Question Bank for Mobile Application Development (MAD)

1. Enlist methods of Async Task in android

Ans:

doInBackground() : This method contains the code which needs to be executed in background. In this method we can send results multiple times to the UI thread by publishProgress() method. To notify that the background processing has been completed we just need to use the return statements

onPreExecute() : This method contains the code which is executed before the background processing starts

onPostExecute() : This method is called after doInBackground method completes processing. Result from doInBackground is passed to this method

onProgressUpdate() : This method receives progress updates from doInBackground method, which is published via publishProgress method, and this method can use this progress update to update the UI thread

2. Explain custom permission.

By defining custom permissions, an app can share its resources and capabilities with other apps. Apps can define their own custom permissions and request custom permissions from other apps by defining <uses-permission> elements.

3. Enlist the steps to publish Android application

Step 1: Create a Developer Account
Step 2: Link Your Merchant Account
Step 3: Create an App
Step 4: Prepare Store Listing
- Graphic Assets:
- Languages and Translations:
- Categorization:
- Contact Details:
- Privacy Policy:

Step 5: Upload APK to an App Release:
Step 6: Provide an Appropriate Content Rating
Step 7: Set Up Pricing & Distribution
Step 8: Rollout Release to Publish Our App

4. Define services in android.

A service is an application component which runs without direst interaction with the user in the background.
Services are used for repetitive and potentially long running operations, i.e., Internet downloads, checking for new data, data processing, updating content providers and the like.
Services run with a higher priority than inactive or invisible activities and therefore it is less likely that the Android system terminates them. Services can also be configured to be restarted if they get terminated by the Android system once sufficient system resources are available again.

5. Enlist the categories of sensors in android

Motion sensors

These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.

Environmental sensors

These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.

Position sensors

These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

6. **Define SQLite**.

SQLite is an embedded, relational database management system (RDBMS).
SQLite is referred to as embedded because it is provided in the form of a library that is linked into applications.

7. State the use of Fragments in Android Application Development.

A fragment is a self-contained, modular section of an application's user interface and corresponding behavior that can be embedded within an activity. Fragments can be assembled to create an activity during the application design phase, and added to or removed from an activity during application runtime to create a dynamically changing user interface.

8. Define signing of Android Application.

Application signing allows developers to identify the author of the application and to update their application without creating complicated interfaces and permissions. Every application that is run on the Android platform must be signed by the developer.

9. State the use of Content Provider

A Content Provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. A content provider can use different ways to store its data and the data can be stored in files, in a database or even over a network.

10. Enlist the steps to involved in creating Maps API Key

Step 1: Open Google developer console and signing with your gmail account: https://console.developers.google.com/project

Step 2: Now create new project. You can create new project by clicking on the Create Project button and give name to your project.
Step 3: Now click on APIs & Services and open Dashboard from it.
Step 4: In this open Enable APIS AND SERICES.
Step 5: Now open Google Map Android API.

Step 6: Now go to Credentials

Step 7: Here click on Create credentials and choose API key

Step 8: Now API your API key will be generated. Copy it and save it somewhere as we will need it when implementing Google Map in our Android project.

11. State use of Location based service in Android.

LBS app follows the location and offer extra services such as locating facilities close at hand, presenting suggestions for route planning and so on.

In Location based service application map is the key components, which present the visual location of our location.

12. Write syntax of Intent-filter tag.

Syntax:

```
<intent-filter     android:icon=”drawable resource”
                     android:label=”string resource”
                     android:priority=”integer”>
. . .
</intent-filter>
```

13. Define services in android.

A service is an application component which runs without direst interaction with the user in the background.
Services are used for repetitive and potentially long running operations, i.e., Internet downloads, checking for new data, data processing, updating content providers and the like.
Services run with a higher priority than inactive or invisible activities and therefore it is less likely that the Android system terminates them. Services can also be configured to be restarted if they get terminated by the Android system once sufficient system resources are available again.

