CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | |

**Internal Assessment Test 3– Sept. 2023**

| **Sub:** | **Mobile Applications** | | | | | | | **Sub Code:** | **20MCA263** |
|---|---|---|---|---|---|---|---|---|---|
| **Date:** | **27/09/2023** | **Duration:** | **90 min's** | **Max Marks:** | **50** | **Sem:** | **II** | **Branch:** | **MCA** |

**Note : Answer FIVE FULL Questions, choosing ONE full question from each Module**

| | | MARKS | OBE | |
|---|---|---|---|---|
| | | | CO | RBT |
| | **PART I** | | | |
| 1 | Define Fragment. Explain the life cycle of fragment? | [10] | CO1 CO2 | L1 |
| | **OR** | | | |
| 2 | Develop a Mobile Application to create a list of fruits using listview and display the item selected by the user? | [10] | CO3 | L3 |
| | **PART II** | | | |
| 3 | What is the use of TimePicker Dialog? Explain with an example. | [10] | CO3 CO4 | L3 |
| | **OR** | | | |
| 4 | Illustrate with an example how to play video in an android application | [10] | CO3 CO4 | L3 |
| | **PART III** | | | |
| 5 | Discuss APK file deployment in detail and steps involved in publishing android application | [10] | CO5 | L2 |
| | **OR** | | | |
| 6 | Explain the process of getting input via dialog box with an example? | [10] | CO3 CO2 | L2 |
| | **PART IV** | | | |
| 7 | Describe the steps for obtaining Google maps API key? | [10] | CO3 | L2 |
| | **OR** | | | |
| 8 | What is content provider list and explain built in content providers. | [10] | CO1 CO2 | L2 |
| | **PARTV** | | | |
| 9 | Explain the process of sending an SMS with an example? | [10] | CO3 CO4 | L3 |
| | **OR** | | | |
| 10 | What is a service? What are the different methods use to create a service? Explain with an example. | [10] | CO1 CO2 | L2 |

1. **Define Fragment. Explain the life cycle of fragment?**

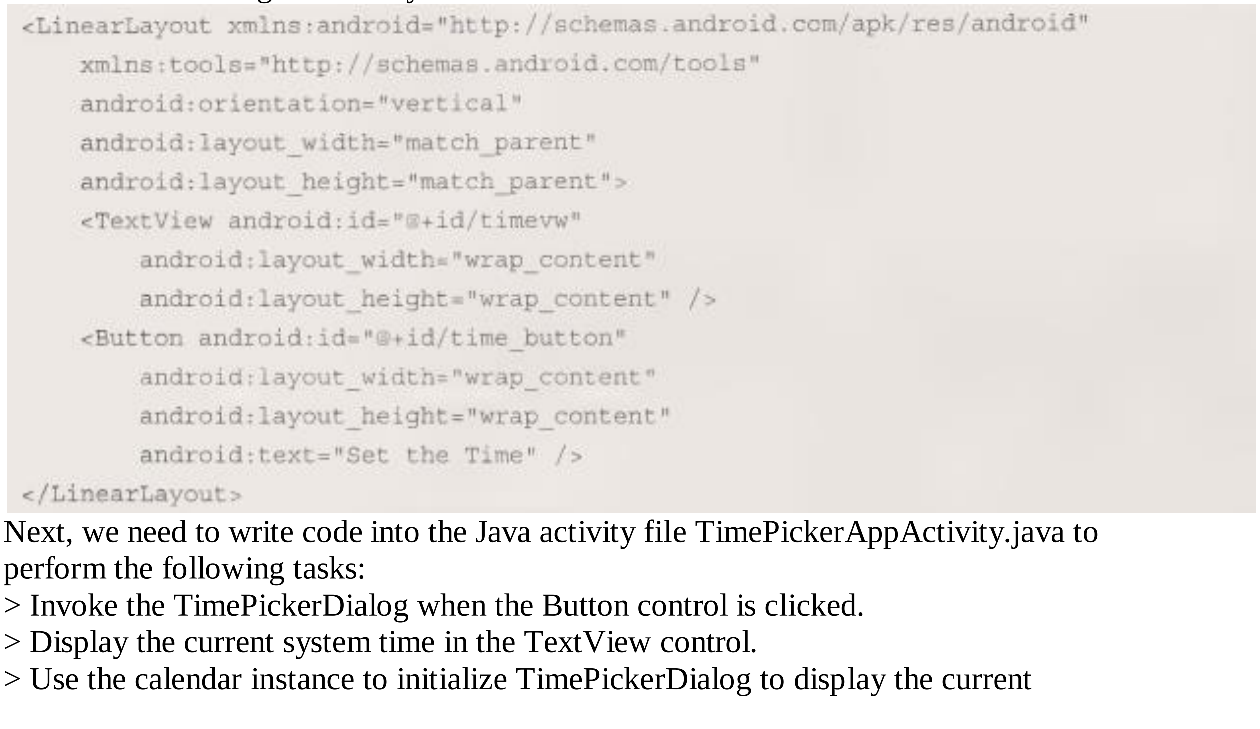
A fragment is a combination of an activity and a layout and contains a set of views that make up an independent and atomic user interface. For example, one or more fragments can be embedded in the activity to fill up the blank space that appears on the right when switching from portrait to landscape.
onAttach()—Called when the fragment is attached to the activity.
onCreate()—Called when creating the fragment. The method is used to initialize the items of the fragment that we want to retain when the fragment is resumed after it is paused or stopped. For example, a fragment can save the state into a Bundle object that the activity can use in the oncreate() callback while re-creating the fragment.
onCreateView()—Called to create the view for the fragment.
onActivityCreated()—Called when the activity's oncreate() method is returned.
onStart ()—Called when the fragment is visible to the user. This method is associated with the activity's onstart ().
onResume ()—Called when the fragment is visible and is running. The method is associated with the activity's onResume ().
onPause()—Called when the fragment is visible but does not have focus. The method is attached to the activity's onPause().
onStop()—Called when fragment is not visible. The method is associated with the activity's onStop().
onDestroyView()—Called when the fragment is supposed to be saved or destroyed. The view hierarchy is removed from the fragment.
onDestroy()—Called when the fragment is no longer in use. No view hierarchy is associated with the fragment, but the fragment is still attached to the activity.
onDetach()—Called when the fragment is detached from the activity and resources allocated to the fragment are released.

**2.Develop a Mobile Application to create a list of fruits using listview and display the item selected by the user?**
**XML File**
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <ListView
      android:id="@+id/listView1"
      android:layout_width="match_parent"
      android:layout_height="277dp" >
   </ListView>

   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="TextView" />

</LinearLayout>
```

**Java File**
```
package com.example.listex2;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.AdapterView.OnItemClickListener;


public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final String[] fruits={"apple","mango","grapes","orange","banana"};
        final TextView selectedOpt=(TextView)findViewById(R.id.textView1);
        ListView fruitsList=(ListView)findViewById(R.id.listView1);

    final ArrayAdapter<String> arrayAdpt=new
ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,fruits);
    fruitsList.setAdapter(arrayAdpt);
    fruitsList.setOnItemClickListener(new OnItemClickListener(){
            @Override
            public void onItemClick(AdapterView<?> parent,View v,int position,long id)
            {
                        selectedOpt.setText("you have selected"+fruits[position]);
            }
    });
}


}
```
**3. What is the use of TimePicker Dialog? Explain with an example.**

Time Picker

The TimePickerDialog allows us to set or select time through the built-in Android TimePicker view. We can set the values of hour and minute with values of hour ranging from o through 23 and minutes from 0 through 59. The dialog provides a callback listener, OnTimeChangedListener Or OnTimeSetListener, which tells us when a time is changed or set by the user.



