Hall Ticket Number:

III/IV B.Tech (Regular/Supplementary) DEGREE EXAMINATION

December, 2024 — **Information Technology**

Fifth Semester — **Mobile Application Development**

Time: Three Hours — **Maximum:**70 Marks

Answer question 1 compulsorily. **(14X1 = 14Marks)**

Answer one question from each unit. **(4X14=56Marks)**

			CO	BL	M
1	a)	What are Background Services.	CO1	L1	1M
	b)	Draw Android Application architecture.	CO1	L2	1M
	c)	List Android Libraries.	CO1	L1	1M
	d)	What are Background Applications?	CO1	L1	1M
	e)	Explain how to create styles.	CO2	L2	1M
	f)	What are activity states?	CO2	L1	1M
	g)	What is constraint layout?	CO2	L1	1M
	h)	Write syntax of defining a Fragment class.	CO2	L2	1M
	i)	List usage of intents.	CO3	L1	1M
	j)	List out various list views in Android	CO3	L1	1M
	k)	What are life cycle handlers?	CO3	L1	1M
	l)	Write query to connect to a Database.	CO4	L2	1M
	m)	What are the various pre-defined content providers in Android?	CO4	L1	1M
	n)	Write handlers used in case of menus in Android.	CO4	L1	1M
		Unit-I			
2	a)	Explain in detail about Android SDK features.	CO1	L2	7M
	b)	Discuss the steps involved in creating your First Android Applications.	CO1	L2	7M
		(OR)			
3	a)	Describe the following android applications a. Intermittent b. Widjets c. Foreground applications	CO1	L2	7M
	b)	Explain Android architecture with neat diagram.	CO1	L2	7M
		Unit-II			
4	a)	Describe Android Application Life cycle	CO2	L2	7M
	b)	Illustrate Application Manifest file.	CO2	L2	7M
		(OR)			
5	a)	Describe following a. Creating Activitiesb. Creating fragments	CO2	L2	7M
	b)	Demonstrate the use of following views in Android i) RadioButton ii) CheckBox iii) ImageView	CO2	L2	7M
		Unit-III			
6	a)	Explain the procedure for creating and saving shared preferences.	CO3	L2	7M
	b)	Develop an android application using explicit intents.	CO3	L6	7M
		(OR)			
7		Describe the following a. Using Intent Filters to service Implicit Intents. b. Using Intent Filters for Plug – In and extensibility.	CO3 CO3	L2 L2	7M 7M
		Unit-IV			
8		Develop an Android program for maintaining a To – Do List database using SQLite database.	CO4	L6	14M
		(OR)			
9		Illustrate following a. Services b. options menu and context menu.	CO4	L2	14M

SCHEME OF EVALUATION

20IT505/JO1A

Hall Ticket Number:

III/IV B.Tech (Regular/Supplementary) DEGREE EXAMINATION

December, 2024 — **Information Technology**

Fifth Semester — **Mobile Application Development**

Time: Three Hours — **Maximum:**70 Marks

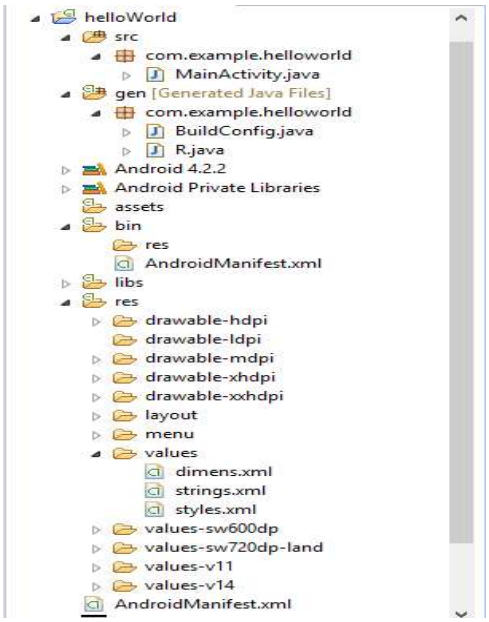
Answer question 1 compulsorily. **(14X1 = 14Marks)**

Answer one question from each unit. **(4X14=56Marks)**

1 a) What are Background Services. ***CO1 L1 1M***

A Service is an application component that can perform long-running operations in the background. It does not provide a user interface.

1 b) Draw Android Application architecture. ***CO1 L2 1M***



1 c) List Android Libraries. ***CO1 L1 1M***

android.util	android.os	android.graphics	android.text
android.database	android.content	android.view	android.widget
com.google.android.maps	android.app	android.provider	android.telephony
android.webkit			

1 d) What are Background Applications? ***CO1 L1 1M***

An app is running in the background if none of its activities are visible to the user, and the app isn't running any foreground services. Ex: Whatsapp, Facebook, alarm etc..

1e) Explain how to create styles. ***CO2 L2 1M***

In res/values folder, create an xml file with the following syntax:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="CodeFont" parent="@android:style/TextAppearance.Medium">
      <item name="android:layout_width">fill_parent</item>
      <item name="android:layout_height">wrap_content</item>
      <item name="android:textColor">#00FF00</item>
```

```
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

For a TextView apply the style 'CodeFont' as shown below:

```
<TextView
style="@style/CodeFont"
android:text="@string/hello" />
```

1 f) What are activity states? ***CO2 L1 1M***

i) Resumed/Running state
ii) Paused state
iii) Stopped stage

1 g) What is constraint layout? ***CO2 L1 1M***

Android ConstraintLayout is used to define a layout by assigning constraints for every child view/widget relative to other views present. A ConstraintLayout is similar to a RelativeLayout, but with more power.

1 h) Write syntax of defining a Fragment class. ***CO2 L2 1M***

```
class ExampleFragment extends Fragment {
public ExampleFragment() {
        super(R.layout.example_fragment);
    }
}
```

1 i) List usage of intents. ***CO3 L1 1M***

An intent is used for
i) launching an activity
ii) starting a service, broadcast receiver
iii) passing data between activities
iv) launching preinstalled apps.

