Dadabhoy Institute of Higher Education

Registration No: - BSCS/3-17/M03028

Student Name: - OSAMA ALI KHAN

Title of Assignment: - final Term Examination

Title of Course: - Mobile Programming

Name of Faculty Member: -: Dr. Arslan Khan

Submission Date: - 26/09/2020

Question number 1:-

Programming interface Level is a whole number worth that remarkably recognizes the structure Programming interface amendment offered by a form of the Android stage.

The Android stage gives a structure Programming interface that applications can use to communicate with the basic Android framework. The system Programming interface comprises of:

A center arrangement of bundles and classes

A lot of XML components and characteristics for pronouncing a show record

A lot of XML components and characteristics for pronouncing and getting to assets

A lot of Purposes

A lot of consents that applications can ask for, just as authorization implementations remembered for the framework

Each progressive adaptation of the Android stage can incorporate updates to the Android application structure Programming interface that it conveys.

Updates to the system Programming interface are planned so that the new Programming interface stays viable with prior renditions of the Programming interface. That is, most changes in the Programming interface are added substance and present new or substitution usefulness. As parts of the Programming interface are overhauled, the more seasoned supplanted parts are deplored yet are not eliminated, so existing applications can at present use them. In an extremely modest number of cases, portions of the Programming interface might be adjusted or taken out, albeit normally such changes are just expected to guarantee Programming interface strength and application or framework security. All other Programming interface parts from prior updates are conveyed forward without change.

The system Programming interface that an Android stage conveys is determined utilizing a whole number identifier called "Programming interface Level". Every Android stage form bolsters precisely one Programming interface Level, in spite of the fact that help is certain for all previous Programming interface Levels (down to Programming interface Level 1). The underlying arrival of the Android stage gave Programming interface Level 1 and ensuing deliveries have increased the Programming interface Level.

The table beneath indicates the Programming interface Level upheld by every variant of the Android stage. For data about the general quantities of gadgets that are running every variant, see the

android API Level:-

Platform Version	API Level
Android 10.0	29
Android 9	28
Android 8.1	27
Android 8.0	26
Android 7.1.1	
Android 7.1	25
Android 7.0	24
Android 6.0	23
Android 5.1	22
Android 5.0	21
Android 4.4W	20
Android 4.4	19
Android 4.3	18
Android 4.2, 4.2.2	17
Android 4.1, 4.1.1	16
Android 4.0.3, 4.0.4	15
Android 4.0, 4.0.1, 4.0.2	14
Android 3.2	13
Android 3.1.x	12
Android 3.0.x	11
Android 2.3.4	
Android 2.3.3	10
Android 2.3.2	
Android 2.3.1	
Android 2.3	9
Android 2.2.x	8
Android 2.1.x	7
Android 2.0.1	6
Android 2.0	5
Android 1.6	4
Android 1.5	3
Android 1.1	2
Android 1.0	1

Question number 2:-

An assistance is an application part used to perform long running errands in foundation.

A help doesn't have any UI and neither can it straightforwardly convey to a movement.

A help can run in the foundation inconclusively, regardless of whether the part that began the administration is pulverized.

Generally a help consistently plays out a solitary activity and stops itself once the planned undertaking is finished.

Bound Service

Administration which call uncertainly in the middle of action. An Android segment may tie itself to a Service utilizing bindservice (). A bound help would run as long as the other application segments are bound to it. When they unbind, the administration crushes itself.

Unbound Service

For this situation, an application part begins the administration, and it would keep on running out of sight, regardless of whether the first segment that started it is pulverized. For example, when begun, a help would keep on playing music in the foundation uncertainly.

Android Service Lifecycle

An assistance can be controlled by the framework, If somebody calls Context.startService() or bindService() strategy.

onStartCommand()

This technique is considered when the administration be begun, by calling startService(). When this technique executes, the administration is begun and can run in the foundation uncertainly. In the event that you actualize this, it is your obligation to stop the administration when its work is done, by calling stopSelf() or stopService(). On the off chance that you are characterizing your administration as, limited help hen you don't have to execute this technique.

onBind()

You have to supersede this technique, just in the event that you are characterizing your administration as limited help. This strategy is called, when another part needs to tie with the administration by calling bindService(). In your execution of this technique, you should give an interface that customers use to speak with the administration, by restoring an IBinder. You should consistently execute this technique, yet in the event that you would prefer not to permit official, at that point you should bring invalid back.

onCreate()

This technique is called while the administration is first made. Here all the administration introduction is finished. This strategy is never called again.

onDestroy()

The framework calls this technique when the administration is not, at this point utilized and is being decimated. This technique is utilized to, tidy up any assets, for example, strings, enlisted audience members, beneficiaries, and so forth. This is the last call the administration gets.