```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView android:id="@+id/timevw"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button android:id="@+id/time_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Set the Time" />
</LinearLayout>
```

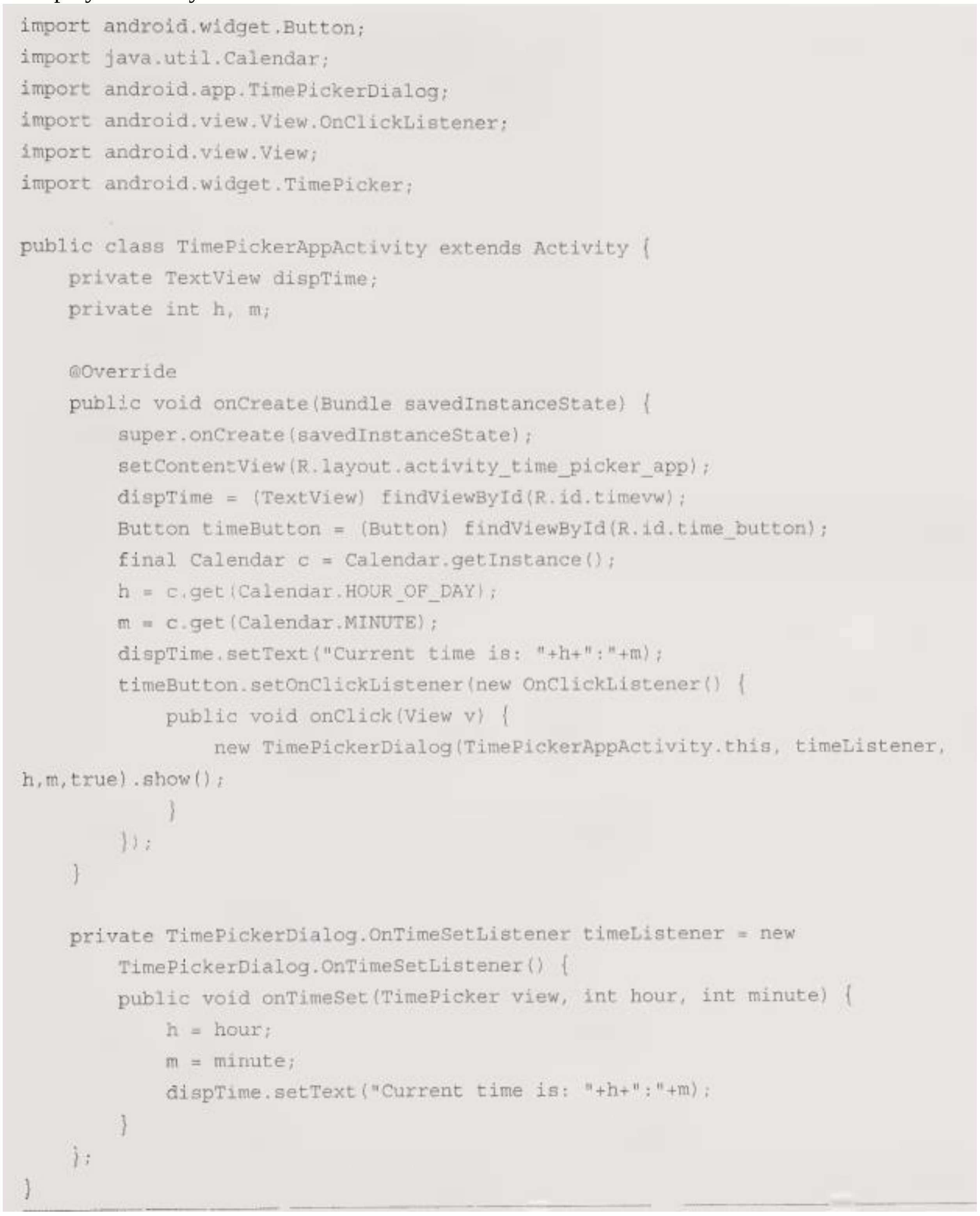
Next, we need to write code into the Java activity file TimePickerAppActivity.java to perform the following tasks:

> Invoke the TimePickerDialog when the Button control is clicked.

> Display the current system time in the TextView control.

> Use the calendar instance to initialize TimePickerDialog to display the current

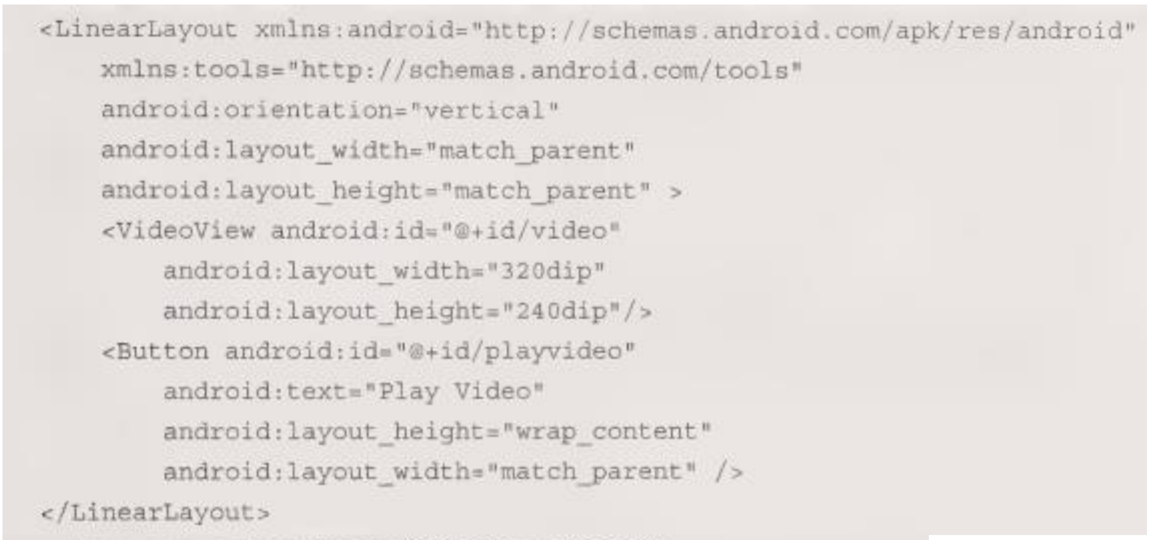
system time.
> Display the newly set time in the TextVview control.



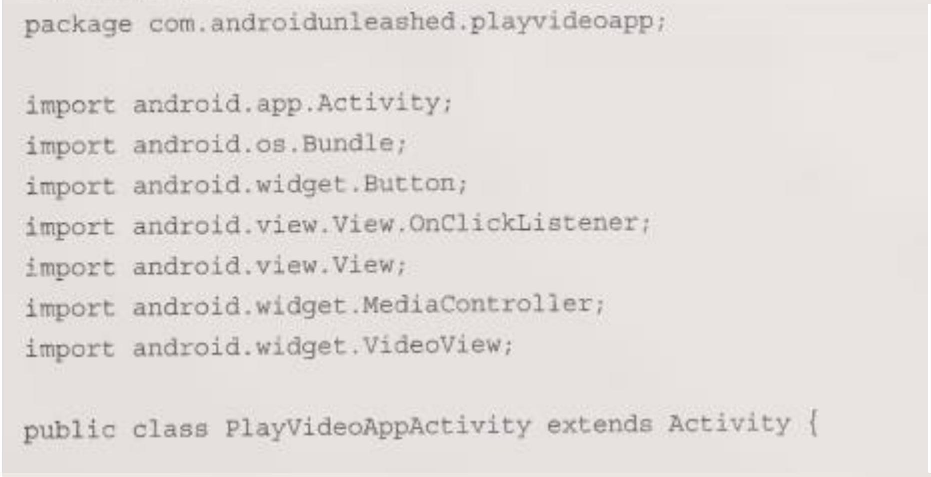
```
import android.widget.Button;
import java.util.Calendar;
import android.app.TimePickerDialog;
import android.view.View.OnClickListener;
import android.view.View;
import android.widget.TimePicker;

public class TimePickerAppActivity extends Activity {
    private TextView dispTime;
    private int h, m;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_time_picker_app);
        dispTime = (TextView) findViewById(R.id.timevw);
        Button timeButton = (Button) findViewById(R.id.time_button);
        final Calendar c = Calendar.getInstance();
        h = c.get(Calendar.HOUR_OF_DAY);
        m = c.get(Calendar.MINUTE);
        dispTime.setText("Current time is: "+h+":"+m);
        timeButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                new TimePickerDialog(TimePickerAppActivity.this, timeListener,
h,m,true).show();
            }
        });
    }