1 j) List out various list views in Android ***CO3 L1 1M***

ListView & Spinner

1k) What are life cycle handlers? ***CO3 L1 1M***

Activity life cycle handlers: onCreate( ), onStart( ), onResume( ), onPause( ), onStop( ), onDestroy( )

1l) Write query to connect to a Database. ***CO4 L2 1M***

For connecting to SQLite database in Android, use getWritableDatabase( ) or getReadableDatabase( ) of SQLiteOpenHelper.

1m) What are the various pre-defined content providers in Android? ***CO4 L1 1M***

Browser — Stores data such as browser bookmarks, browser history, and so on
CallLog — Stores data such as missed calls, call details, and so on
Contacts — Stores contact details
MediaStore — Stores media files such as audio, video and images
Settings — Stores the device's settings and preferences

1n) Write handlers used in case of menus in Android. ***CO4 L1 1M***

ContextMenu handlers: onCreateContextMenu( ) and onContextItemSelected( )
OptionsMenu handlers: onCreateOptionsMenu( ) and onOptionsItemSelected( )

Unit-I

2 a) Explain in detail about Android SDK features. ***CO1 L2 7M***

The following are the main Android SDK features: *<7M >*

- ✓ GSM, EDGE, 3G, 4G, and LTE networks for telephony or data transfer, enabling you to make or receive calls or SMS messages, or to send and retrieve data across mobile networks.
- ✓ Comprehensive APIs for location-based services such as GPS and network-based location detection.
- ✓ Full support for applications that integrate map controls as part of their user interfaces
- ✓ Wi-Fi hardware access and peer-to-peer connections
- ✓ Full multimedia hardware control, including playback and recording with the camera and microphone.
- ✓ Uses SQLite, a light weight relational database, for data storage.
- ✓ Media libraries for playing and recording a variety of audio/video or still-image formats
- ✓ APIs for using sensor hardware, including accelerometers, gyroscopes, barometers, magnetometers, dedicated gaming controls, proximity and pressure sensors, thermometers.
- ✓ Libraries for using Bluetooth and NFC hardware for peer-to-peer data transfer
- ✓ IPC message passing
- ✓ Shared data stores and APIs for contacts, social networking, calendar, and multi-media
- ✓ Background Services, applications, and processes
- ✓ Home-screen Widgets and Live Wallpaper
- ✓ The ability to integrate application search results into the system searches
- ✓ An integrated open-source HTML5 WebKit-based browser
- ✓ Mobile-optimized, hardware-accelerated graphics, including a path-based 2D graphics library and support for 3D graphics using OpenGL ES 2.0
- ✓ Localization through a dynamic resource framework
- ✓ An application framework that encourages the reuse of application components and the replacement of native applications.
- ✓ Touchscreen support
- ✓ Tethering: Supports sharing of Internet connections as a wired/wireless hotspot
- ✓ Video Calling
- ✓ Android supports capturing a screenshot by pressing the power and volume-down buttons at the same time.
- ✓ Most Android devices include microSD slot for external storage and can read microSD cards formatted with FAT32, Ext3 or Ext4file system.

2 b) Discuss the steps involved in creating your First Android Applications. *CO1 L2 7M*

1. Open Android Studio.
2. Under the "Quick Start" menu, select "Start a new Android Studio project."
3. On the "Create New Project" window that opens, name your project "HelloWorld".
4. If you choose to, set the company name as desired.
5. Note where the project file location is and change it if desired.
6. Click "Next."
7. Make sure on that "Phone and Tablet" is the only box that is checked.
8. If you are planning to test the app on your phone, make sure the minimum SDK is below your phone's operating system level.
9. Click "Next."
10. Select "Empty Views Activity."
11. Click "Next."
12. Leave all of the Activity name fields as they are.
13. Click "Finish."

follow the below steps to make changes in activity_main.xml

1. Make sure that the Design tab is open on the activity_main.xml display.
2. Click and drag the "Hello, world!" from the upper left corner of the phone display to the center of the screen.
3. In the project file system on the left side of the window, open the values folder.
4. In the values folder, double-click the strings.xml file.
5. In this file, find the line "Hello world!".
6. After the "Hello world!" message, add "Welcome to my app!"
7. Navigate back to the activity_main.xml tab.
8. Make sure that your centered text now reads "Hello world! Welcome to my app!"

Add a Button to the Main Activity

Add Button's "onClick" Method:
Steps:

1. **Select the** MainActivity.java **tab along the top of the work environment.**
2. Add the following lines of code at the end of the onCreate method:

```
Button button = (Button) findViewById(R.id.button);
button.setOnClickListener(new View.onClickListener() {
@Override
public void onClick(View v) {
goToSecondActivity();
}});
```

3. Add the following method to the bottom of the **MainActivity class**:

```
private void goToSecondActivity() {
Intent intent = new Intent(this, SecondActivity.class);
startActivity(intent);
}
```

4. Click the + next to import at the third line of **MainActivity.java** to expand the import statements.
5. Add the following to the end of the import statements if they are not already there:

```
import android.content.Intent;
import android.view.View;
import android.widget.TextView;
```

6. Test the Application

(OR)

3 a) Describe the following android applications

a. Intermittent b. Widgets c. Foreground applications ***CO1 L2 7M***

< 3M >

Intermittent applications: Intermittent Activity Expects some interactivity but does most of its work in the background. Often these applications will be set up and then run silently, notifying users when appropriate. A common example would be a media player. Complex applications are difficult to pigeonhole into a single category and include elements of all three. When creating your application, you need to consider how it's likely to be used and then design.

Often you'll want to create an application that reacts to user input but is still useful when it's not the active foreground Activity. These applications are generally a union of a visible controller Activity with an invisible background Service. These applications need to be aware of their state when interacting with the user. This might mean updating the Activity UI when it's visible and sending notifications to keep the user updated when it's in the background.

