.
   - ❖ The MIME type is a data type related to the data that is being matched. The match subtypes can be used while matching the data types.
   - ❖ The scheme is referred as URI protocol part.
   - ❖ There is a section of URI called hostname or data authority which is present between scheme and path. The Intent Filter's scheme need to pass to match the hostname.
   - ❖ The data path matches only when scheme and hostname of data tag are matched.
4. If multiple components are resolved from the process of activity initiated implicitly then all the matches are provided to user.

(ii) Finding and Using Intents Received within an Activity

An application component must find the action that is to be performed along with the data when it is initiated through implicit intent. The getIntent( ) method is used to determine the intent that is used to start an activity. The following snippet code illustrates the getIntent method.

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Intent in = getIntent( );
    String a = intent.getAction( );
    Uri d = intent.getData( );
}
```

The getData( ) and getAction( ) methods are used for finding the data and action of Intent respectively. The additional information of extras Bundle can be extracted by using the type-safe get<type>Extra methods. The method getIntent( ) returns the starting Intent with which the Activity is created. In certain situations, the activity receives Intents ever after it is launched. The onNewIntent handler can be overridden in the activity for receiving as well as handling new Intents after the Activity creation.

```
public void onNewIntent(Intent new_I)
{
    super.onNewIntent(new_I);
}
```

3.3 NOTIFICATIONS

3.3.1 Creating and Displaying Notifications

Q20. Explain in detail about notification system.

Model Paper-I, Q7

Answer :

Notification

In android, notifications are the messages used to notify/alert the users regarding the events which require attention. These notifications are handled by notification manager.

Notification class is used to create an object and to provide the information displayed such as an icon with the text displayed on a status bar when a notification is extended and the number of times a notification is triggered etc.

Notification Manager

Notification manager class is used to define an object that manages the notifications. It displays the information that is hidden in a Notification object by using notify( ) method.

A method called getSystemService( ) is used to obtain a valid NotificationManager.

NotificationManager notif_manager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

Once the NotificationManager object is obtained, the user can invoke the notify( ) method.

Syntax

notify(uniqueID, notification_object)

Example

notification_manager.notify(0, notification);

Creating Notifications

Android provides various ways to inform users by using notifications such as statusbar icon, ringtone (sound), vibration, blinking lights etc.

Generally, the notifications are displayed via status bar in the form of an icon along with some ticker text (optional). The user can simply pull down the status bar to view the notification list and clear all the notifications using the clear button on status bar. Suppose, if a user clicks on notification, it switches to that Intent defined by the notification and then the notification will disappear from the status bar automatically. Note that, the status bar only displays the notifications to the user and launches the particular application only when it is selected by the user.

The creation of notifications involves the following steps,

1. Create a notification object.

   Notification notify = new Notification( );

2. Configure the object with the notification properties.

   notify.icon = R.drawable.icon;

   notify.tickerText = "You got a new notification";

   notify.when = System.currentTimeMillis( );

   notify.flags = Notification.FLAG_AUTO_CANCEL;

The different public members used to configure the notify object are,

(i) **icon**

It is used to assign the notification icon.

(ii) **tickerText**

It is used to assign the small notification text.

(iii) **when**

It is used to assign the time of occurrence. Here, the system time is used to specify the time of occurrence of a notification.

(iv) **flag**

It is used to assign the constant that finds further action to be performed when the notification is clicked. In general, the flag member is assigned to FLAG_AUTO_CANCEL which means that specific notification will disappear from the status bar once it is clicked.

A user can also assign the notification icon, tickerText, the time of occurrence in a single statement as given below,

Notification notify = new Notification(R.drawable.ic_launcher, "you got a new notification", SystemcurrentTimesMillis( ));

Creating PendingIntent

When a notification is received, the user clicks on it to perform a specific action. Therefore, a class called "PendingIntent" is used to redirect to the required Intent. It allows the user to create Intents that can be triggered by the existing applications in the device when the event occurs. A sample code snippet to create the PendingIntent is as follows,