    private TimePickerDialog.OnTimeSetListener timeListener = new
        TimePickerDialog.OnTimeSetListener() {
        public void onTimeSet(TimePicker view, int hour, int minute) {
            h = hour;
            m = minute;
            dispTime.setText("Current time is: "+h+":"+m);
        }
    };
}
```

**4. Illustrate with an example how to play video in an android application**

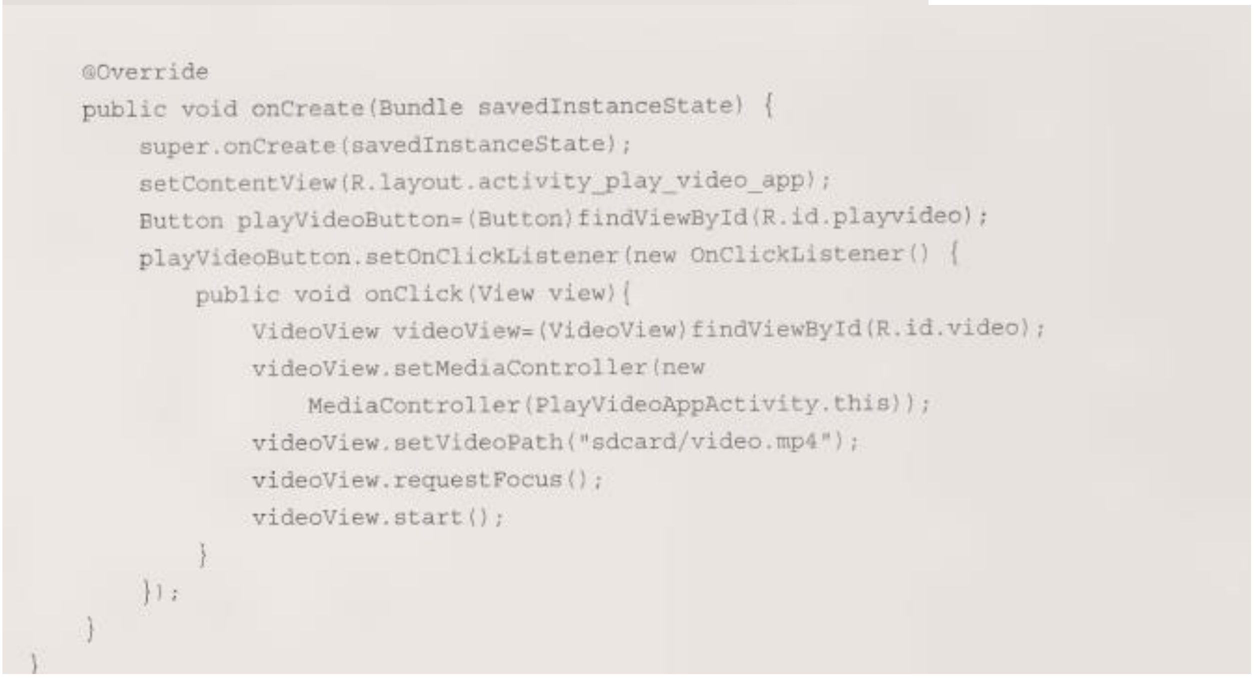
**Playing Video:**
To play video in an application, Android provides a videoview control, which, along with the MediaController, provides several buttons for controlling video play. These buttons allow us to play, pause, rewind, and fast-forward the video content displayed via the VideovView control. To understand the steps for playing a video, let's create a new Android project called PlayvideoApp. We can play a video that is available on the Internet or one that is loaded onto an SD card of our device or emulator.



```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <VideoView android:id="@+id/video"
        android:layout_width="320dip"
        android:layout_height="240dip"/>
    <Button android:id="@+id/playvideo"
        android:text="Play Video"
        android:layout_height="wrap_content"
        android:layout_width="match_parent" />
</LinearLayout>
```



```
package com.androidunleashed.playvideoapp;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.view.View.OnClickListener;
import android.view.View;
import android.widget.MediaController;
import android.widget.VideoView;

public class PlayVideoAppActivity extends Activity {
```



```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_play_video_app);
        Button playVideoButton=(Button)findViewById(R.id.playvideo);
        playVideoButton.setOnClickListener(new OnClickListener() {
            public void onClick(View view){
                VideoView videoView=(VideoView)findViewById(R.id.video);
                videoView.setMediaController(new
                    MediaController(PlayVideoAppActivity.this));
                videoView.setVideoPath("sdcard/video.mp4");
                videoView.requestFocus();
                videoView.start();
            }
        });
    }
}
```

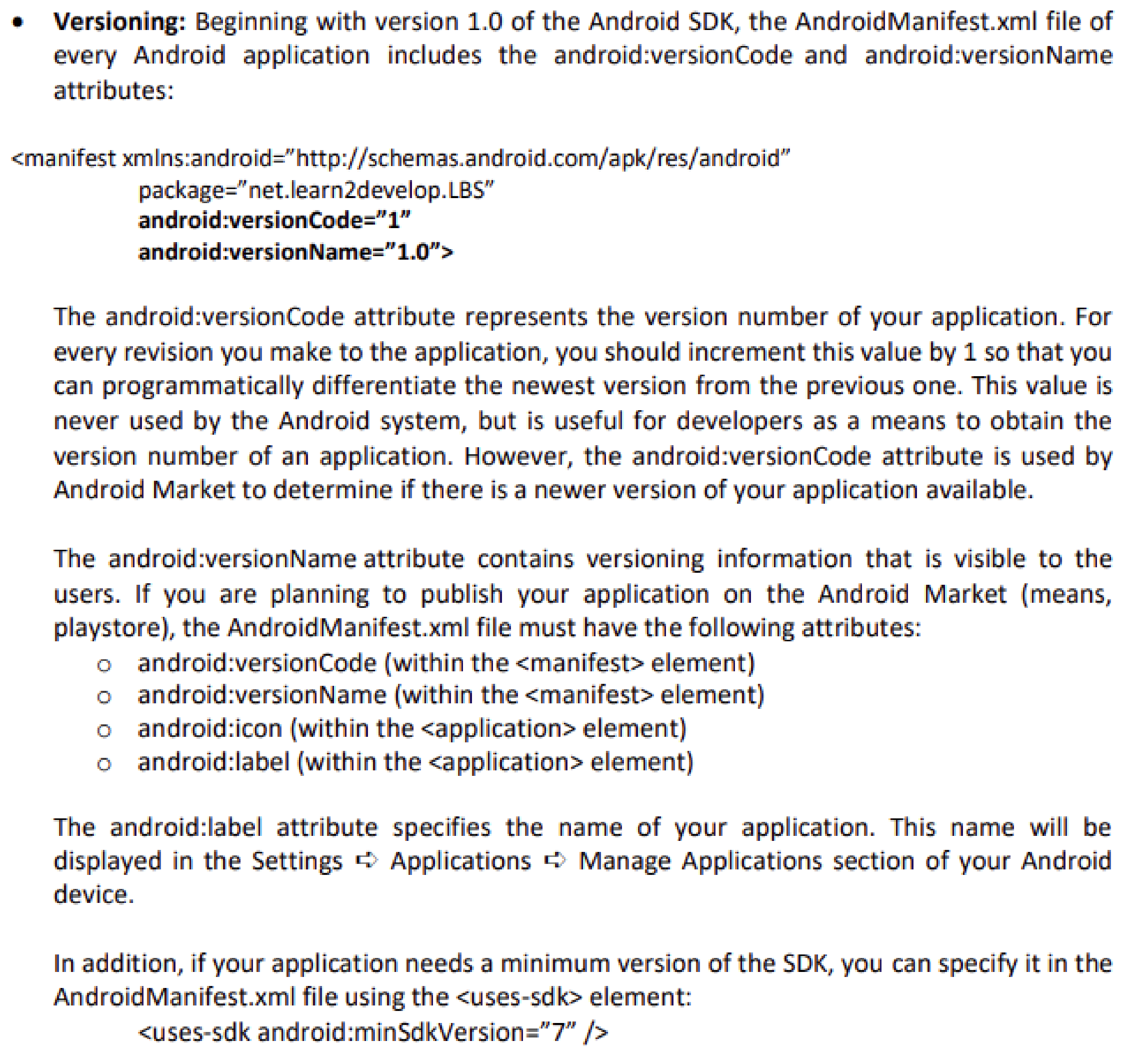
We capture the videoview control from the layout and map it to the videoview object. Then we use a MediaController and set it the media controller of the videoview object. The videoview object is used for displaying video content and the button controls that enable us to perform play, pause, rewind, or fast-forward actions on the video. A MediaController provides these buttons. Hence the videoview's media controller is set by calling setMediaController() to display the different button controls. Then, we use the setVideoPath() method of the videoview object to refer to an SD card (sdcard) for the video.mp4 file. We can also use set VideoURI() method to access the video from the Internet. After setting the focus to the videoview control through request Focus () method, we use its start () method to start the video.

**5. Discuss APK file deployment in detail and steps involved in publishing android application.**

Following are the steps to publish an Android application:

1. Set the versioning information of the application.
2. Generate a certificate, digitally sign the Android | application, and generate the APK (Android Package) | file. Android applications are distributed as Android package files (.APK).
3. Distribute to Google Play or other marketplace to host and sell our application.

## Setting Versioning Information of an Application



- **Versioning:** Beginning with version 1.0 of the Android SDK, the AndroidManifest.xml file of every Android application includes the android:versionCode and android:versionName attributes:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="net.learn2develop.LBS"
          android:versionCode="1"
          android:versionName="1.0">
```