Examples: *Email & Media Players, chat applications.*

< 2M >

Widgets: Is a small control of the application placed on the home screen. Allows you to put favourite applications on home screen in order to quickly access them. Widget only applications are commonly used to display dynamic information, such as battery levels, weather forecasts, or the date and time.

You can think of them as "at-a-glance" views of an app's most important data and functionality that are accessible right on the user's home screen. Users can move widgets across their home screen panels, and, if supported, resize them to tailor the amount of information in the widget to their preference.

Widgets can be Information Widgets, Collection widgets, Control Widgets and Hybrid widgets.

<2M>

Foreground applications: The app "is in the foreground" when it is the app that the user sees on the screen. Apps that are in the foreground are always running and interacts with the user using a UI screen,

Ex: Games,

An app is considered to be in the foreground if any of the following is true:

- ✓ It has a visible activity, whether the activity is started or paused.
- ✓ It has a foreground service.
- ✓ Another foreground app is connected to the app, either by binding to one of its services or by making use of one of its content providers.

If none of those conditions is true, the app is considered to be in the background.

3 b) Explain Android architecture with neat diagram. ***CO1 L2 7M***

The Android software stack is, put simply, a Linux kernel and a collection of C/C++ libraries exposed through an application framework that provides services for, and management of, the run time and applications.

The following diagram shows the Architecture/Software Stack of an Android. <1M>

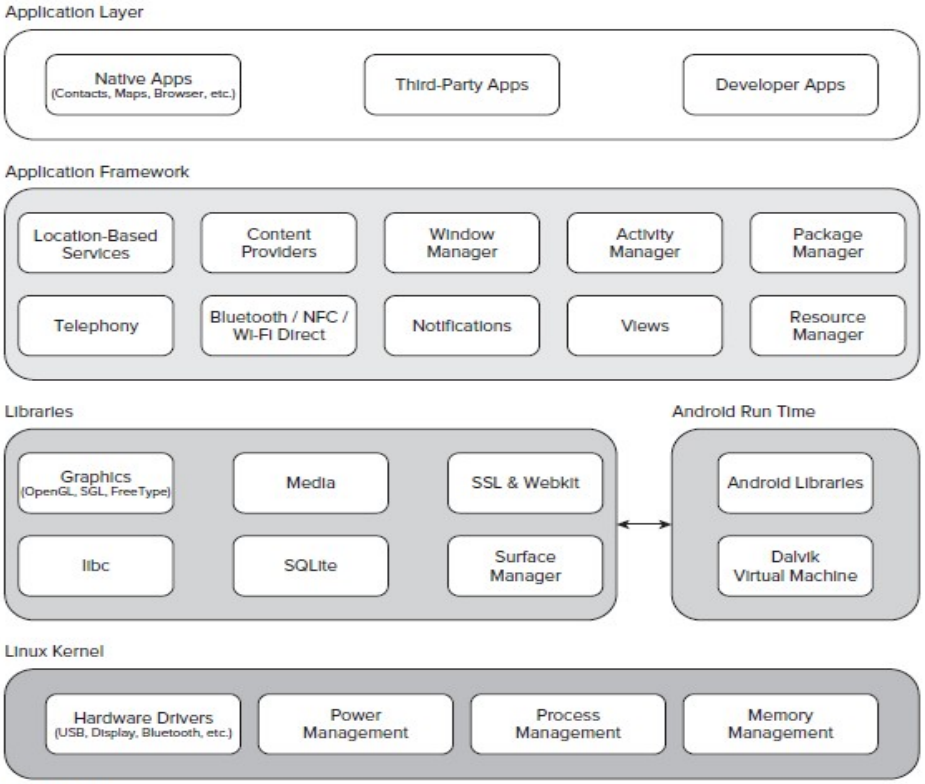


FIGURE 1-1

Android Software Stack is divided into 4 layers : <4 Layers → 6M>

1. Linux Kernel layer
2. Libraries & Android Runtime layer
3. Application framework
4. Application layer

Linux Kernel layer:

This is the heart of the Android OS. This layer contains all the low level device drivers for the various hardware components of an Android device. The kernel also provides an abstraction layer between the hardware and the remainder of the stack.

It provides the following functionalities in the Android system.

- ✓ Hardware Abstraction
- ✓ Memory Management Programs
- ✓ Security Settings
- ✓ Power Management Software
- ✓ Other Hardware Drivers
- ✓ Support for Shared Libraries
- ✓ Network stack

Libraries & Android Runtime Layer:

Libraries: Running on top of the kernel, Android includes various C/C++ core libraries such as libc and SSL. This layer contains all the code that provides the main features of an Android OS. For example, the SQLite library provides database support so that an application can use it for data storage. The WebKit library provides functionalities for web browsing.

- **Surface Manager: A surface manager to provide display management.** It is used for compositing window manager with off-screen buffering. Off-screen buffering means you can not directly draw into the screen, but your drawings go to the off-screen buffer. There it is combined with other drawings and form the final screen the user will see. This off screen buffer is the reason behind the transparency of windows.

- **SQLite:** SQLite is the database engine used in android for large data storage purposes.
- **WebKit:** It is the browser engine used to display HTML content
- **OpenGL:** Used to render 3D graphics content to the screen
- **SGL (Scalable Graphics Library):** 2D Graphics
- **Open GL|ES:** 3D Library
- **Media Framework:** Supports playbacks and recording of various audio, video and picture formats.
- **Free Type:** Font Rendering
- **libc** (System C libraries)
- **Open SSL (Secure Socket Layer)**: A cryptographic protocol for providing secure communication over the internet. (ex: internet banking).

Android Runtime:

Android Libraries: At the same layer as the libraries, the Android runtime provides a set of core libraries that enable developers to write Android apps using the Java programming language.

Dalvik VM: The Android runtime also includes the Dalvik virtual machine, which enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine (Android applications are compiled into the Dalvik executables). Dalvik is a register-based specialized virtual machine that's been optimized to ensure that a device can run multiple instances efficiently. It relies on the Linux kernel for threading and low-level memory management.