14. Discuss Developer Console.

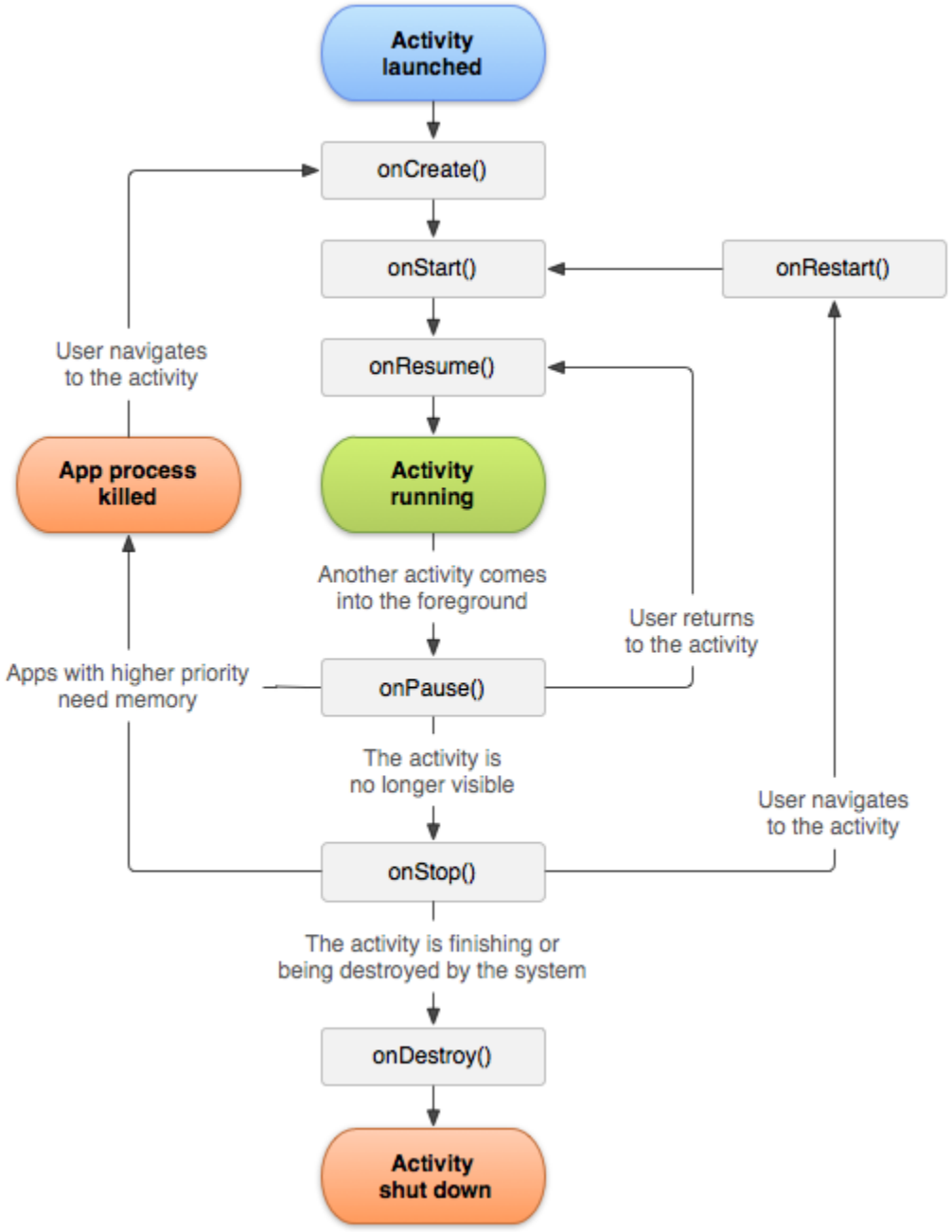
The Google Play Developer Console is your home for publishing operations and tools. You can manage all phases of publishing on Google Paly through the Developer Console from any web browser.