Administration Manifest Declaration

In principle, An assistance can be called from other application except if it is limited. You can guarantee that your administration is accessible to just your application by including the android:exported quality and setting it to "bogus". This adequately prevents different applications from beginning your administration, in any event, when utilizing an unequivocal expectation.

Starting Android Service

Intent intent = new Intent(this, HelloService.class);

startService(intent);

stopService(intent);

You can start a service from an activity or other application component by passing an Intent to startService(). The Android system calls the service’s onStartCommand() method and passes it the Intent.

In our example, we will start service by calling startService() method while start service button is clicked

Question number 3(A):-

1. **Gson**

Gson is a Java library utilized for serializing and deserializing Java objects from and into JSON. An errand you will often need to do on the off chance that you speak with APIs. We generally use JSON on the grounds that it's lightweight and a lot less complex than XML.

```
// Serialize
String userJSON = new Gson().toJson(user);

// Deserialize
User user = new Gson().fromJson(userJSON, User.class);
```

2. **Retrofi**

From their site: "Retrofit transforms your REST API into a Java interface." It's an exquisite answer for sorting out API brings in a venture. The solicitation strategy and relative URL are included with a comment, which makes code perfect and basic.

With comments, you can undoubtedly include a solicitation body, control the URL or headers and include question boundaries.

Adding a return type to a strategy will make it coordinated, while including a Callback will permit it to complete nonconcurrently with progress or disappointment..

```
public interface RetrofitInterface {

    // asynchronously with a callback
    @GET("/api/user")
    User getUser(@Query("user_id") int userId, Callback<User>
callback);

    // synchronously
    @POST("/api/user/register")
    User registerUser(@Body User user);
}


// example
RetrofitInterface retrofitInterface = new RestAdapter.Builder()

.setEndpoint(API.API_URL).build().create(RetrofitInterface.class);

// fetch user with id 2048
retrofitInterface.getUser(2048, new Callback<User>() {
    @Override
    public void success(User user, Response response) {
```

```
    }

    @Override
    public void failure(RetrofitError retrofitError) {

    }
});
```

3. EventBus

EventBus is a library that disentangles correspondence between various pieces of your application. For instance, sending something from an Activity to a running Service, or simple connection between pieces. Here is a model we use if the Internet association is lost, telling the best way to inform an action

```
public class NetworkStateReceiver extends BroadcastReceiver {

    // post event if there is no Internet connection
    public void onReceive(Context context, Intent intent) {
        super.onReceive(context, intent);
        if(intent.getExtras()!=null) {
            NetworkInfo ni=(NetworkInfo)
intent.getExtras().get(ConnectivityManager.EXTRA_NETWORK_INFO);
            if(ni!=null &&
ni.getState()==NetworkInfo.State.CONNECTED) {
                // there is Internet connection
            } else if(intent

.getBooleanExtra(ConnectivityManager.EXTRA_NO_CONNECTIVITY,Boole
an.FALSE)) {
                // no Internet connection, send network state
changed
                EventBus.getDefault().post(new
NetworkStateChanged(false));
            }
}

// event
public class NetworkStateChanged {

    private mIsInternetConnected;

    public NetworkStateChanged(boolean isInternetConnected) {
        this.mIsInternetConnected = isInternetConnected;
    }
```

```
    public boolean isInternetConnected() {
        return this.mIsInternetConnected;
    }
}

public class HomeActivity extends Activity {

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

        EventBus.getDefault().register(this); // register
EventBus
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        EventBus.getDefault().unregister(this); // unregister
EventBus
    }

    // method that will be called when someone posts an event
NetworkStateChanged
    public void onEventMainThread(NetworkStateChanged event) {
        if (!event.isInternetConnected()) {
            Toast.makeText(this, "No Internet connection!",
Toast.LENGTH_SHORT).show();
        }
    }

}
```