Application framework layer:

The application framework provides the classes used to create Android applications. It also provides a generic abstraction for hardware access and manages the user interface and application resources. Exposes the various capabilities of the Android OS to application developers so that they can make use of them in their applications. Our applications directly interact with these blocks of the Android architecture. These programs manage the basic functions of phone like resource management, voice call management etc.

Important blocks in this layer are:

- **Activity Manager**: Manages the activity life cycle of applications. To understand the Activity component in Android
- **Content Providers:** Manage the data sharing between applications.
- **Telephony Manager:** Manages all voice calls. We use telephony manager if we want to access voice calls in our application.

- **Location Manager:** Location management, using GPS or cell tower
- **Resource Manager:** Manage the various types of resources we use in our Application.

Applications Layer:

At this top layer, you will find applications that ship with the Android device (such as Phone, Contacts, Browser, etc.), as well as applications that you download and install from the Android Market. Any applications that you write are located at this layer.

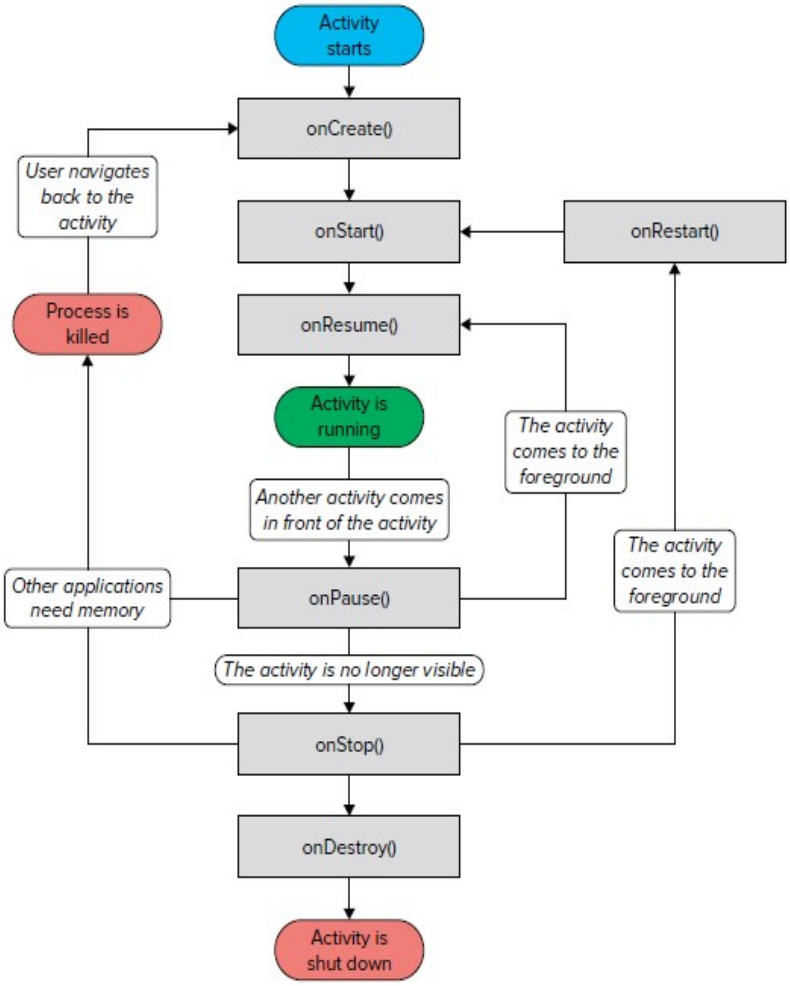
Example applications are:

- SMS client app
- Dialer
- Web browser
- Contact manager
- Google Maps
- Gallery

UNIT-II

4 a) Describe Android Application Life cycle ***CO2 L2 7M***

The following diagram shows the life cycle of an activity. **<2M>**



An activity main states are : **<2M>**

Resumed: In this state, activity is in the foreground and the user can interact with it. Also called "running" state.

Paused: In this state, the activity is partially obscured by another activity. The other activity does not cover full screen. The paused activity does not receive user input and cannot execute any code.

Stopped: In this state, the activity is completely hidden and not visible to the user. Activity information and all of its state (variables) is retained, but it cannot execute any code.

The Activity base class defines a series of events that governs the life cycle of an activity. The Activity class defines the following events: **<3M>**

Event	Description
onCreate()	Called when the activity is first created
onStart()	Called when the activity becomes visible to the user
onResume()	Called when the activity starts interacting with the user
onPause()	Called when the current activity is being paused and the previous activity is being resumed.
onStop()	Called when the activity is no longer visible to the user
onDestroy()	Called before the activity is destroyed by the system (either manually or by the system to conserve memory).
onRestart()	Called when the activity has been stopped and is restarting again.

Case 1: When the activity is first loaded into memory, the following methods are called in sequence.

onCreate( ) →onStart( ) →onResume( )

Case 2: When you now press the **back button** on the Android Emulator, sequence of methods called are

onPause( ) →onStop( ) →onDestroy( )

Case 3: Click the **Home** button and hold it there. In the Home Screen, click the App Launcher Icon. Then the following methods are called in sequence.

onCreate( ) →onStart( ) →onResume( )

Case 4: Press the **Phone** button on the Android Emulator so that the activity is pushed to the background. Then the following two methods are called in sequence.

onPause( ) →onStop( )

Case 5: Exit the phone dialer by pressing the Back button. The activity is now visible again. Then the following two methods are called in sequence.

onRestart( ) →onStart( ) →onResume( )

4 b) Illustrate Application Manifest file. ***CO2 L2 7M***

<2M>

Every project in Android includes a Manifest XML file, which is AndroidManifest.xml, located in the root directory of its project hierarchy. The manifest file is an important part of our app because it defines the structure and metadata of our application, its components, and its requirements. This file includes nodes for each of the Activities, Services, Content Providers, and Broadcast Receivers that make the application, and using Intent Filters and Permissions determines how they coordinate with each other and other applications.
The manifest file also specifies the application metadata, which includes its icon, version number, themes, etc., and additional top-level nodes can specify any required permissions, and unit tests, and define hardware, screen, or platform requirements. The manifest comprises a root manifest tag with a package attribute set to the project's package. It should also include an xmls:android attribute that will supply several system attributes used within the file. We use the versionCode attribute is used to define the current application version in the form of an integer that increments itself with the iteration of the version due to update. Also, the versionName attribute is used to specify a public version that will be displayed to the users.
We can also specify whether our app should install on an SD card of the internal memory using the installLocation attribute.