15. Write a program to display the maps.(only Java File)

```
import androidx.fragment.app.FragmentActivity;
import android.os.Bundle;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {

   private GoogleMap mMap;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_maps);

      SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
            .findFragmentById(R.id.map);
      mapFragment.getMapAsync(this);
   }


   @Override
   public void onMapReady(GoogleMap googleMap) {
      mMap = googleMap;
```

```
        LatLng sydney = new LatLng(-34, 151);
        mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    }
}
```

16. Explain Activity Life Cycle in details.



Activities have a predefined life-cycle and which certain methods are called. Following Table

important activity lifecycle methods:

Method	Purpose
onCreate()	Called then the activity is created. Used to initialize the activity, for example create the user interface.
onResume()	Called if the *activity* get visible again and the user starts interacting with the activity again. Used to initialize fields, register listeners, bind to services, etc.
onPause()	Called once another activity gets into the foreground. Always called before the *activity* is not visible anymore. Used to release resources or save application data. For example you unregister listeners, intent receivers, unbind from services or remove system service listeners.
onStop()	Called once the activity is no longer visible. Time or CPU intensive shut-down operations, such as writing information to a database should be down in the onStop() method. This method is guaranteed to be called as of API 11.

The flow of these methods is depicted in the diagram.

1. Activity States:

• The Android OS uses a priority queue to assist in managing activities running on the device. Based on the state a particular Android activity is in, it will be assigned a certain priority within the OS.

• This priority system helps Android identify activities that are no longer in use, allowing the OS to reclaim memory and resources.

• Fig. illustrates the states an activity can go through, during its lifetime:

• These states are often broken into three main teams as follows:

1. Active or Running:

• Activities are thought of active or running if they're within the foreground, additionally referred to as the top of the activity stack. this can be thought of the highets priority activity within the Android Activity stack, and as such only be killed by the OS in extreme things, like if the activity tries to use more memory than is available on the device as this might cause the UI to become unresponsive.

2. Paused:

• When the device goes to sleep, or an activity continues to be visible but partially hidden by a new, non-full-sized or clear activity, the activity is taken into account paused.

• Paused activities are still alive, that is, they maintain all state and member information, and stay attached to the window manager.

• This can be thought of to be the second highest priority activity within the android Activity stack and, as such, can solely be killed by the OS if killing this activity can satisfy the resource requirement needed to keep the Active/Running Activity stable and responsive.

3. Stopped:
• Activities that are utterly obscured by another activity are thought of stopped or within the background.
• Stopped activities still try and retain their state and member info for as long as possible but stopped activities are thought of to be loweat priority of the three states and, as such, the OS can kill activities during this state initial to satisfy the resource needs of higher priority activities.

17. Build a GUI for Student table with 5 records, write queries to add delete and view the records from the SQLite databases.

18. Explain SMS services in android application development

Android devices can send and receive messages to or from any other phone that supports Short Message Service (SMS). Android offers the Messenger app that can send and receive SMS messages.

Sending and receiving SMS messages

Access to the SMS features of an Android device is protected by user permissions. Just as our app needs the user's permission to use phone features, so also does an app need the user's permission to directly use SMS features.

We have two choices for sending SMS messages:
1. Use an implicit Intent to launch a messaging app such as Messenger, with the ACTION_SENDTO action.

- This is the simplest choice for sending messages. The user can add a picture or other attachment in the messaging app, if the messaging app supports adding attachments.
- Our app doesn't need code to request permission from the user.
- If the user has multiple SMS messaging apps installed on the Android phone, the App chooser will appear with a list of these apps, and the user can choose which one to use. (Android smartphones will have at least one, such as Messenger.)
- The user can change the message in the messaging app before sending it.
- The user navigates back to our app using the Back button.

2. Send the SMS message using the sendTextMessage() method or other methods of the SmsManager class.

- This is a good choice for sending messages from our app without having to use another installed app. Our code must ask the user for permission before sending the message if the user hasn't already granted permission.
- The user stays in our app during and after sending the message.
- We can manage SMS operations such as dividing a message into fragments, sending a multipart message, get carrier-dependent configuration values, and so on.