The android:versionCode attribute represents the version number of your application. For every revision you make to the application, you should increment this value by 1 so that you can programmatically differentiate the newest version from the previous one. This value is never used by the Android system, but is useful for developers as a means to obtain the version number of an application. However, the android:versionCode attribute is used by Android Market to determine if there is a newer version of your application available.

The android:versionName attribute contains versioning information that is visible to the users. If you are planning to publish your application on the Android Market (means, playstore), the AndroidManifest.xml file must have the following attributes:

- android:versionCode (within the <manifest> element)
- android:versionName (within the <manifest> element)
- android:icon (within the <application> element)
- android:label (within the <application> element)

The android:label attribute specifies the name of your application. This name will be displayed in the Settings ➪ Applications ➪ Manage Applications section of your Android device.

In addition, if your application needs a minimum version of the SDK, you can specify it in the AndroidManifest.xml file using the <uses-sdk> element:

```
<uses-sdk android:minSdkVersion="7" />
```

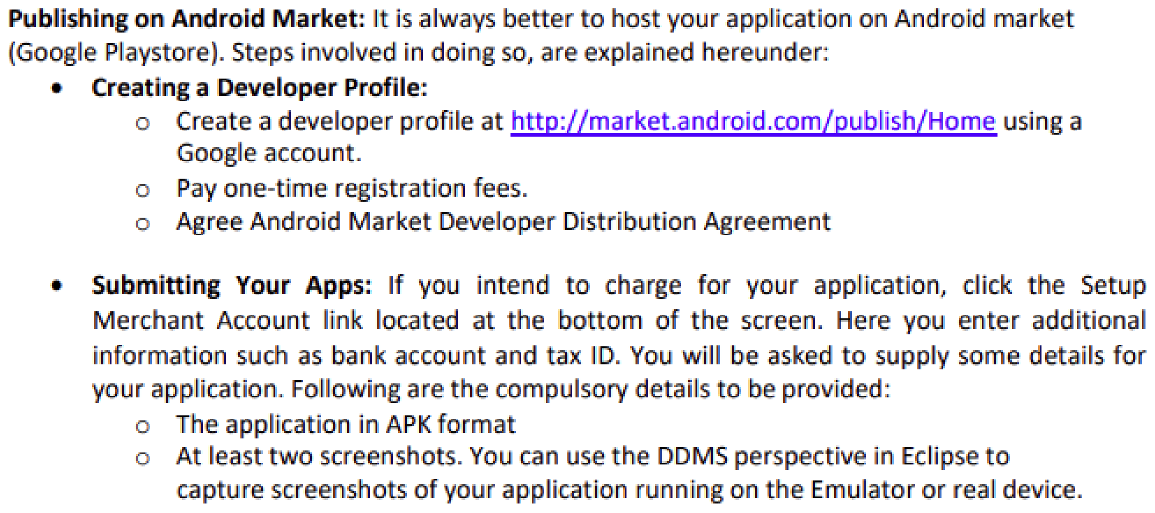
We need three things to essentially inform users about our application:
» The features that our application requires to run—for example, whether it requires Bluetooth, multitouch screen, and so on, to operate
» The necessary hardware configuration that our application requires for running
» The permissions that our application requires to operate
To inform users about these three things, we use the three tags <uses-features>, <uses-configuration>, and <uses-permission> in our AndroidManifest.xml file.

**Generating a Certificate, Digitally Signing the Android Applications, and Generating the APK**
Signing Applications Using the Export Android Application Wizard
The Export Android Application Wizard simplifies the process of creating and signing our application package. To launch the wizard, follow these steps:
1. Select the Android project in the Package Explorer window and select the File, Export Option or right-click the project in the Package Explorer window and select the Export option.
2. In the Export dialog, expand the Android item and select Export Android Application. Click Next.
3. Our Android project name, createServiceApp, is displayed in the Project box. Click Next.
4. The next dialog prompts us to either select an existing keystore or create a new one. Select the create new keystore option to create a new certificate, that is, keystore, for our application. In the Location box, specify

the name and path of the keystore. Assign the name as CreateService to our new keystore. In the Password and Confirm boxes, enter the password to protect the keystore. After entering the password , click Next.
5. Provide an alias for the private key and enter a password to protect the private key. Applications published on Google Play require a certificate with a validity period ending after October 22, 2033. Hence, enter a number that is greater than 2033 minus the current year in the validity box. Also fill the First and Last Name box with your name and click Next.
6. Enter a path to store the destination APK file. Click Finish.
Once you have signed your APK files, you need a way to get them onto your users' devices. Three methods are here:

- Deploying manually using the adb.exe tool
- Hosting the application on a web server
- Publishing through the Android Market

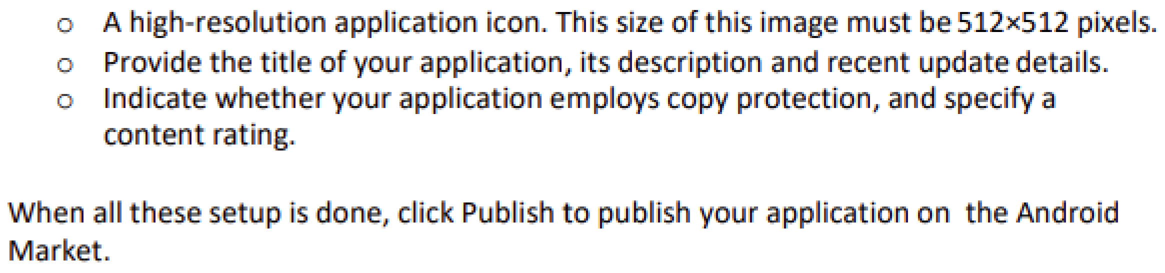
**Distributing Applications with Google Play**



**Publishing on Android Market:** It is always better to host your application on Android market (Google Playstore). Steps involved in doing so, are explained hereunder:

- **Creating a Developer Profile:**
    - Create a developer profile at http://market.android.com/publish/Home using a Google account.
    - Pay one-time registration fees.
    - Agree Android Market Developer Distribution Agreement

- **Submitting Your Apps:** If you intend to charge for your application, click the Setup Merchant Account link located at the bottom of the screen. Here you enter additional information such as bank account and tax ID. You will be asked to supply some details for your application. Following are the compulsory details to be provided:
    - The application in APK format
    - At least two screenshots. You can use the DDMS perspective in Eclipse to capture screenshots of your application running on the Emulator or real device.



- A high-resolution application icon. This size of this image must be 512×512 pixels.
- Provide the title of your application, its description and recent update details.
- Indicate whether your application employs copy protection, and specify a content rating.

When all these setup is done, click Publish to publish your application on the Android Market.