AndroidManifest.xml file contains the following:

1. manifest **<5M>**

The main component of the AndroidManifest.xml file is known as manifest. Additionally, the packaging field describes the activity class's package name. It must contain an <application> element with the xmlns:android and package attribute specified.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.geeksforgeeks">
    <!-- manifest nodes -->
    <application>
    </application>
</manifest>
```

2. uses-sdk

It is used to define a minimum and maximum SDK version by means of an API Level integer that must be available on a device so that our application functions properly, and the target SDK for which it has been designed using a combination of minSdkVersion, maxSdkVersion, and targetSdkVersion attributes, respectively.

```
<uses-sdk
    android:minSdkVersion="18"
    android:targetSdkVersion="27" />
```

3. uses-permission

It outlines a system permission that must be granted by the user for the app to function properly and is contained within the <manifest> element.

```
<uses-permission
    android:name="android.permission.CAMERA"
    android:maxSdkVersion="18" />
```

4. application

A manifest can contain only one application node. It uses attributes to specify the metadata for your application (including its title, icon, and theme).

5. uses-library

It defines a shared library against which the application must be linked. This element instructs the system to add the library's code to the package's class loader. It is contained within the <application> element.

```
<uses-library
    android:name="android.test.runner"
    android:required="true" />
```

6. activity

The Activity sub-element of an application refers to an activity that needs to be specified in the AndroidManifest.xml file. It has various characteristics, like label, name, theme, launchMode, and others.

```
<activity
   android:name=".MainActivity"
   android:exported="true">
</activity>
```

7. intent-filter

It is the sub-element of activity that specifies the type of intent to which the activity, service, or broadcast receiver can send a response.

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

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

8. action

It adds an action for the intent-filter. It is contained within the <intent-filter> element.

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

9. category

It adds a category name to an intent-filter. It is contained within the <intent-filter> element.

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

10. uses-configuration

The uses-configuration components are used to specify the combination of input mechanisms that are supported by our application.

11. uses-features

It is used to specify which hardware features your application requires. This will prevent our application from being installed on a device that does not include a required piece of hardware such as NFC hardware

(OR)

5 a) Describe the following

i. Creating Activities ii. Creating fragments ***CO2 L2 7M***

Creating Activities: ***<3M>***

Steps:

1. First make sure you have selected the Project view's Android subview.
2. Then right-click the app folder in Android view, click the " New —> Activity " menu. There will list some activity template menu, you can select the one you need. In this example, we select the Empty Activity template.
3. Then the Configure Activity window will popup. You can input or modify the activity name and layout name as you need. Commonly the layout name is separated by underlines. You can check the Launcher Activity checkbox to make it the android application startup activity.
4. Click the Finish button to complete the activity creation process. After that, you can see the three core files in the Project View on the left panel.
5. In this example, AndroidManifest.xml is located in the app/manifests folder. ButtonListenerExampleActivity java file is located in app/java folder. activity_button_listener_example.xml is located in the app/res/layout folder.
6. Then you can double click the above file to write code in it.

Creating Fragments: ***<4M>***

Steps:

Step 1: Create a New Project in Android Studio

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio. The code for that has been given in both Java and Kotlin Programming Language for Android.

Step 2: Create New Fragment. Right-Click on the First button inside Java, then click on New. Then Go to Fragment.

Step 3: The next step is to choose the Fragment type.

Fragment in Android refers to a single screen with a user interface. For Beginners, "Blank Fragment" is recommended).

Step 4: Click on Fragment(Blank).

Now, A new Dialog Box will appear.

Step 5: Then, fill in the Fragment Name text field. In Android studio, files named in CamelCase are preferred.

An XML file is used to provide functionalities of the user interface for our app. An XML file of the fragment is created using the first word in the fragment name.

Step 6: Click on Finish. And you have created your Fragment.

5 b) Demonstrate the use of following views in Android

i) RadioButton ii) CheckBox iii) ImageView ***CO2 L2 7M***

RadioButton view: **<3M>**

Step 1: in activity_main.xml add:

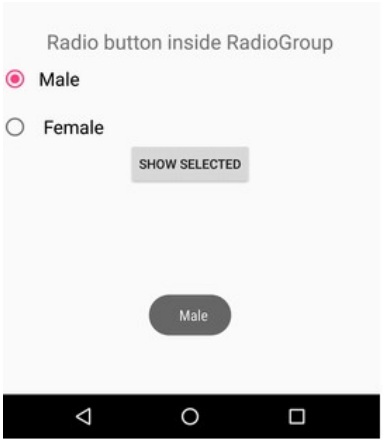
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   tools:context="example.javatpoint.com.radiobutton.MainActivity">

<TextView
      android:id="@+id/textView2"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_marginTop="30dp"
      android:gravity="center_horizontal"
      android:textSize="22dp"
      android:text="Radio button inside RadioGroup" />

<!--  Customized RadioButtons  -->
   <RadioGroup
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/radioGroup">

      <RadioButton
         android:id="@+id/radioMale"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:text="  Male"
         android:layout_marginTop="10dp"
         android:checked="false"
```

```
        android:textSize="20dp" />

    <RadioButton
        android:id="@+id/radioFemale"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="   Female"
        android:layout_marginTop="20dp"
        android:checked="false"
        android:textSize="20dp" />
  </RadioGroup>

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Show Selected"
    android:id="@+id/button"
    android:onClick="onclickbuttonMethod"
    android:layout_gravity="center_horizontal" />
</LinearLayout>
```