To receive SMS messages, the best practice is to use the onReceive() method of the BroadcastReceiver class. The Android framework sends out system broadcasts of events such as

receiving an SMS message, containing intents that are meant to be received using a BroadcastReceiver.
• Our app receives SMS messages by listening for the SMS_RECEIVED_ACTION broadcast. Most smartphones and mobile phones support what is known as "PDU mode" for sending and receiving SMS.
• PDU (Protocol Data Unit) contains not only the SMS message, but also metadata about the SMS message, such as text encoding, the sender, SMS service center address, and much more. To access this metadata, SMS apps almost always use PDUs to encode the contents of a SMS message.
• The sendTextMessage() and sendMultimediaMessage() methods of the SmsManager class encode the contents for we. When receiving a PDU, we can create an SmsMessage object from the raw PDU using createFromPdu().

19. Develop a program for sending email.

Activity_main.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/linearLayout1"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <TextView
      android:id="@+id/textViewPhoneNo"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="To : "
      android:textAppearance="?android:attr/textAppearanceLarge" />

   <EditText
      android:id="@+id/editTextTo"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:inputType="textEmailAddress" >

      <requestFocus />

   </EditText>

   <TextView
      android:id="@+id/textViewSubject"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Subject : "
      android:textAppearance="?android:attr/textAppearanceLarge" />
```

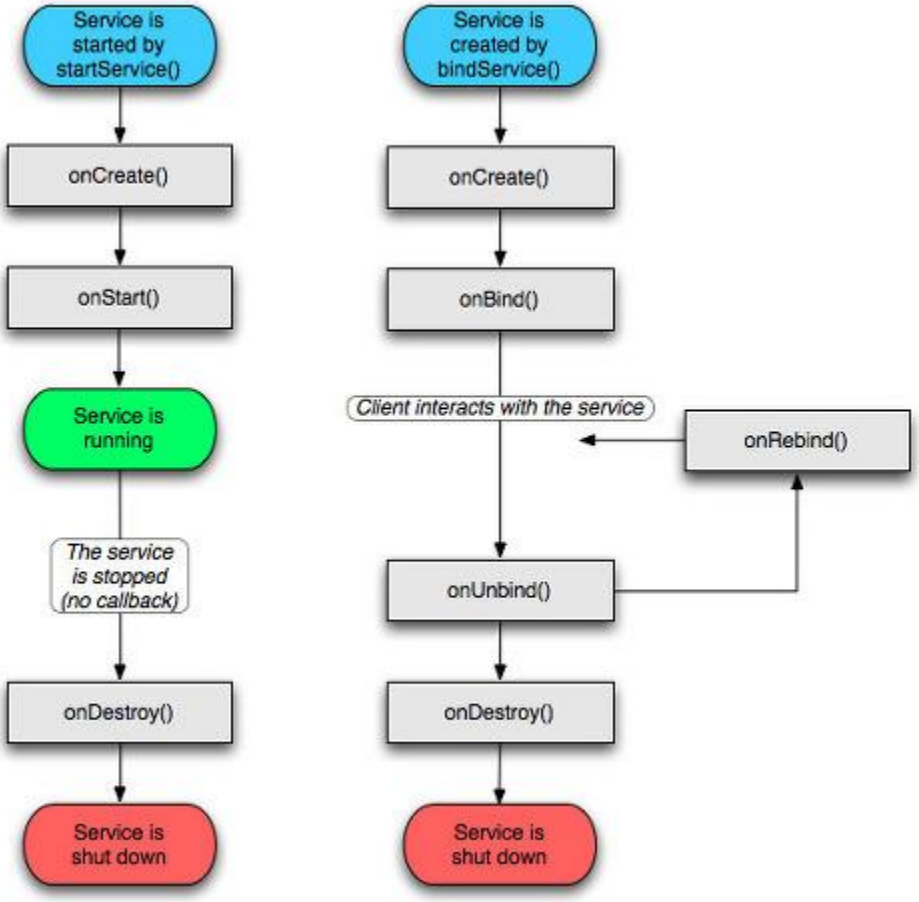
```
    <EditText
        android:id="@+id/editTextSubject"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
         >
    </EditText>