4. **ActiveAndroid**

ActiveAndroid is an ORM for Android. It's a reflection over SQLite which permits you to speak with an information base on a gadget without composing SQL explanations. An Object that broadens ActiveAndroid Model can be spared to the information base this way:
user.save();
which can easily replace a big SQL statement like this:

INSERT INTO Users (Nickname, Name, Address, City, PostalCode, Country) VALUES ('Batman','Bruce W','Palisades 21','Gotham','40000','USA');
An example of retrieving all users:

List<User> users = new Select().from(User.class).execute();
of which SQL counterpart would look like this:

SELECT Nickname, Name, Address, City, PostalCode, Country FROM Users;
ActiveAndroid is a nice way to remove a lot of boilerplate code used for working with databases. There are other open source solutions like GreenDAO and ORMLite

5. **Widespread Image Loader**

UIL is a library which gives offbeat, out of the crate stacking and storing of pictures. It's truly direct to utilize
imageLoader.displayImage(imageUri, imageView);
Although Picasso has a nicer API, it lacks in customization. With the UIL configuration builder almost everything can be configured (important for fetching and caching of really large images, which Picasso fails to do).

Good open source libraries will make your development a hell of a lot easier and faster. Popular libraries are often well tested and simple to use. In most cases you can easily import them into your Android Studio project from Maven. Add them into dependencies in your build.gradle file like this:

```
dependencies {
   compile 'com.google.code.gson:gson:2.2.4'
   compile 'com.squareup.okhttp:okhttp:1.3.0'
   compile 'com.squareup.retrofit:retrofit:1.3.0'
   compile 'de.greenrobot:eventbus:2.2.+'
   compile 'com.nostra13.universalimageloader:universal-image-loader:1.9.1'
```

6. **Moshi**

Moshi is a library that converts JSON into Java and Kotlin models. A lot of people refer to the Moshi as GSON 3.0. This library is superior to GSON in several aspects: it's faster, it includes Kotlin support, it's maintained, it throws predictable exceptions and it doesn't use broken DateTime adapter by default. Plus, converting JSON to Java model (and vice-versa) is straightforward with Moshi.

```
val moshi = Moshi.Builder().build()
val jsonAdapter = moshi.adapter(Model::class.java)

/* JSON to Model */
val model = jsonAdapter.fromJson(json)
```

```
/* Model to JSON */
val json = jsonAdapter.toJson(model)
```

We also admire Moshi for its JSON API support. JSON API is the specification for building APIs, and lot of APIs we work with are written using that specification. Moshi JSON API makes our life easier because it converts JSON API response into meaningful Java objects. Retrofit also has support for Moshi, and it all just clicks together.

Question number 3(B):-

1. **The Linux Kernel**

The establishment of the Android stage is the Linux part. For instance, the Android Runtime (ART) depends on the Linux part for fundamental functionalities, for example, stringing and low-level memory the board.

Utilizing a Linux piece permits Android to exploit key security includes and permits gadget producers to create equipment drivers for a notable bit.

2. **Equipment Abstraction Layer (HAL)**

The equipment reflection layer (HAL) gives standard interfaces that uncover gadget equipment capacities to the more elevated level Java API structure. The HAL comprises of various library modules, every one of which executes an interface for a particular kind of equipment segment, for example, the camera or bluetooth module. At the point when a structure API settles on a decision to get to gadget equipment, the Android framework stacks the library module for that equipment part.

3. **Android Runtime**

For gadgets running Android rendition 5.0 (API level 21) or higher, each application runs in its own cycle and with its own example of the Android Runtime (ART). Workmanship is composed to run various virtual machines on low-memory gadgets by executing DEX records, a bytecode design planned uniquely for Android that is enhanced for negligible memory impression. Manufacture toolchains, for example, Jack, gather Java sources into DEX bytecode, which can run on the Android stage.

A portion of the significant highlights of ART incorporate the accompanying:

Early (AOT) and without a moment to spare (JIT) aggregation

Enhanced trash assortment (GC)

On Android 9 (API level 28) and higher, transformation of an application bundle's Dalvik Executable arrangement (DEX) records to more minimized machine code.