In MainActivity.java file, add:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
  Button button;
  RadioButton genderradioButton;
  RadioGroup radioGroup;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    radioGroup=(RadioGroup)findViewById(R.id.radioGroup);
  }
  public void onclickbuttonMethod(View v){
    int selectedId = radioGroup.getCheckedRadioButtonId();
    genderradioButton = (RadioButton) findViewById(selectedId);
    if(selectedId==-1){
      Toast.makeText(MainActivity.this,"Nothing selected", Toast.LENGTH_SHORT).show();
    }
    else{
      Toast.makeText(MainActivity.this,genderradioButton.getText(), Toast.LENGTH_SHORT).show();
    }
  }
}
```

Output:



ii) CheckBox: **<3M>**

in activity_main.xml add:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.checkbox.MainActivity">

    <CheckBox
        android:id="@+id/checkBox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="144dp"
        android:layout_marginTop="68dp"
        android:text="Pizza"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <CheckBox
        android:id="@+id/checkBox2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="144dp"
        android:layout_marginTop="28dp"
        android:text="Coffee"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/checkBox" />

    <CheckBox
        android:id="@+id/checkBox3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="144dp"
        android:layout_marginTop="28dp"
        android:text="Burger"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/checkBox2" />
```

```
<Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="144dp"
        android:layout_marginTop="184dp"
        android:text="Order"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/checkBox3" />

</android.support.constraint.ConstraintLayout>
```

In **MainActivity.java** file, add:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    CheckBox pizza,coffe,burger;
    Button buttonOrder;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        addListenerOnButtonClick();
    }
    public void addListenerOnButtonClick(){
        //Getting instance of CheckBoxes and Button from the activty_main.xml file
        pizza=(CheckBox)findViewById(R.id.checkBox);
        coffe=(CheckBox)findViewById(R.id.checkBox2);
        burger=(CheckBox)findViewById(R.id.checkBox3);
        buttonOrder=(Button)findViewById(R.id.button);

        //Applying the Listener on the Button click
        buttonOrder.setOnClickListener(new View.OnClickListener(){

            @Override
            public void onClick(View view) {
                int totalamount=0;
                StringBuilder result=new StringBuilder();
                result.append("Selected Items:");
                if(pizza.isChecked()){
                    result.append("\nPizza 100Rs");
                    totalamount+=100;
                }
                if(coffe.isChecked()){
                    result.append("\nCoffe 50Rs");
                    totalamount+=50;
                }
                if(burger.isChecked()){
                    result.append("\nBurger 120Rs");
                    totalamount+=120;
                }
```

```
            result.append("\nTotal: "+totalamount+"Rs");
            //Displaying the message on the toast
            Toast.makeText(getApplicationContext(), result.toString(), Toast.LENGTH_LONG).show();
        }
    });
  }
}
```

imageView: **<1M>**

in activity_main.xml add:

```
<ImageView
android:id="@+id/simpleImageView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:src="@drawable/lion" />
```

UNIT-III

6 a) Explain the procedure for creating and saving shared preferences. CO3 L2 7M

Using the SharedPreferences class, you can create named maps of name/value pairs that can be persisted across sessions and shared among application components running within the same application sandbox.

Creating and Saving Shared Preferences: **< 4M >**

To create or modify a Shared Preference, call getSharedPreferences on the current Context, passing in the name of the Shared Preference to change.

SharedPreferences mySharedPreferences = getSharedPreferences(MY_PREFS,Activity.MODE_PRIVATE);
Shared Preferences are stored within the application's sandbox, so they can be shared between an application's components but aren't available to other applications.

To modify a Shared Preference, use the SharedPreferences.Editor class. Get the Editor object by calling edit on the Shared Preferences object you want to change.

```
SharedPreferences.Editor editor = mySharedPreferences.edit();
```

Use the put<type> methods to insert or update the values associated with the specifi ed name:

```
// Store new primitive types in the shared preferences object.
        editor.putBoolean("isTrue", true);
        editor.putFloat("lastFloat", 1f);
        editor.putInt("wholeNumber", 2);
        editor.putLong("aNumber", 3l);
        editor.putString("textEntryValue", "Not Empty");
```

To save edits, call apply or commit on the Editor object to save the changes asynchronously or synchronously, respectively.

```
// Commit the changes.
editor.apply();
```

Retrieving Shared Preferences: **<3M >**

Accessing Shared Preferences, like editing and saving them, is done using the getSharedPreferences method. Use the type-safe get<type> methods to extract saved values. Each getter takes a key and a default value (used when no value has yet been saved for that key.)

// Retrieve the saved values.

```
boolean isTrue = mySharedPreferences.getBoolean("isTrue", false);
float lastFloat = mySharedPreferences.getFloat("lastFloat", 0f);
int wholeNumber = mySharedPreferences.getInt("wholeNumber", 1);
long aNumber = mySharedPreferences.getLong("aNumber", 0);
String stringPreference =mySharedPreferences.getString("textEntryValue", "");
```

You can return a map of all the available Shared Preferences keys values by calling getAll, and check for the existence of a particular key by calling the contains method.