    <TextView
        android:id="@+id/textViewMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Message : "
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <EditText
        android:id="@+id/editTextMessage"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="top"
        android:inputType="textMultiLine"
        android:lines="5" />

    <Button
        android:id="@+id/buttonSend"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Send" />

</LinearLayout>
```

MainActivity.Java

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

public class SendEmailActivity extends Activity {

        Button buttonSend;
        EditText textTo;
        EditText textSubject;
        EditText textMessage;

        @Override
```

```
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.main);

		buttonSend = (Button) findViewById(R.id.buttonSend);
		textTo = (EditText) findViewById(R.id.editTextTo);
		textSubject = (EditText) findViewById(R.id.editTextSubject);
		textMessage = (EditText) findViewById(R.id.editTextMessage);

		buttonSend.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {

			  String to = textTo.getText().toString();
			  String subject = textSubject.getText().toString();
			  String message = textMessage.getText().toString();

			  Intent email = new Intent(Intent.ACTION_SEND);
			  email.putExtra(Intent.EXTRA_EMAIL, new String[]{ to});
			  //email.putExtra(Intent.EXTRA_CC, new String[]{ to});
			  //email.putExtra(Intent.EXTRA_BCC, new String[]{to});
			  email.putExtra(Intent.EXTRA_SUBJECT, subject);
			  email.putExtra(Intent.EXTRA_TEXT, message);

			  //need this to prompts email client only
			  email.setType("message/rfc822");

			  startActivity(Intent.createChooser(email, "Choose an Email client :"));

			}
		});
	}
}
```

20. Explain Service Life Cycle in details.



You just need to call either *startService*() or *bindService*() from any of your android components. Based on how your service was started it will either be "started" or "bound"

- **Started**

A service is **started** when an application component, such as an activity, starts it by calling *startService()*. Now the service can run in the background indefinitely, even if the component that started it is destroyed.

- **Bound**

A service is **bound** when an application component binds to it by calling *bindService()*. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

Like any other components service also has callback methods. These will be invoked while the service is running to inform the application of its state. Implementing these in your custom service would help you in performing the right operation in the right state.

There is always only a single instance of service running in the app. If you are calling startService() for a single service multiple times in your application it just invokes the onStartCommand() on that service. Neither is the service restarted multiple times nor are its multiple instances created

onCreate()

This is the first callback which will be invoked when any component starts the service. If the same service is called again while it is still running this method wont be invoked. Ideally one time setup and intializing should be done in this callback.

onStartCommand()

This callback is invoked when service is started by any component by calling startService(). It basically indicates that the service has started and can now run indefinetly. Now its your responsibilty to stop the service. This callback also has a int return type. This return type describes what happens to the service once it is destroyed by the system.There are three possible return types

- **START_STICKY**

Returning this indicates that once the service is killed by the system it will be recreated and onStartCommand method will be invoked again.But the previous intent is not redelivered. Instead *onStartCommand* is called with a null intent

- **START_NOT_STICKY**

Returning this indicates that the service wont be recreated if it is killed by the system

- **START_REDELIVER_INTENT**

Returning this indicates that once the service is killed by the system it will be recreated and *onStartCommand* method will be invoked again.But here the original intent is redelivered again.

onBind()

This is invoked when any component starts the service by calling *onBind*. Basically the component has now binded with the service. This method needs to return your implementation of *IBinder* which will be used for Interprocess Communication. If you dont want your service to bind with any component you should just return null nevertheless this method needs to be implemented

onUnbind()

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

onRebind()

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

onDestroy()

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

receivers. But very rarely it happens that the service is destroyed with onDestroy not being invoked