Better investigating help, including a devoted examining profiler, nitty gritty symptomatic exemptions and crash detailing, and the capacity to set watchpoints to screen explicit fields

Preceding Android adaptation 5.0 (API level 21), Dalvik was the Android runtime. In the event that your application runs well on ART, at that point it should take a shot at Dalvik too, yet the converse may not be valid.

Android likewise incorporates a lot of center runtime libraries that give the vast majority of the usefulness of the Java programming language, including some Java 8 language includes, that the Java API structure employments.

4. **Local C/C++ Libraries**

Many center Android framework parts and administrations, for example, ART and HAL, are worked from local code that require local libraries written in C and C++. The Android stage gives Java structure APIs to uncover the usefulness of a portion of these local libraries to applications. For instance, you can get to OpenGL ES through the Android structure's Java OpenGL API to include uphold for drawing and controlling 2D and 3D designs in your application.

On the off chance that you are building up an application that requires C or C++ code, you can utilize the Android NDK to get to a portion of these local stage libraries straightforwardly from your local code.

5. **Java API Framework**

The whole list of capabilities of the Android OS is accessible to you through APIs written in the Java language. These APIs structure the structure blocks you have to make Android applications by rearranging the rese of center, particular framework segments and administrations, which incorporate the accompanying:

A rich and extensible View System you can use to construct an application's UI, including records, lattices, text boxes, fastens, and even an embeddable internet browser

A Resource Manager, giving admittance to non-code assets, for example, limited strings, illustrations, and design documents

A Notification Manager that empowers all applications to show custom alarms in the status bar

An Activity Manager that deals with the lifecycle of applications and gives a typical route back stack

Content Providers that empower applications to get to information from different applications, for example, the Contacts application, or to share their own information

Designers have full admittance to a similar structure APIs that Android framework applications use.

Framework Apps

Android accompanies a lot of center applications for email, SMS informing, schedules, web perusing, contacts, and the sky is the limit from there. Applications included with the stage have no extraordinary status among the applications the client decides to introduce. So an outsider application can turn into the client's default internet browser, SMS courier, or even the default console (a few exemptions apply, for example, the framework's Settings application).

The framework applications work both as applications for clients and to give key capacities that engineers can access from their own application. For instance, if your application might want to convey a SMS message, you don't have to fabricate that usefulness yourself—you can rather summon whichever SMS application is as of now introduced to convey a message to the beneficiary you determine.

Question number 4(A):-

Broadcast recipient

Definition

A transmission recipient (collector) is an Android part which permits you to enlist for framework or application occasions. All enrolled beneficiaries for an occasion are informed by the Android runtime once this occasion occurs.

For instance, applications can enlist for the ACTION_BOOT_COMPLETED framework occasion which is terminated once the Android framework has finished the boot cycle.

Usage

A beneficiary can be enrolled by means of the AndroidManifest.xml record.

On the other hand to this static enrollment, you can likewise enlist a beneficiary progressively through the Context.registerReceiver() technique.

The executing class for a beneficiary broadens the BroadcastReceiver class.

On the off chance that the occasion for which the transmission beneficiary has enrolled occurs, the onReceive() technique for the recipient is called by the Android framework.

. Life pattern of a transmission recipient

After the onReceive() of the recipient class has completed, the Android framework is permitted to reuse the beneficiary.

Nonconcurrent handling

Before API level 11, you were unable to play out any offbeat activity in the onReceive() strategy, on the grounds that once the onReceive() technique had been done, the Android framework was permitted to reuse that segment. In the event that you have conceivably long running tasks, you should trigger a help.

Since Android API 11 you can call the goAsync() technique. This strategy restores an object of the PendingResult type. The Android framework considers the recipient as alive until you call the PendingResult.finish() on this item. With this choice you can trigger nonconcurrent handling in a collector. When that string has finished, its assignment calls finish() to demonstrate to the Android framework that this part can be reused.

Limitations for characterizing broadcast recipient

As of Android 3.1 the Android framework rejects all recipient from accepting goals of course if the comparing application has never been begun by the client or if the client unequivocally halted the application by means of the Android menu (in Manage Application).

This is an extra security include as the client can be certain that lone the applications he began will get broadcast plans.

This doesn't mean the client needs to begin the application again after a reboot. The Android framework recollects that the client previously began it. Just one beginning is required without a constrained stop by the client.