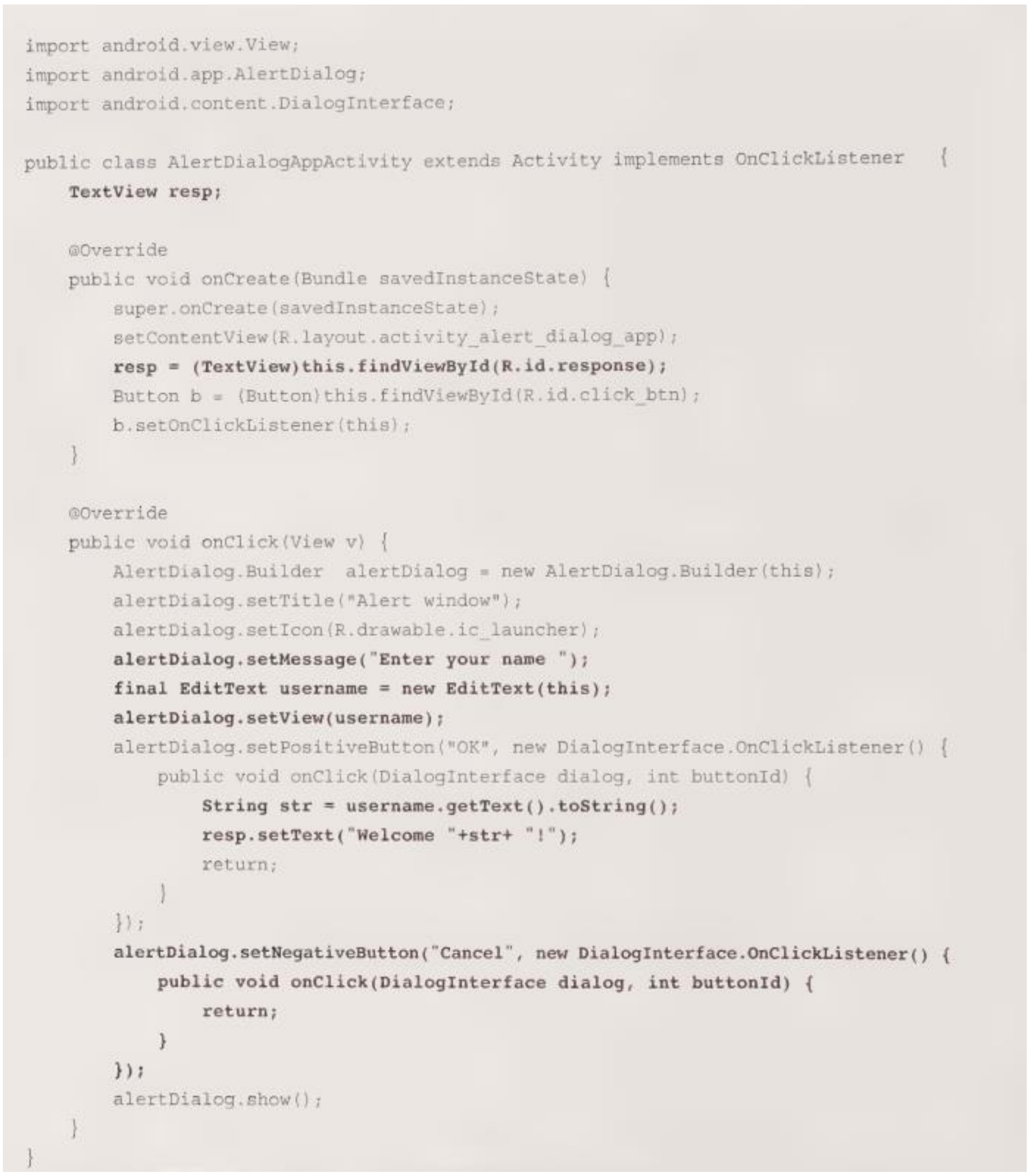
**6. Explain the process of getting input via dialog box with an example?**
An AlertDialog is a popular method of getting feedback from the user. This pop-up dialog remains there until closed by the user and hence is used for showing critical messages that need immediate attention or to get essential feedback before proceeding further.
In the appliaction
> Dynamically create an EditText control and set it as part of the AlertDialog to prompt the user for input.
> Add a TextView control to the layout file to display the data entered by the user in AlertDialog.
To make it more specific, our application asks the user to input a name through AlertDialog, and when the user selects the ok button after entering a name, a welcome message is displayed through the TextView control defined in the layout file. We also add a Cancel button to the AlertDialog, allowing the user to cancel the operation, which terminates the dialog.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/click_btn"
        android:text="Click for Alert Dialog"/>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/response"/>
</LinearLayout>
```

Next we add code to the Java Activity file AlertDialogAppActivity.java to do the following tasks:

> Dynamically create an EditText control and set it as the content of the AlertDialog.
> Access the TextView control from the layout file main.xml and map it to a TextView object.
> Fetch the name entered by the user in the EditText control and assign it to the TextView object for displaying a welcome message.
>Register an event listener for the cancel button. Recall that the purpose of the Cancel button is to cancel the operation and terminate the AlertDialog.

```
package com.androidunleashed.alertdialogapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
```



```
import android.view.View;
import android.app.AlertDialog;
import android.content.DialogInterface;

public class AlertDialogAppActivity extends Activity implements OnClickListener   {
    TextView resp;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_alert_dialog_app);
        resp = (TextView)this.findViewById(R.id.response);
        Button b = (Button)this.findViewById(R.id.click_btn);
        b.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        AlertDialog.Builder  alertDialog = new AlertDialog.Builder(this);
        alertDialog.setTitle("Alert window");
        alertDialog.setIcon(R.drawable.ic_launcher);
        alertDialog.setMessage("Enter your name ");
        final EditText username = new EditText(this);
        alertDialog.setView(username);
        alertDialog.setPositiveButton("OK", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int buttonId) {
                String str = username.getText().toString();
                resp.setText("Welcome "+str+ "!");
                return;
            }
        });
        alertDialog.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int buttonId) {
                return;
            }
        });
        alertDialog.show();
    }
}
```

**7. Describe the steps for obtaining Google maps API key?**

We  need to apply for a free Google Maps API key before you can integrate Google Maps into your Android application. The steps for obtaining a Google key are as follows:

1. To get a Google key, the application needs to be signed with a certificate and you need to notify Google about the Hash (MDS) fingerprint of the certificate. To test the application on the Android emulator, search for the SDK debug certificate located in the default folder: c:\Users\<user_name\.android. The filename of the debug certificate is debug. keystore. For deploying to a real Android device, substitute the debug.keystore file with your own keystore file.
2. Copy the debug.keystore to any drive.
3. Using the debug keystore certificate, extract its MDS fingerprint using the keytool. exe application provided with the JDK installation. This fingerprint is needed to apply for the free Google Maps key. The keytool .exe file can be found in the c: \ Program Files\Java\jdk_version_number\bin folder.
4. Open the command prompt and go to the c:\Program Files\Java\jdk_version_ number\bin folder using the cb command.
5. Issue the following command to extract the MDS fingerprint:

keytool.exe -list -alias androiddebugkey -keystore "E:\debug.keystore" -storepass android -keypass android

Now you need to sign up for the Google Maps API. Open the browser and go to http: // code .google.com/android/maps-api-signup.htm1. Follow the instructions on the page and supply the extracted MDS fingerprint to complete the signup process and obtain the Google Maps key. After successful completion of the signup process, the Google Maps API key is displayed.

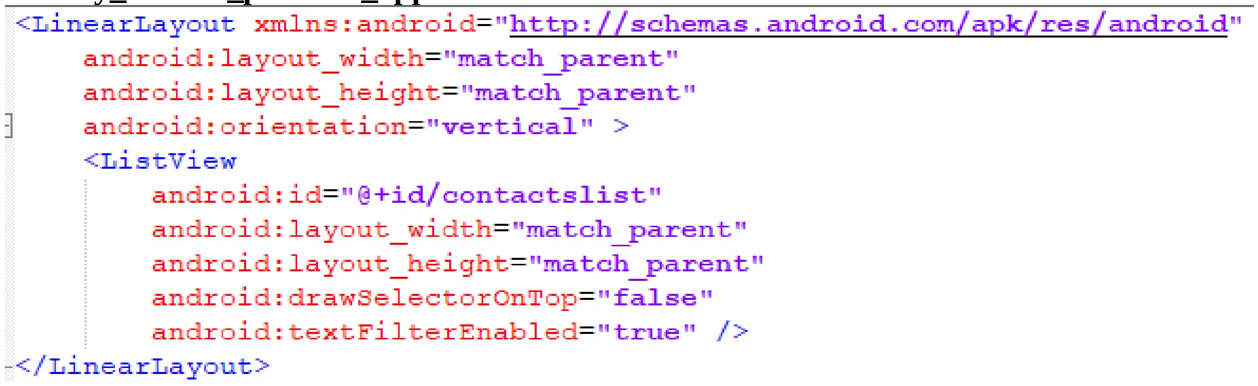
**8. What is content provider list and explain built in content providers.**

A content provider acts as a data store and provides an interface to access its contents. Unlike a database, where information can be accessed only by the package in which it was created, information in a content provider can be shared across packages. The following lists a few characteristics of content providers:
> Like in a database, we can query, add, edit, delete, and update data in content providers.
» Data can be stored in a database, files, and over a network.
>» A content provider acts as a wrapper around the data store to make it resemble web services. That is, the data in content providers is exposed as a service.

**Content Providers Application**

The contact information on our device can be accessed in an Android application. Let's create a new Android project called content ProviderApp. This application accesses the contact information and displays it via Listview.

**Activity_content_provider_app.xml**