21. Write a program to capture an image using camera and display it.

Xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="10dp"
   android:paddingRight="10dp">
   <Button
      android:id="@+id/btnTakePicture"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Take a Photo"
      android:textStyle="bold"
      android:layout_centerHorizontal="true"
      android:layout_alignParentBottom="true" />
   <ImageView
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:id="@+id/capturedImage"
      android:layout_above="@+id/btnTakePicture"/>
</RelativeLayout>
```

Java File:

```
import android.content.Intent;
import android.graphics.Bitmap;
import android.provider.MediaStore;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
   private Button btnCapture;
```

```
    private ImageView imgCapture;
    private static final int Image_Capture_Code = 1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btnCapture =(Button)findViewById(R.id.btnTakePicture);
        imgCapture = (ImageView) findViewById(R.id.capturedImage);
        btnCapture.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent cInt = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
                startActivityForResult(cInt,Image_Capture_Code);
            }
        });
    }
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == Image_Capture_Code) {
            if (resultCode == RESULT_OK) {
                Bitmap bp = (Bitmap) data.getExtras().get("data");
                imgCapture.setImageBitmap(bp);
            } else if (resultCode == RESULT_CANCELED) {
                Toast.makeText(this, "Cancelled", Toast.LENGTH_LONG).show();
            }
        }
    }
}
```

22. Explain zoom control (IN/OUT) with help of an example.

We can display Google Maps in our Android application. We can put the map to any preferred location. On the emulator there is no way to zoom in or out from a particular location, but it can possible on a real Android device where we can touch that map to zoom it. Hence we should learn how users can zoom in or out of the map using the built-in zoom controls.

main.xml file

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

    android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent">

    <com.google.android.maps.MapView
        android:id="@+id/mapView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:enabled="true"
        android:clickable="true"
        android:apiKey="0l4sCTTyRmXTNo7k8DREHvEaLar2UmHGwnhZVHQ"
        />

    <LinearLayout android:id="@+id/zoom"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        />

</RelativeLayout>
```

MapsActivity.java file

```
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapView;
import com.google.android.maps.MapView.LayoutParams;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.LinearLayout;

public class MapsActivity extends MapActivity
{
    MapView mapView;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);


        mapView = (MapView) findViewById(R.id.mapView);
        LinearLayout zoomLayout = (LinearLayout)findViewById(R.id.zoom);
        View zoomView = mapView.getZoomControls();

        zoomLayout.addView(zoomView,
            new LinearLayout.LayoutParams(
                LayoutParams.WRAP_CONTENT,
                LayoutParams.WRAP_CONTENT));
        mapView.displayZoomControls(true);
```

```
    }

    @Override
    protected boolean isRouteDisplayed() {
        // TODO Auto-generated method stub
        return false;
    }
}
```

23. Develop an application to send and receive SMS. (Write only .java and permission tag in Manifest file)

MainActivity.java

```
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.ActionBarActivity;
import android.view.View;
public class MainActivity extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void goToInbox(View view) {
        Intent intent = new Intent(MainActivity.this, ReceiveSmsActivity.class);
        startActivity(intent);
    }

    public void goToCompose(View view) {
        Intent intent = new Intent(MainActivity.this, SendSmsActivity.class);
        startActivity(intent);
    }
```

}

SendSmsActivity.java

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class SendSmsActivity extends Activity {

   Button sendSMSBtn;
   EditText toPhoneNumberET;
   EditText smsMessageET;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_send_sms);
      sendSMSBtn = (Button) findViewById(R.id.btnSendSMS);
      toPhoneNumberET = (EditText) findViewById(R.id.editTextPhoneNo);
      smsMessageET = (EditText) findViewById(R.id.editTextSMS);
      sendSMSBtn.setOnClickListener(new View.OnClickListener() {
         public void onClick(View view) {
            sendSMS();
         }
      });
   }

   protected void sendSMS() {
      String toPhoneNumber = toPhoneNumberET.getText().toString();
      String smsMessage = smsMessageET.getText().toString();
      try {
         SmsManager smsManager = SmsManager.getDefault();
         smsManager.sendTextMessage(toPhoneNumber, null, smsMessage, null, null);
         Toast.makeText(getApplicationContext(), "SMS sent.",
               Toast.LENGTH_LONG).show();
      } catch (Exception e) {
         Toast.makeText(getApplicationContext(),
               "Sending SMS failed.",
               Toast.LENGTH_LONG).show();
         e.printStackTrace();
      }
   }

   public void goToInbox(View view) {
```

```
        Intent intent = new Intent(SendSmsActivity.this, ReceiveSmsActivity.class);
        startActivity(intent);
    }
}
```

ReceiveSmsActivity.java