```
int req_code = 0,
int f = 0;
Intent in = new Intent(getBaseContext( ), TargetActivity.class);
PendingIntent p-Intent = PendingIntent.getActivity(getBaseContext( ), 0, in, 0);
```

In the above code, an Intent object called "in" is created by passing the current application context and the activity class "TargetActivity.class" as parameters. In the next statement, a pending Intent object is created by passing four parameters. They are as follows,

(i) The current(getBaseContext( )) is the application context in which the PendingIntent called p-Intent launches the activity.

(ii) The request code for a sender is not used therefore, '0' is passed.

(iii) The Intent of an activity (in) that is about to launch.

(iv) The 'in is a flag to specify the unspecified part of the intent to be sent. As there is no unspecified part, the value '0' is passed to flag.

The setLatestEventInfo( ) is used to show the text after expansion of the notification and the other pending Intent to be launched. This method is derived from Notification class.

Syntax

```
setLatestEventInfo(application_context, title, text, pending_intent);
```

Here,

application_context is the current application context

title is the title of the notification

text specifies the text after the expansion of a notification

pending_intent is an object of PendingIntent class to pass the information, when a user clicks the notification.

Example

```
notify.setLatestEventInfo(getBaseContext( ), "E-mail", "3 unread messages", p-Intent);
```

Notification-Builder

Notification-Builder is a builder class used to configure the notifications by using several methods. They are as follows,

1. **setAutoCancel( )**

This method is used to find whether a user wishes to make the notification invisible when it is clicked. Pass the boolean value true to make the notification invisible.

Syntax

```
setAutoCancel(Boolean autoCancel);
```

2. **setWhen( )**

This method is used to provide the time of occurrence of a notification.

Syntax

```
setWhen(long timeofOccurrence);
```

3. **setContentText( )**

This method is used to provide the text for a notification.

Syntax

setContextText(CharSequence text);

4. **setContentTitle( )**

This method is used to provide the title of the notification after the status bar is expanded.

Syntax

setContentText(CharSequence title);

5. **setSmallIcon( )**

This method is used to display a small icons representing the notification in the status bar.

Syntax

setSmallIcon(int icon);

6. **setTicker( )**

This method is used to provide the ticker text to be displayed when the user receives a notification.

Syntax

setTicker(CharSequence textMessage);

7. **setContentIntent( )**

This method is used to provide a PendingIntent that is to be sent when the notification is clicked.

Syntax

setContentIntent(PendingIntent intnt);

The usage of the above mentioned Notification.Builder class methods is as follows,

```
Notification.Builder b = new Notification.Builder(MyActivity.this())
b.setSmallIcon(R.drawable.ic_launcher)
.setAutoCancel(true)
.setTicker("you have a new notification")
.setWhen(System.currentTimeMillis( ))
.setContentTitle("E-mail")
.setContentText("You have three unread messages.")
.setContentIntent(P-Intent);
Notification notification1 = builder.getNotification( );
```

Q21. Create an android application that creates notification to the users.

Answer :

main.xml

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

```
<Button
    android : id = "@ + id/create_btn"
    android ; layout_width = "wrap_content"
    android : layout_height = "wrap_content"
    android : text = "Create a Notification"
    android : layout_gravity = "center"/>
</LinearLayout>
```

trgt.xml

```
<?xml version = "1.0" encoding = "utf-8"?>
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:orientation = "vertical">
<Textview
    android:id = "@+id/mesg_view"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:text = "target activity"
    android:layout_gravity = "center"/>
</LinearLayout>
```

TargetActivity.java

```
package com.androidunleashed.notificationapp;
import android.app.Activity;
import android.os.Bundle;
public class TargetActivity extends Activity {
    @override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.trgt)
    }
}
```

NotificationExActivity.java

```
package com.androidunleashed.notificationapp;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
```