```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <ListView
        android:id="@+id/contactslist"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:drawSelectorOnTop="false"
        android:textFilterEnabled="true" />
</LinearLayout>
```

**ContentProviderAppActivity.java**

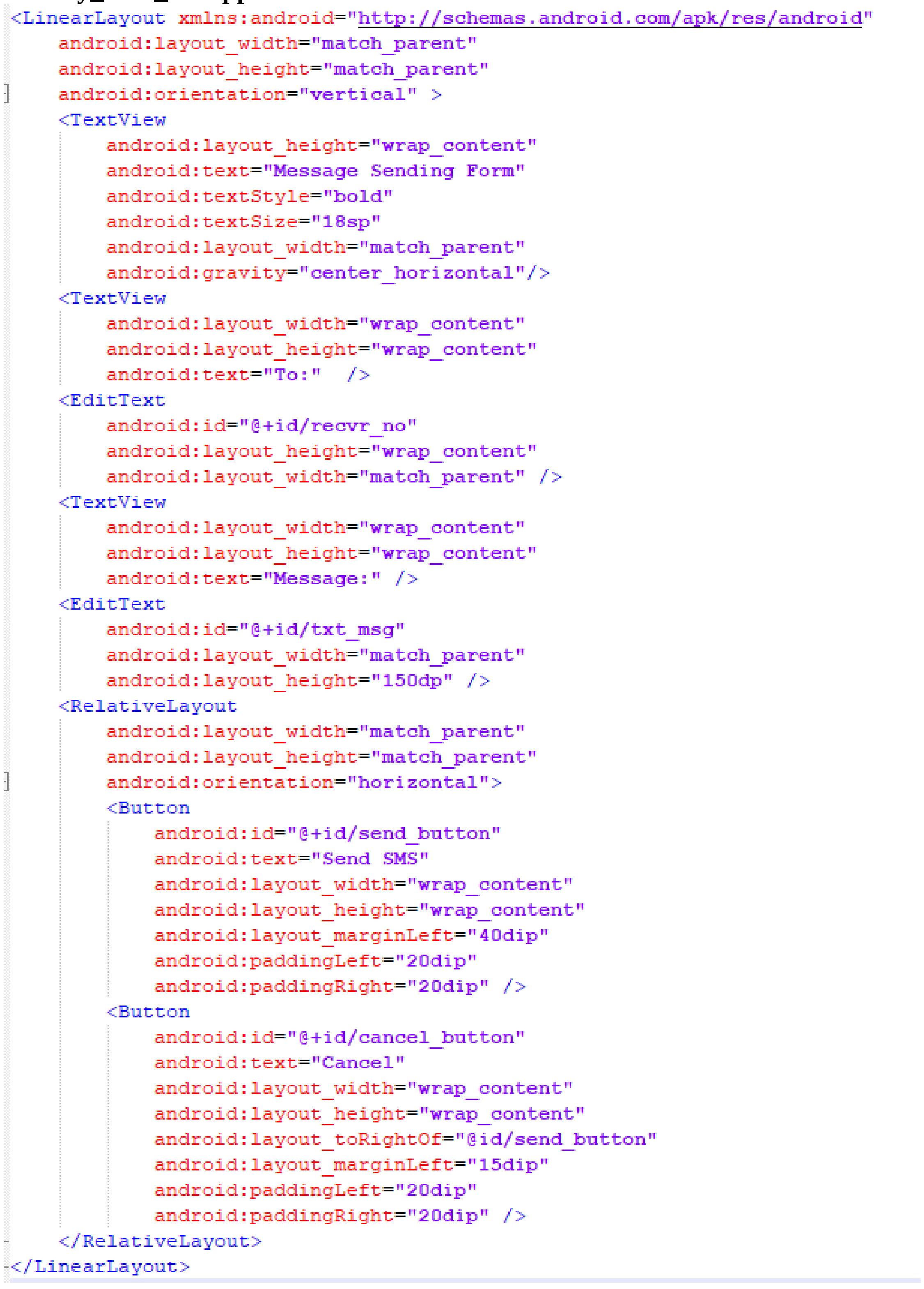
```
package com.androidunleashed.contentproviderapp;

import android.app.Activity;
import android.os.Bundle;
import android.net.Uri;
import android.database.Cursor;
import android.content.CursorLoader;
import android.provider.ContactsContract;
import android.widget.ListView;
import java.util.ArrayList;
import android.widget.ArrayAdapter;

public class ContentProviderAppActivity extends Activity {
   ArrayList<String> contactRows=new ArrayList<String>();
   final String[] nocontact={"No Contacts on the Device"};

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_content_provider_app);
      final ListView contactsList=(ListView) findViewById(R.id.contactslist);
      Uri contactsUri = Uri.parse("content://contacts/people");
      String[] projection = new String[] {ContactsContract.Contacts._ID,
ContactsContract.Contacts.DISPLAY_NAME };
      Cursor c;
      CursorLoader cursorLoader = new CursorLoader(this, contactsUri, projection, null, null , null);
```

```
        c = cursorLoader.loadInBackground();
        contactRows.clear();
        c.moveToFirst();
        while(c.isAfterLast()==false){
           String contactID = c.getString(c.getColumnIndex(ContactsContract.Contacts._ID));
           String contactDisplayName =
c.getString(c.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
           contactRows.add(contactID+ " "+contactDisplayName);
           c.moveToNext();
        }
        if (c != null && !c.isClosed()) {
           c.close();
        }
        if(contactRows.isEmpty()) {
           ArrayAdapter<String> arrayAdpt=new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, nocontact);
           contactsList.setAdapter(arrayAdpt);
        }
        else {
           ArrayAdapter<String> arrayAdpt=new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, contactRows);
           contactsList.setAdapter(arrayAdpt);
        }
    }
}
```
We define a contactsuri URI for the Contacts provider. Thereafter, a projection String array is defined to specify the columns that we want to extract from the contacts database. With the help of a cursorLoader, we load rows from the Contacts provider and assign them to the cursor c. Thereafter, through a while loop, the information in the cursor is extracted. Because we want to display only the ID and contact name, the informationn the contactsContract.Contacts. ID and ContactsContract.Contacts. DISPLAY_NAME Columns is accessed and assigned to the contactRows ArrayList. If contactRows is not empty, an ArrayAdapter object called arrayAdpt is defined through it. Finally, a ListView control is filled with the information in the arrayaAdpt ArrayAdapter. To access information in the Contacts Provider in our app, we need to add the following permission into the AndroidManifest .xml file:
**<uses-permission android:name="android.permission.READ_CONTACTS"/>**

**9. Explain the process of sending an SMS with an example?**
Create a new Android project called sendsmMsapp. The first step is to design the user interface for sending messages via SMS. The user interface consists of three TextView, two EditText, and two Button controls. One of the Text View controls is for displaying the title of the screen, Message Sending Form. The other two TextView controls are used on the left of the EditText controls to display text that tells the user what has to be entered in the EditText controls. These two TextView controls are used to display To: and Message:. The two EditText controls are used to enter the phone number of the recipient and the message to be sent.

**Getting Permission to Send SMS Messages**
To send and receive SMS messages in an application, we need to use permissions. Add permissions to the element of Our AndroidManifest.xml file:
**AndroidManifest.xml**
**<uses-permission android:name="android.permission.SEND_SMS"/>**

**Activity_send_smsapp.xml**