```
import android.app.Activity;
import android.content.ContentResolver;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;

public class ReceiveSmsActivity extends Activity implements OnItemClickListener {

    private static ReceiveSmsActivity inst;
    ArrayList<String> smsMessagesList = new ArrayList<String>();
    ListView smsListView;
    ArrayAdapter arrayAdapter;

    public static ReceiveSmsActivity instance() {
        return inst;
    }

    @Override
    public void onStart() {
        super.onStart();
        inst = this;
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_receive_sms);
        smsListView = (ListView) findViewById(R.id.SMSList);
        arrayAdapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
smsMessagesList);
        smsListView.setAdapter(arrayAdapter);
        smsListView.setOnItemClickListener(this);
```

```
        refreshSmsInbox();
    }

    public void refreshSmsInbox() {
        ContentResolver contentResolver = getContentResolver();
        Cursor smsInboxCursor = contentResolver.query(Uri.parse("content://sms/inbox"), null, null, null,
null);
        int indexBody = smsInboxCursor.getColumnIndex("body");
        int indexAddress = smsInboxCursor.getColumnIndex("address");
        long timeMillis = smsInboxCursor.getColumnIndex("date");
        Date date = new Date(timeMillis);
        SimpleDateFormat format = new SimpleDateFormat("dd/MM/yy");
        String dateText = format.format(date);

        if (indexBody < 0 || !smsInboxCursor.moveToFirst()) return;
        arrayAdapter.clear();
        do {
            String str = smsInboxCursor.getString(indexAddress) +" at "+
                    "\n" + smsInboxCursor.getString(indexBody) +dateText+ "\n";
            arrayAdapter.add(str);
        } while (smsInboxCursor.moveToNext());
    }

    public void updateList(final String smsMessage) {
        arrayAdapter.insert(smsMessage, 0);
        arrayAdapter.notifyDataSetChanged();
    }

    public void onItemClick(AdapterView<?> parent, View view, int pos, long id) {
        try {
            String[] smsMessages = smsMessagesList.get(pos).split("\n");
            String address = smsMessages[0];
            String smsMessage = "";
            for (int i = 1; i < smsMessages.length; ++i) {
                smsMessage += smsMessages[i];
            }

            String smsMessageStr = address + "\n";
            smsMessageStr += smsMessage;
            Toast.makeText(this, smsMessageStr, Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void goToCompose(View view) {
        Intent intent = new Intent(ReceiveSmsActivity.this, SendSmsActivity.class);
```

```
      startActivity(intent);
   }
}
```

smsReceiverBroadast.java

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage;
import android.widget.Toast;

import java.text.SimpleDateFormat;
import java.util.Date;

public class SmsBroadcastReceiver extends BroadcastReceiver {

   public static final String SMS_BUNDLE = "pdus";

   public void onReceive(Context context, Intent intent) {
      Bundle intentExtras = intent.getExtras();
      if (intentExtras != null) {
         Object[] sms = (Object[]) intentExtras.get(SMS_BUNDLE);
         String smsMessageStr = "";
         for (int i = 0; i < sms.length; ++i) {
            SmsMessage smsMessage = SmsMessage.createFromPdu((byte[]) sms[i]);

            String smsBody = smsMessage.getMessageBody().toString();
            String address = smsMessage.getOriginatingAddress();
            long timeMillis = smsMessage.getTimestampMillis();

            Date date = new Date(timeMillis);
            SimpleDateFormat format = new SimpleDateFormat("dd/MM/yy");
            String dateText = format.format(date);

            smsMessageStr += address +" at "+"\t"+ dateText + "\n";
            smsMessageStr += smsBody + "\n";
         }
         Toast.makeText(context, smsMessageStr, Toast.LENGTH_SHORT).show();

         //this will update the UI with message
         ReceiveSmsActivity inst = ReceiveSmsActivity.instance();
         inst.updateList(smsMessageStr);
      }
   }
}
```