Send the transmission to your application for testing

You can utilize the accompanying order from the adb order line apparatus. The class name and bundle names which are focused on by means of the order line instrument should be as characterized in the show. You ought to send the aim you produced to your particular segment, for instance on the off chance that you send a general ACTION_BOOT_COMPLETED broadcast, this will trigger a great deal of things in an Android framework.

trigger a transmission and convey it to a segment

adb shell am movement/administration/broadcast - an ACTION - c CATEGORY - n NAME

for instance (this goes into one line)

adb shell am communicated - a

android.intent.action.BOOT_COMPLETED - c android.intent.category.HOME - n

package_name/class_name

Forthcoming Intent

A forthcoming aim is a symbolic that you provide for another application. For instance, the notice supervisor, caution chief or other outsider applications). This permits the other application to reestablish the consents of your application to execute a predefined bit of code.

To play out a transmission by means of a forthcoming expectation, get a PendingIntent through the getBroadcast() technique for the PendingIntent class. To play out a movemen by means of a forthcoming purpose, you get the action through PendingIntent.getActivity().

Framework communicates

A few framework occasions are characterized as conclusive static fields in the Intent class. Other Android framework classes additionally characterize occasions, eg., the TelephonyManager characterizes occasions for the difference in the telephone state.

The accompanying table records a couple of significant framework occasions.

Table 1. Framework Events

Event Description

Intent.ACTION_BOOT_COMPLETED

Boot finished. Requires the android.permission.RECEIVE_BOOT_COMPLETED consent

Intent.ACTION_POWER_CONNECTED

Force got associated with the gadget.

Intent.ACTION_POWER_DISCONNECTED

Force got detached to the gadget.

Intent.ACTION_BATTERY_LOW

Set off on low battery. Normally used to decrease exercises in your application which devour power.

Intent.ACTION_BATTERY_OKAY

Battery status great once more.

Consequently beginning Services from a Receivers

A typical prerequisite is to naturally begin a help after a framework reboot, i.e., for synchronizing information. For this you can enlist a recipient for the android.intent.action.BOOT_COMPLETED framework occasion. This requires the android.permission.RECEIVE_BOOT_COMPLETED consent.

The accompnying model exhibits the enlistment for the BOOT_COMPLETED occasion in the Android show record.

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

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

package="de.vogella.android.ownservice.local"

android:versionCode="1"

android:versionName="1.0" >

<uses-sdk android:minSdkVersion="10"/>

<uses-authorization android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>

<application

android:icon="@drawable/symbol"

android:label="@string/app_name" >

<activity

android:name=".ServiceConsumerActivity"

android:label="@string/app_name" >

<intent-filter>

<action android:name="android.intent.action.MAIN"/>

<category android:name="android.intent.category.LAUNCHER"/>

</goal filter>

</activity>

```
<receiver android:name="MyScheduleReceiver" >
<intent-filter>
<action android:name="android.intent.action.BOOT_COMPLETED"/>
</goal filter>
</receiver>
<receiver android:name="MyStartServiceReceiver" >
</receiver>
</application>
</manifest>
```

The get would begin the administration as exhibited in the accompanying model code.

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
public class MyReceiver expands BroadcastReceiver {
@Override
public void onReceive(Context setting, Intent plan) {
/accept WordService is an enlisted administration
Plan purpose = new Intent(context, WordService.class);
context.startService(intent);
}
}
```

On the off chance that your application is introduced on the SD card, at that point it isn't accessible after the android.intent.action.BOOT_COMPLETED occasion. For this situation register it for the 'android.intent.action.ACTION_EXTERNAL_APPLICATIONS_AVAILABLE ' occasion.

Recall that as of Android API level 11 the client needs to have begun the application at any rate once before your application can get android.intent.action.BOOT_COMPLETED occasions.

Exercise: Register a beneficiary for approaching calls

Target

In this activity you characterize a transmission beneficiary which tunes in to phone state changes. On the off chance that the telephone gets a call, at that point our recipient will be informed and log a message.

Make venture

Make another task called de.vogella.android.receiver.phone. Additionally make an action.

TIP:Reember that your beneficiary is possibly called if the client began it once. This requires an action.

Execute beneficiary for the telephone occasion

Make the MyPhoneReceiver class.