```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/txt_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="20dip"
            android:paddingRight="20dip" />
        <Button
            android:id="@+id/cancel_button"
            android:text="Cancel"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toRightOf="@id/send_button"
            android:layout_marginLeft="15dip"
            android:paddingLeft="20dip"
            android:paddingRight="20dip" />
    </RelativeLayout>
</LinearLayout>
```

**SendSMSAppActivity.java**

```java
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 SendSMSAppActivity extends Activity {
    EditText phoneNumber, message;
    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 sendBtn = (Button) this.findViewById(R.id.send_button);
        sendBtn.setOnClickListener(new Button.OnClickListener(){
            public void onClick(View v) {
                phoneNumber = (EditText) findViewById(R.id.recvr_no);
                message = (EditText) findViewById(R.id.txt_msg);
                if(phoneNumber.getText().toString().trim().length() >0 && message.getText().toString().trim().length() >0) {
                    SmsManager sms = SmsManager.getDefault();
                    sms.sendTextMessage(phoneNumber.getText().toString(), null,
                    message.getText().toString(), sentPendIntent, delivered_pendintnet);
                }
                else {
                    Toast.makeText(SendSMSAppActivity.this, "Either phone number or text is  missing", Toast.LENGTH_SHORT).show();
                }
            }
        });
        Button cancelBtn = (Button) this.findViewById(R.id.cancel_button);
        cancelBtn.setOnClickListener(new Button.OnClickListener(){
            public void onClick(View v) {
                phoneNumber.setText("");
                message.setText("");
            }
        });
    }
}
```

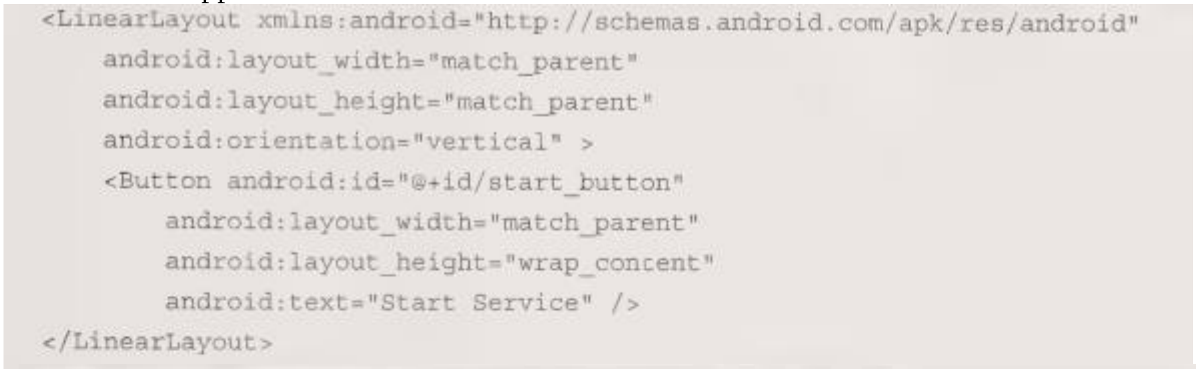
To send an SMS message with Java code, we use the SmsManager Class. We cannot instantiate this class directly and must call the getDefault () static method to create its object. The method provided by the smsManager Class for sending SMS messages is the sendText - Message‘) method. The syntax for the sendTextMessage() method is
sendTextMessage (recipient phoneno, service _centadd, sms_msg, sent_intent, delivery_intent)
where recipient _phoneno is the recipient’s phone number, and service_centadd is the Service center address. We use the value nul11 for the default smMsc (Short Message Service Center). The sms_msg is the text message of the SMS, sent Intent is the pending
Intent to invoke when the message is sent, and delivery_Intent is the pending Intent to invoke when the message is delivered. To monitor the status of the SMS message and confirm if it was sent correctly, we create two PendingIntent objects that are then passed as arguments to the sendTextMessage () method. The two PendingIntent objects are created with the following statements:
String SENT = "SMS SENT";
String DELIVERED = "SMS DELIVERED";
final PendingIntent sentPendIntent = PendingIntent.getBroadcast (this, 0, new Intent (SENT), 0);
final PendingIntent delivered pendintnet = PendingIntent.getBroadcast (this, 0, new Intent (DELIVERED), 0);
The two PendingIntent objects are passed into the last two arguments of the sendText - Message () method:
SmsManager sms = SmsManager.getDefault () ;
sms .sendTextMessage (phoneNumber.getText () .toString(), null, message.getText(). ToString (), sentPendIntent, delivered_pendintnet);

We are informed, via the two PendingIntent objects, whether the message was successfully sent, delivered, or failed. The SmsManager fires SENT and DELIVERED when the SMS messages are sent and delivered. The two PendingIntent objects are used to send broadcasts when an SMS message is sent or delivered. We also create and register two BroadcastReceivers, which listen for Intents that match SENT and DELIVERED as shown by the following statements:
registerReceiver(sentReceiver, new IntentFilter(SENT) ) ;
registerReceiver (deliveredReceiver, new IntentFilter (DELIVERED) ) ;

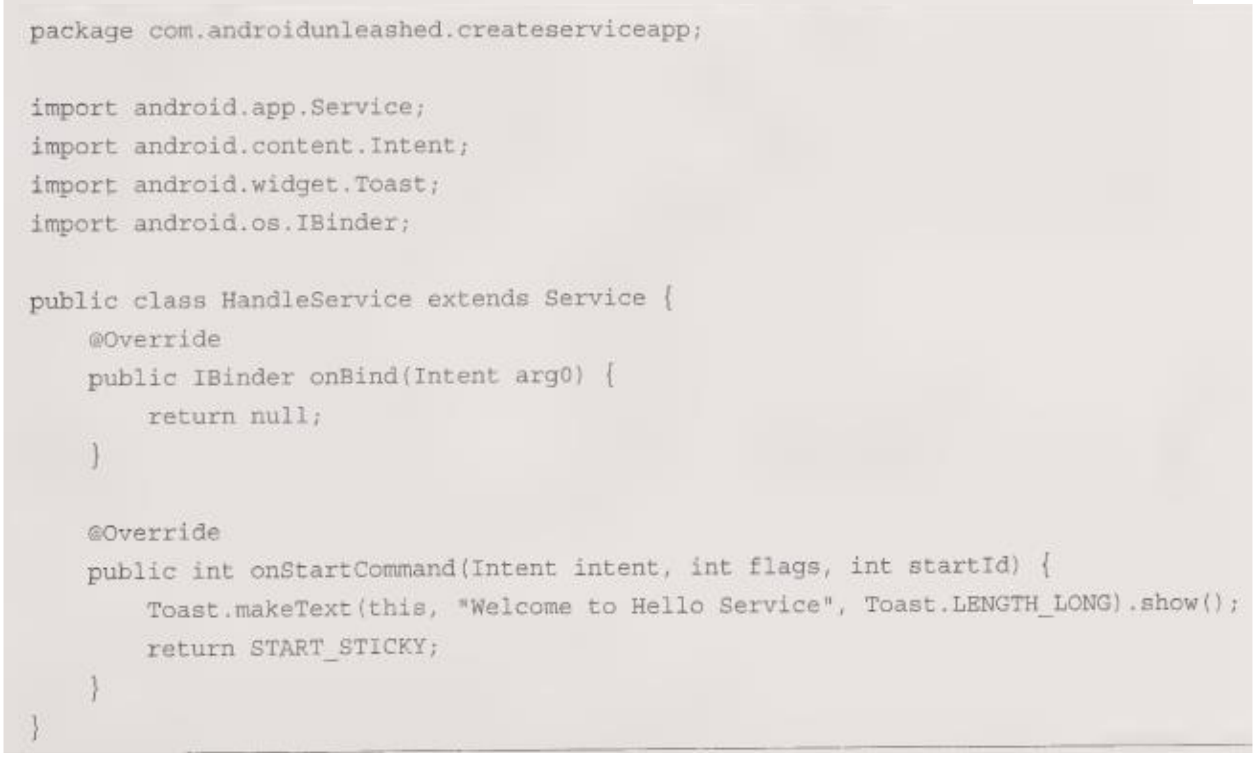
**10. What is a service? What are the different methods use to create a service? Explain with an example.**

A service is an application component that performs the desired task without providing a user interface. Hence, services can only be accessed from another application and never invoked by a user. A service does not create its own thread. Instead it runs in the main thread of the application. So, to maintain efficiency of an application, we should create a new thread within the service to perform its processing tasks.
To create a service, define a class that extends the Service base class. All services extend the Service class. The methods that we can implement in the class are
onBind()—The method returns an IBinder object that enables an activity to use a Service that can directly access members and methods inside it.
onStartCommand()—Called when the Service starts. The method is used to check whether any data is passed to the Service for processing. We can also use this method to return a constant that configures the service. For example, we can make the method return the constant start_sticky, which makes the Service run until it is explicitly stopped.
onDestroy()—Called when the Service is stopped using the stopService() method.
With this method, we release resources consumed by the Service.
Following arethe methods used for starting, stopping, binding, and unbinding from the service.
To start a Service, use the startService() method. The following example starts the Service represented by HandleService.class file:
startService(new Intent (getBaseContext (), HandleService.class) ;
To start a Service from an external application, we need to define its complete package name. The following example starts the HandleService.class Service from an external application:
startService (new Intent ("com.androidunleashed.handleservice") );
To stop a Service, use the stopService() method. The following example stops the HandleService.class Service:

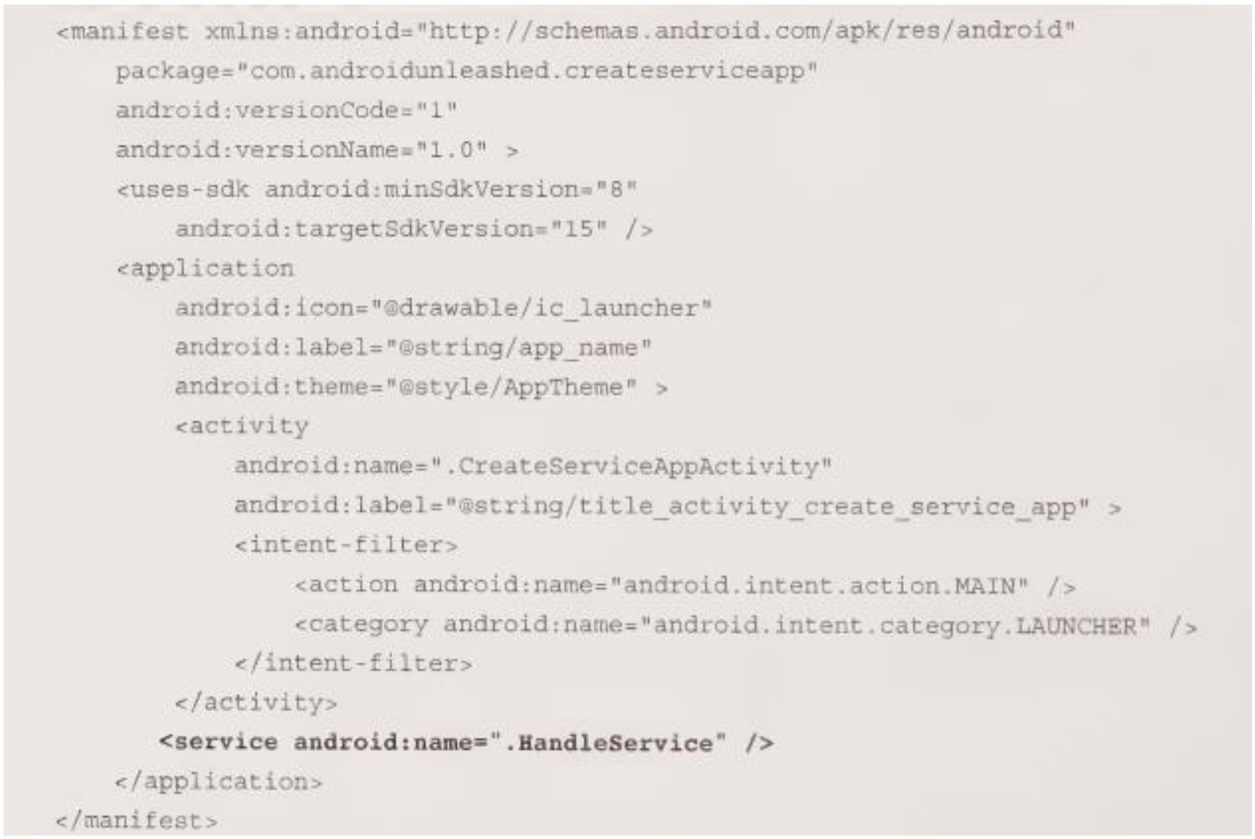
stopService (new Intent (getBaseContext(), HandleService.class) );
A client can bind to the Service with the bindservice() method. The following example binds the client to the HandleService.class Service using the ServiceConnection:
bindService (new Intent (this, HandleService.class), servConn, Context.BIND AUTO CREATE) '
The client(s) can be unbound from the Service with the unbindService() method.
The following example unbinds the service that is connected through the servconn ServiceConnection:
unbindService (servConn) ;
To understand how to use a started Service, we create a new Android project called CreateServiceApp.