Manifest file:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.javapapers.android.androidsmsapp">

   <uses-permission android:name="android.permission.WRITE_SMS" />
   <uses-permission android:name="android.permission.READ_SMS" />
   <uses-permission android:name="android.permission.RECEIVE_SMS" />
   <uses-permission android:name="android.permission.SEND_SMS" />
   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme">
      <activity
         android:name=".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>
      <activity
         android:name=".ReceiveSmsActivity"
         android:label="@string/app_name"></activity>
      <activity
         android:name=".SendSmsActivity"
         android:label="@string/app_name"></activity>
      <receiver
         android:name=".SmsBroadcastReceiver"
         android:exported="true">
         <intent-filter android:priority="999">
            <action android:name="android.provider.Telephony.SMS_RECEIVED" />
         </intent-filter>
      </receiver>

   </application>

</manifest>
```

24. How to create and connect SQLite? Explain with example.

In android, by using **SQLiteOpenHelper** class we can easily create the required database and tables for our application. To use **SQLiteOpenHelper**, we need to create a subclass that overrides the **onCreate()** and **onUpgrade()** call-back methods.

Following is the code snippet of creating the database and tables using the **SQLiteOpenHelper** class in our android application.

.

```
public class DbHandler extends SQLiteOpenHelper {
   private static final int DB_VERSION = 1;
   private static final String DB_NAME = "usersdb";
   private static final String TABLE_Users = "userdetails";
   private static final String KEY_ID = "id";
   private static final String KEY_NAME = "name";
   private static final String KEY_LOC = "location";
   private static final String KEY_DESG = "designation";
   public DbHandler(Context context){
      super(context,DB_NAME, null, DB_VERSION);
   }
   @Override
   public void onCreate(SQLiteDatabase db){
      String CREATE_TABLE = "CREATE TABLE " + TABLE_Users + "("
            + KEY_ID + " INTEGER PRIMARY KEY AUTOINCREMENT," + KEY_NAME + " TEXT,"
            + KEY_LOC + " TEXT,"
            + KEY_DESG + " TEXT"+ ")";
      db.execSQL(CREATE_TABLE);
   }
   @Override
   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
      // Drop older table if exist
      db.execSQL("DROP TABLE IF EXISTS " + TABLE_Users);
      // Create tables again
      onCreate(db);
   }
}
```

26) How to monitor the location in android explain with example

- One feature of the LocationManager class is its ability to monitor a specific location. This is achieved using the addProximityAlert() method.
- The following code snippet shows how to monitor a particular location such that if the user is within a five-meter radius from that location, your application will fire an intent to launch the web browser:

```
import android.app.PendingIntent;
import android.content.Intent;
import android.net.Uri;
 //---use the LocationManager class to obtain locations data---
 lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
 //---PendingIntent to launch activity if the user is within
 // some locations---
 PendingIntent pendingIntent = PendingIntent.getActivity(  this, 0, new
Intent(android.content.Intent.ACTION_VIEW, Uri.parse("http://www.amazon.com")), 0);
 lm.addProximityAlert(37.422006, -122.084095, 5, -1, pendingIntent);
```

The addProximityAlert() method takes five arguments:

➢ Latitude

➢ Longitude

➢ Radius (in meters)

➢➢Expiration (duration for which the proximity alert is valid, after which it is deleted; –1 for no expiration)

➢➢Pending intent

Note that if the Android device's screen goes to sleep, the proximity is also checked once every four minutes in order to preserve the battery life of the device.