```
Map<String, ?> allPreferences = mySharedPreferences.getAll();
boolean containsLastFloat = mySharedPreferences.contains("lastFloat");
```

6 b) Develop an android application using explicit intents. ***CO3 L6 7M***

In activity_main.xml, add: **<1M>**

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context="example.javatpoint.com.explicitintent.FirstActivity">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginEnd="8dp"
      android:layout_marginStart="8dp"
      android:layout_marginTop="8dp"
      android:text="First Activity"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintHorizontal_bias="0.454"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintRight_toRightOf="parent"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent"
      app:layout_constraintVertical_bias="0.06" />

   <Button
      android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginEnd="8dp"
      android:layout_marginStart="8dp"
      android:layout_marginTop="392dp"
      android:onClick="callSecondActivity"
      android:text="Call second activity"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

In MainActivityOne.java, add: **<2M>**

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class FirstActivity extends AppCompatActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_first);
   }
   public void callSecondActivity(View view){
      Intent i = new Intent(getApplicationContext(), SecondActivity.class);
      i.putExtra("Value1", "Android");
      i.putExtra("Value2", "Simple Tutorial");
      // Set the request code to any code you like, you can identify the
      // callback via this code
      startActivity(i);
   }
}
```

In activitytwo_main.xml, add: **<2M>**

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context="example.javatpoint.com.explicitintent.SecondActivity">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginEnd="8dp"
      android:layout_marginStart="8dp"
      android:layout_marginTop="8dp"
      android:text="Second Activity"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintHorizontal_bias="0.454"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintRight_toRightOf="parent"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent"
      app:layout_constraintVertical_bias="0.06" />

   <Button
      android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginEnd="8dp"
      android:layout_marginStart="8dp"
      android:layout_marginTop="392dp"
      android:onClick="callFirstActivity"
      android:text="Call first activity"
```

```
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

In MainActivityTwo.java, add: **<2M>**

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class SecondActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_second);
      Bundle extras = getIntent().getExtras();
      String value1 = extras.getString("Value1");
      String value2 = extras.getString("Value2");
      Toast.makeText(getApplicationContext(),"Values are:\n First value: "+value1+
            "\n Second Value: "+value2, Toast.LENGTH_LONG).show();
   }
   public void callFirstActivity(View view){
      Intent i = new Intent(getApplicationContext(), FirstActivity.class);
      startActivity(i);
   }
}
```

(OR)

Describe the following:

7 a. Using Intent Filters to service Implicit Intents. **CO3 L2 7M <3M>**

If an Intent is a request for an action to be performed on a set of data, how does Android know which application (and component) to use to service the request? Intent Filters are used to register Activities, Services, and Broadcast Receivers as being capable of performing an action on a particular kind of data.

Using Intent Filters, application components tell Android that they can service action requests from others, including components in the same, native, or third-party applications.

To register an application component as an Intent handler, use the intent-filter tag within the component's manifest node.

Using the following tags (and associated attributes) within the Intent Filter node, you can specify a component's supported actions, categories, and data:

❑ **action** Use the android:name attribute to specify the name of the action being serviced. Actions should be unique strings, so best practice is to use a naming system based on the Java package naming conventions.

❑ **category** Use the android:category attribute to specify under which circumstances the action should be serviced. Each Intent Filter tag can include multiple category tags. You can specify your own categories or use the standard values provided by Android and listed below:

❑ **DEFAULT** Set this to make a component the default action for the data values defi ned by the Intent Filter. This is also necessary for Activities that are launched using an explicit Intent.

❑ **HOME** The home Activity is the first Activity displayed when the device starts (the launch screen). By setting an Intent Filter category as home without specifying an action, you are presenting it as an alternative to the native home screen.

❑ **LAUNCHER** Using this category makes an Activity appear in the application launcher.

<2M>

❑ **data** The data tag lets you specify matches for data your component can act on; you can include several schemata if your component is capable of handling more than one. You can use any combination of the following attributes to specify the data that your component supports:

❑ android:host Specifi es a valid host name (e.g., com.google).

❑ android:mimetype Lets you specify the type of data your component is capable of handling. For example, <type android:value="vnd.android.cursor.dir/*"/> would match any Android cursor.

❑ android:path Valid —path‖ values for the URI (e.g., /transport/boats/)

❑ android:port Valid ports for the specifi ed host ❑ android:scheme Requires a particular scheme (e.g., content or http).

The following code snippet shows how to confi gure an Intent Filter for an Activity that can perform the SHOW_DAMAGE action as either a primary or alternative action. (You'll create earthquake content in the next chapter.)

<2M>