```
<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/start_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_concent"
        android:text="Start Service" />
</LinearLayout>
```

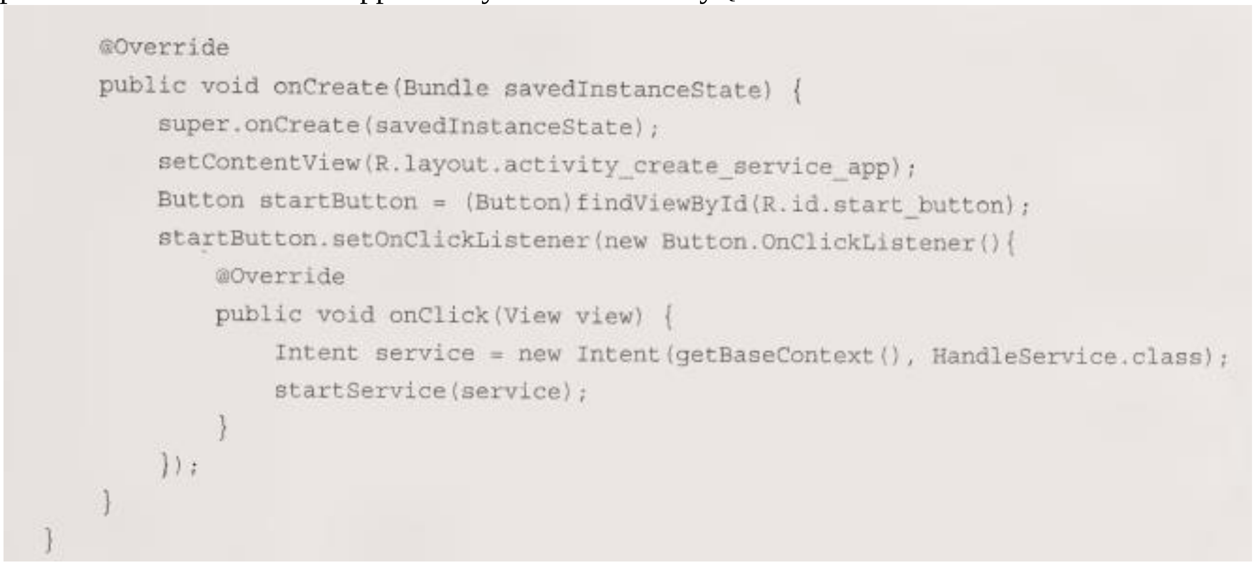


```
package com.androidunleashed.createserviceapp;

import android.app.Service;
import android.content.Intent;
import android.widget.Toast;
import android.os.IBinder;

public class HandleService extends Service {
    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Toast.makeText(this, "Welcome to Hello Service", Toast.LENGTH_LONG).show();
        return START_STICKY;
    }
}
```



```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidunleashed.createserviceapp"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="8"
        android:targetSdkVersion="15" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".CreateServiceAppActivity"
            android:label="@string/title_activity_create_service_app" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
       <service android:name=".HandleService" />
    </application>
</manifest>
```

public class CreateServiceAppActivity extends Activity {



```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_service_app);
        Button startButton = (Button)findViewById(R.id.start_button);
        startButton.setOnClickListener(new Button.OnClickListener(){
            @Override
            public void onClick(View view) {
                Intent service = new Intent(getBaseContext(), HandleService.class);
                startService(service);
            }
        });
    }
}
```