```
bundle de.vogella.android.receiver.phone;

import android.content.BroadcastReceiver;

import android.content.Context;

import android.content.Intent;

import android.os.Bundle;

import android.telephony.TelephonyManager;

import android.util.Log;ublic class MyPhoneReceiver broadens BroadcastReceiver {

@Overrid

public void onReceive(Context setting, Intet aim) {

Group additional items = intent.getExtras();

on the off chance that (additional items != invalid) {

String state = extras.getString(TelephonyManager.EXTRA_STATE);

Log.w("MY_DEBUG_TAG", state);

in the case of (state.equals(TelephonyManager.EXTRA_STATE_RINGING)) {

String phoneNumber = additional items

.getString(TelephonyManager.EXTRA_INCOMING_NUMBER);

Log.w("MY_DEBUG_TAG", phoneNumber);

}

}

}

}
```

Solicitation authorization

Add the android.permission.READ_PHONE_STATE authorization to your show docment which permits you to tune in to state changes in your beneficiary. Additionally Register yor beneficiary in your show record. The subsequent show ought to be like the accompanying posting.

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

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

package="de.vogella.android.receiver.phone"

android:versionCode="1"

android:versionName="1.0" >

<uses-sdk android:minSdkVersion="15"/>

<uses-consent android:name="android.permission.READ_PHONE_STATE" >

</utilizes permission>

<application

android:icon="@drawable/symbol"

android:label="@string/app_name" >

<activity

android:name=".MainActivity"

android:label="@string/title_activity_main" >

<intent-filter>

<action android:name="android.intent.action.MAIN"/>

<category android:name="android.intent.category.LAUNCHER"/>

</plan filter>

</activity>

<receiver android:name="MyPhoneReceiver" >

<intent-filter>

<action android:name="android.intent.action.PHONE_STATE" >

</action>

</plan filter>

</receiver>

</application>
```

</manifest>

Approve executions

Introduce your application and recreate a call by means of the emulator controls. Approve that your collector is called and logs a message to the LogCat see.

Exercise: System administrations and recipient. Target

In this section we will plan a recipient by means of the Android ready administrator framework administration. Once called, it utilizes the Android vibrator chief and a popup message (Toast) to inform the client.

Actualize venture

Make another task called de.vogella.android.alarm with the action called AlarmActivity.

Make the accompanying design.

<?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" >

<EditText

android:id="@+id/time"

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:hint="Number of seconds"

android:inutType="numberDecimal" >

</EditText>

<Button

android:id="@+id/alright"

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:onClick="startAlert"

android:text="Start Counter" >

</Button>

</LinearLayout>

Make the accompanying transmission collector class. This class will get the vibrator administration.

```
bundle de.vogella.android.alarm;

import android.content.BroadcastReceiver;

import android.content.Context;

import android.content.Intent;

import android.os.Vibrator;

import android.widget.Toast;

public class MyBroadcastReceiver broadens BroadcastReceiver {

@Override

public void onReceive(Context setting, Intent goal) {

Toast.makeText(context, "Don't panik however your time is up!!!!.",

Toast.LENGTH_LONG).show();

/Vibrate the cell phone

Vibrator = (Vibrator) context.getSystemService(Context.VIBRATOR_SERVICE);

vibrator.vibrat
```

Register this class as a transmission collector in AndroidManifest.xml and solicitation approval to vibrate the telephone.

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

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

package="de.vogella.android.alarm"

android:versionCode="1"

android:versionName="1.0" >

<uses-sdk android:minSdkVersion="15"/>

<uses-authorization android:name="android.permission.VIBRATE" >

</utilizes permission>

<application

android:icon="@drawable/symbol"
```

```
android:label="@string/app_name" >

<activity

android:name=".AlarmActivity"

android:label="@string/app_name" >

<intent-filter>

<action android:name="android.intent.action.MAIN"/>

<category android:name="android.intent.category.LAUNCHER"/>

</goal filter>

</activity>

<receiver android:name="MyBroadcastReceiver" >

</receiver>

</application>

</manifest>
```

Change the code of your AlarmActivity class to the accompanying. This movement makes a purpose to begin the recipient and register this plan with the caution supervisor administration.