```
import android.widget.Notification;
import android.widget.Button;
import android.view.View.onClickListener;
public class NotificationExActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button create_btn = (Button) findViewById(R.id.create_btn)
        create_btn.setOnClickListener(new onClickListener( ) {
            @Override
            public void onClick(View arg0) {
                Intent intnt = new Intent (getBaseContext ( ), TargetActivity.class);
                pendingIntent p-Intent = pendingIntent. getActivity(getBaseContext ( ), 0, intent, 0);
                NotificationManager notification_manager = (NotificationManager) getSystemService
                                                                    (NOTIFICATION_SERVICE);
                Notification note = new Notification( );
                Notification.Builder builder = new Notification.Builder(getBaseContext( ))
                .setSmallIcon(R.drawable.ic_launcher)
                .setAutoCancel(true)
                .setTicker("you have a new notification")
                .setWhen(System.currentTimeMillis( ))
                .setContentTitle("E-mail")
                .setContentText("You have three unread messages")
                .setContentIntent(p-Intent);
                note = builder.getNotification ( );
                notification_manager.notify(0, note);
            }
        });
    }
}
```

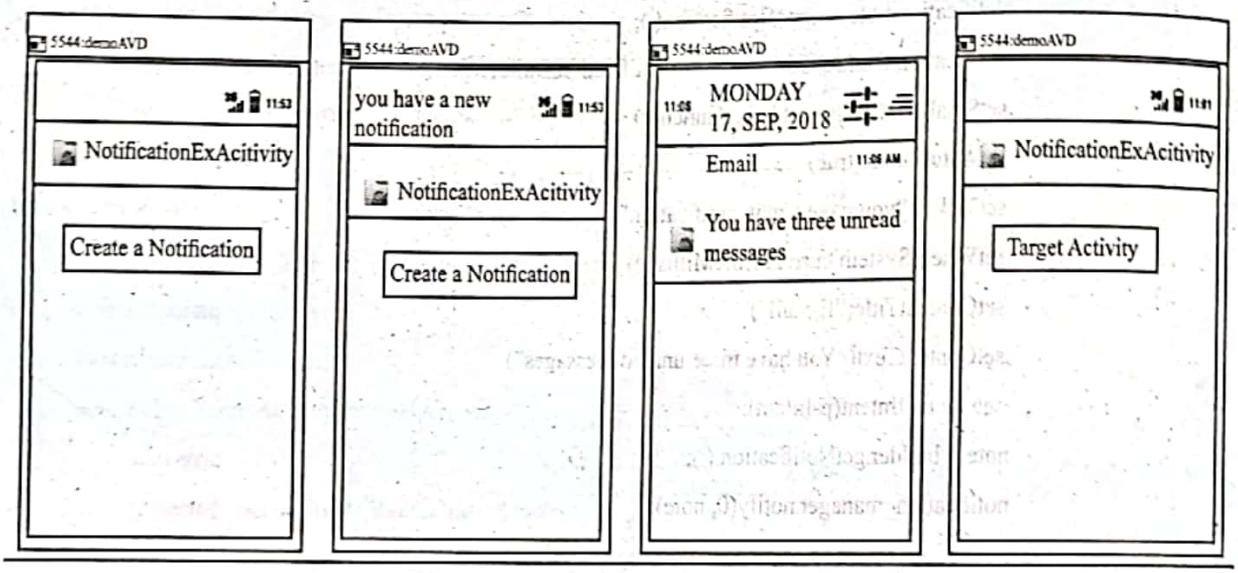
Add the Java Activity File to the AndroidManifest.xml to Recognize the Activity File .

AndroidManifest.xml

```
<?xml version = "1.0" encoding = "utf-8"?>
<manifest xmlns : android = "http://schemas.android.com/apk/res/android"
    package = "com.androidunleashed.notificationapp".
    android : versionCode = "1"
    android : versionName = "1.0" >
    <uses-sdk android:minSdkVersion = "8"
        android : targetSdkVersion = "15"/>
```

```
<application
    android : icon = "@drawable/ic_launcher"
    android : lable = "@string/app_name"
    android : theme = "@style/AppTheme" >
    <activity
        android : name = ".NotificationAppActivity"
        android : label = "@string/title_activity_notification_app" >
        <intent-filter>
            <action android : name = "android.intent.action.MAIN" />
            <category android : name = "android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android : name = ".TargetActivity" android : label = "@string/app_name" />
</application>
</manifest>
```

Output



3.3.2 Displaying Toasts

Q22. Write the usage of toast to display messages.

Answer : Model Paper-III, Q6(b)

The toast is defined as a view or notification that shows certain information for specific period of time. It pops up on the screen to display a message for the user. A normal message is the text that is communicated between the client and server. A toast message is the a text that is shown to the user about a problem or something else.

The toast class contains two methods which are as follows,

(i) makeText( )

This method is used to create an instance of the Toast class. The general form of it is as follows,

static Toast.makeText(word, string, length)

In the above general form, the first parameter is a word that represents content, the second parameter is a string the represents the message to be displayed and third parameter is length that represents the duration of the message to be displayed. This parameter accepts LENGTH_SHORT or LENGTH_LONG.

(iii) show()

This method is used to display the toast. If this method is not used then toast is not displayed.

Example

```
import android.support.V7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;
public class MainActivity extends AppcompatActivity
{
protected void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
Toast.makeText(getApplicationContext( ), "SendingMessage...", Toast.LENGTH_SHORT).show( );
}
}
```

Output