```
bundle de.vogella.android.alarm;

import android.app.Activity;

import android.app.AlarmManager;

import android.app.PendingIntent;

import android.content.Intent;

import android.os.Bundle;

import android.view.View;

import android.widget.EditText;

import android.widget.Toast;

public class AlarmActivity broadens Activity {

/** Called when the movement is first made. */

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.main);

}

public void startAlert(View see) {

EditText text = (EditText) findViewById(R.id.time);

int I = Integer.parseInt(text.getText().toString());

Expectation purpose = new Intent(this, MyBroadcastReceiver.class);

pendingIntent = PendingIntent.getBroadcast(

this.getApplicationContext(), 234324243, expectation, 0);

alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);

alarmManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis()

+ (I * 1000), pendingIntent);

Toast.makeText(this, "Alert set in " + I + " seconds",

Toast.LENGTH_LONG).show();

}

}
```

Approve usage

Run your application on the gadget. Set your time and start the alert. After the characterized number of seconds a Toast ought to be shown. Remember that the vibration caution doesn't deal with the Android emulator.

Caution application running

Dynamic transmission beneficiary enrollment

Powerfully enrolled recipient

Beneficiary can be enrolled through the Android show document. You can likewise enlist and unregister a collector at runtime by means of the Context.registerReceiver() and Context.unregisterReceiver() strategies.

Remember to unregister a powerfully enrolled recipient by utilizing Context.unregisterReceiver() strategy. In the event that you overlook this, the Android framework reports a spilled broadcast collector mistake. For example, in the event that you enlisted a get in onResume() strategies for your movement, you ought to unregister it in the onPause() technique.

Utilizing the bundle director to debilitate static recipients

You can utilize the PackageManger class to empower or debilitate collectors enlisted in your AndroidManifest.xml document.

ComponentName collector = new ComponentName(context, myReceiver.class);

PackageManager pm = context.getPackageManager();

pm.setComponentEnabledSetting(receiver,

PackageManager.COMPONENT_ENABLED_STATE_ENABLED,

PackageManager.DONT_KILL_APP);

Clingy (broadcast) aims

An aim to trigger a collector (broadcast plan) isn't accessible any longer after it was sent and handled by the framework. On the off chance that you utilize the sendStickyBroadcast(Intent) technique, the comparing aim is clingy, which means the expectation you are sending remains around after the transmission is finished.

The Android framework utilizes clingy broadcast for certain framework data. For instance, the battery status is send as clingy plan and can get got whenever. The accompanying model exhibits that.

/Register for the battery changed occasion

IntentFilter channel = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);

/Intent is clingy so utilizing invalid as beneficiary works fine

/return esteem contains the status

Goal batteryStatus = this.registerReceiver(null, channel);

/Are we charging/charged?

int status = batteryStatus.getIntExtra(BatteryManager.EXTRA_STATUS, - 1);

boolean isCharging = status == BatteryManager.BATTERY_STATUS_CHARGING

|| status == BatteryManager.BATTERY_STATUS_FULL;

boolean isFull = status == BatteryManager.BATTERY_STATUS_FULL;

How are we charging?

int chargePlug = batteryStatus.getIntExtra(BatteryManager.EXTRA_PLUGGED, - 1);

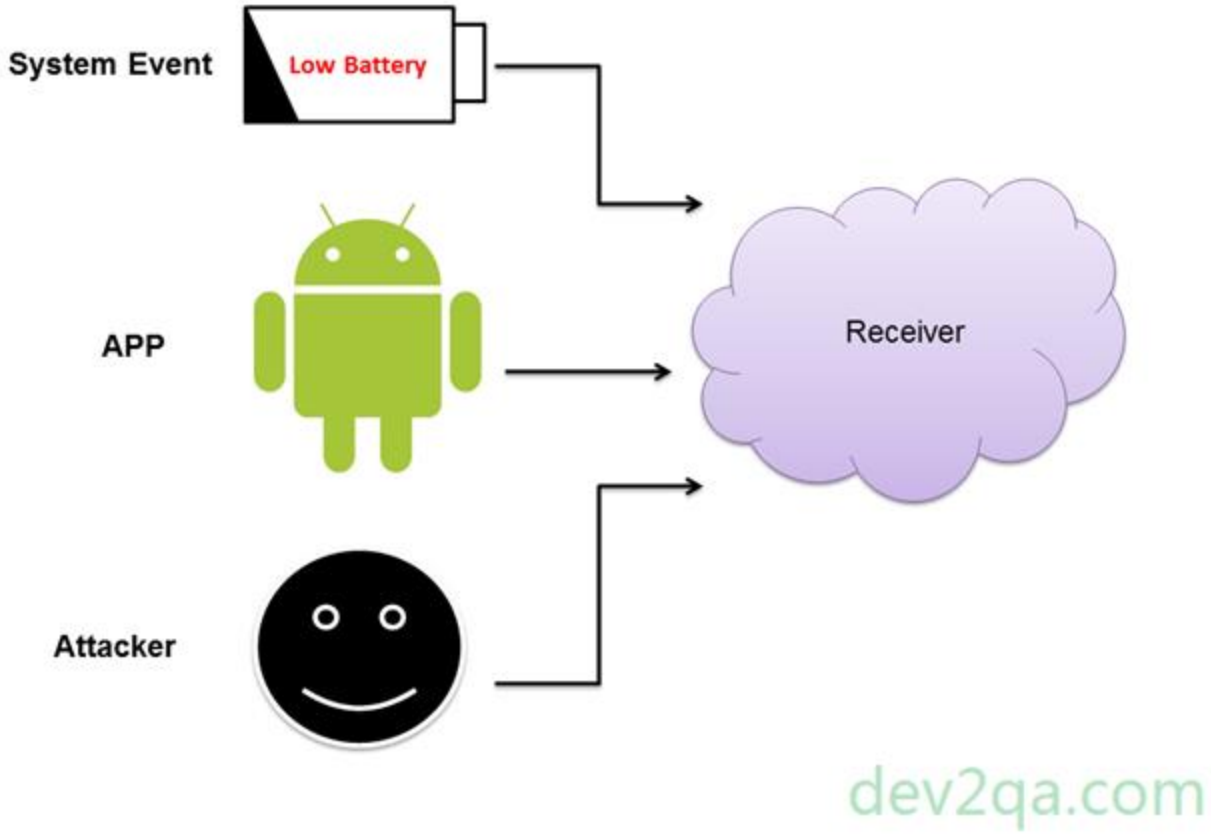
boolean usbCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_USB;

boolean acCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_AC;

You can recover that information through the return estimation of registerReceiver(BroadcastReceiver, IntentFilter) '. This likewise works for an invalid 'BroadcastReceiver.

In every different manners, this acts similarly as sendBroadcast(Intent).

Clingy broadcast plans ordinarily require uncommon consents.



Question number 4(B):-

In android, Content Provider will act as a central repository to store the data of the application in one place and make that data available for different applications to access whenever it's required

In android, we can configure Content Providers to allow other applications securely access and modify our app data based on our requirements.

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

In android, we can use content provider whenever we want to share our app data with other apps and it allow us to make a modifications to our application data without effecting other applications which depends on our app.

In android, the content provider is having different ways to store app data. The app data can be stored in a SQLite database or in files or even over a network based on our requirements. By using content providers we can manage data such as audio, video, images and personal contact information.

We have different type of access permissions available in content provider to share the data. We can set a restrict access permissions in content provider to restrict data access limited to only our application and we can configure different permissions to read or write a data.

Access Data from Content Provider

To access a data from content provider, we need to use ContentResolver object in our application to communicate with the provider as a client. The ContentResolver object will communicate with the provider object (ContentProvider) which is implemented by an instance of class.

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

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

Following is the pictorial representation of requesting an operation from UI using Activity or Fragment to get the data from ContentProvider object.

Creating a Content Provider

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

We need to create a content provider class that extends the ContentProvider base class.

We need to define our content provider URI to access the content.

The ContentProvider class defines a six abstract methods (insert(), update(), delete(), query(), getType()) which we need to implement all these methods as a part of our subclass.

We need to register our content provider in AndroidManifest.xml using <provider> tag.

Following are the list of methods which need to implement as a part of ContentProvider class.

Android Content Provider - Insert, Query, Update, Delete, getType, onCreate Methods

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

insert() - This method will insert a new row into our content provider and it will return the content URI for newly inserted row.

update() - This method will update existing rows in our content provider and it returns the number of rows updated.

delete() - This method will delete the rows in our content provider and it returns the number of rows deleted.

getType() - This method will return the MIME type of data to given content URI.

onCreate() - This method will initialize our provider. The android system will call this method immediately after it creates our provider.