```
<activity android:name=".EarthquakeDamageViewer" android:label="View Damage">
<intent-filter>
<action android:name="com.paad.earthquake.intent.action.SHOW_DAMAGE"> </action>
<category android:name="android.intent.category.DEFAULT"/>
<category android:name="android.intent.category.ALTERNATIVE_SELECTED" />
<data android:mimeType="vnd.earthquake.cursor.item/*"/>
</intent-filter>
</activity>
```

Describe the following:

7 b. Using Intent Filters for Plug – In and extensibility. **CO3 L2 7M**

<3M>

So far you've learned how to explicitly create implicit Intents, but that's only half the story. Android lets future packages provide new functionality for existing applications, using Intent Filters to populate menus dynamically at run time.

This provides a plug-in model for your Activities that lets them take advantage of functionality you haven't yet conceived of through new application components, without your having to modify or recompile your projects.

The addIntentOptions method available from the Menu class lets you specify an Intent that describes the data that is acted upon by this menu. Android resolves this Intent and returns every action specified in the Intent Filters that matches the specifi ed data. A new Menu Item is created for each, with the text populated from the matching Intent Filters' labels.

The elegance of this concept is best explained by example. Say the data your application displays are a list of places. At the moment, the menu actions available might include "view" and "show directions to." Jump a few years ahead, and you've created an application that interfaces with your car, allowing your phone to handle driving. Thanks to the runtime menu generation, by including a new Intent Filter — with a DRIVE_CAR action — within the new Activity's node, Android will automagically add this action as a new Menu Item in your earlier application.

Runtime menu population provides the ability to retrofi t functionality when you create new components capable of performing actions on a given type of data. Many of Android's native applications use this functionality, giving you the ability to provide additional actions to native Activities.

Supplying Anonymous Actions to Applications: **<2M>**

To make actions available for other Activities, publish them using intent-filter tags within their manifest nodes.

The Intent Filter describes the action it performs and the data upon which it can be performed. The latter will be used during the Intent resolution process to determine when this action should be available.

The category tag must be either or both ALTERNATIVE and SELECTED_ALTERNATIVE. The text used by Menu Items is specifi ed by the android:label attribute.

The following XML shows an example of an Intent Filter used to advertise an Activity's ability to nuke moon bases from orbit.

<activity android:name=".NostromoController">
<intent-filter android:label="Nuke From Orbit">
<action android:name="com.pad.nostromo.NUKE_FROM_ORBIT"/>
<data android:mimeType="vnd.moonbase.cursor.item/*"/>
<category android:name="android.intent.category.ALTERNATIVE"/>
<category android:name="android.intent.category.SELECTED_ALTERNATIVE" />
</intent-filter> </activity>

The Content Provider and other code needed for this example to run aren't provided; in the following sections, you'll see how to write the code that will make this action available dynamically from another Activity's menu.

Incorporating Anonymous Actions in Your Activity's Menu: **<2M>**

To add menu options to your menus at run time, you use the addIntentOptions method on the menu object in question, passing in an Intent that specifi es the data for which you want to provide actions. Generally, this will be handled within your Activity's onCreateOptionsMenu or onCreateContextMenu handlers. also

The Intent you create will be used to resolve components with Intent Filters that supply actions for the data you specify. The Intent is being used to fi nd actions, so don't assign it one; it should only specify the data on which to perform actions. You should specify the category CATEGORY_SELECTED_ALTERNATIVE. of the action, either

The skeleton code for creating an Intent for menu-action resolution is shown below:
Intent intent = new Intent();
intent.setData(MyProvider.CONTENT_URI);
CATEGORY_ALTERNATIVE or intent.addCategory(Intent.CATEGORY_ALTERNATIVE);
Pass this Intent into addIntentOptions on the menu you wish to populate, as well as any option fl ags, the name of the calling class, the menu group to use, and menu ID values. You can also specify an array of Intents you'd like to use to create additional Menu Items.

UNIT-IV

8. Develop an Android program for maintaining a To – Do List database using SQLite database.
CO4 L6 14M

ToDoListSQLHelper.java ***<3M>***

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.provider.BaseColumns;

public class TodoListSQLHelper extends SQLiteOpenHelper {
   public static final String DB_NAME = "com.androidtodo";
   public static final String TABLE_NAME = "TODO_LIST";
   public static final String COL1_TASK = "todo";
   public static final String _ID = BaseColumns._ID;

   public TodoListSQLHelper(Context context) {
```

```
        //1 is todo list database version
        super(context, DB_NAME, null, 1);
    }

    @Override
    public void onCreate(SQLiteDatabase sqlDB) {
        String createTodoListTable = "CREATE TABLE " + TABLE_NAME + " ( _id INTEGER PRIMARY
KEY AUTOINCREMENT, " +
                COL1_TASK + " TEXT)";
        sqlDB.execSQL(createTodoListTable);
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqlDB, int i, int i2) {
        sqlDB.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(sqlDB);
    }
}
```

ToDoActivity.java: **<11M>**

```
import android.app.AlertDialog;
import android.app.ListActivity;
import android.content.ContentValues;
import android.content.DialogInterface;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.ListAdapter;
import android.widget.SimpleCursorAdapter;
import android.widget.TextView;

public class TodoActivity extends ListActivity {
    private ListAdapter todoListAdapter;
    private TodoListSQLHelper todoListSQLHelper;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_todo);
        updateTodoList();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.todo, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
```

```
            case R.id.action_add_task:
                AlertDialog.Builder todoTaskBuilder = new AlertDialog.Builder(this);
                todoTaskBuilder.setTitle("Add Todo Task Item");
                todoTaskBuilder.setMessage("describe the Todo task...");
                final EditText todoET = new EditText(this);
                todoTaskBuilder.setView(todoET);
                todoTaskBuilder.setPositiveButton("Add Task", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialogInterface, int i) {
                        String todoTaskInput = todoET.getText().toString();
                        todoListSQLHelper = new TodoListSQLHelper(TodoActivity.this);
                        SQLiteDatabase sqLiteDatabase = todoListSQLHelper.getWritableDatabase();
                        ContentValues values = new ContentValues();
                        values.clear();

                        //write the Todo task input into database table
                        values.put(TodoListSQLHelper.COL1_TASK, todoTaskInput);
                        sqLiteDatabase.insertWithOnConflict(TodoListSQLHelper.TABLE_NAME, null, values,
SQLiteDatabase.CONFLICT_IGNORE);

                        //update the Todo task list UI
                        updateTodoList();
                    }
                });
                todoTaskBuilder.setNegativeButton("Cancel", null);
                todoTaskBuilder.create().show();
                return true;

            default:
                return false;
        }
    }

    //update the todo task list UI
    private void updateTodoList() {
        todoListSQLHelper = new TodoListSQLHelper(TodoActivity.this);
        SQLiteDatabase sqLiteDatabase = todoListSQLHelper.getReadableDatabase();

        //cursor to read todo task list from database
        Cursor cursor = sqLiteDatabase.query(TodoListSQLHelper.TABLE_NAME,
                new String[]{TodoListSQLHelper._ID, TodoListSQLHelper.COL1_TASK},
                null, null, null, null, null);

        //binds the todo task list with the UI
        todoListAdapter = new SimpleCursorAdapter(
                this,
                R.layout.todotask,
                cursor,
                new String[]{TodoListSQLHelper.COL1_TASK},
                new int[]{R.id.todoTaskTV},
                0
        );
        this.setListAdapter(todoListAdapter);
    }
```

```
    //closing the todo task item
    public void onDoneButtonClick(View view) {
        View v = (View) view.getParent();
        TextView todoTV = (TextView) v.findViewById(R.id.todoTaskTV);
        String todoTaskItem = todoTV.getText().toString();

        String deleteTodoItemSql = "DELETE FROM " + TodoListSQLHelper.TABLE_NAME +
                " WHERE " + TodoListSQLHelper.COL1_TASK + " = '" + todoTaskItem + "'";

        todoListSQLHelper = new TodoListSQLHelper(TodoActivity.this);
        SQLiteDatabase sqlDB = todoListSQLHelper.getWritableDatabase();
        sqlDB.execSQL(deleteTodoItemSql);
        updateTodoList();
    }
}
```

(OR)

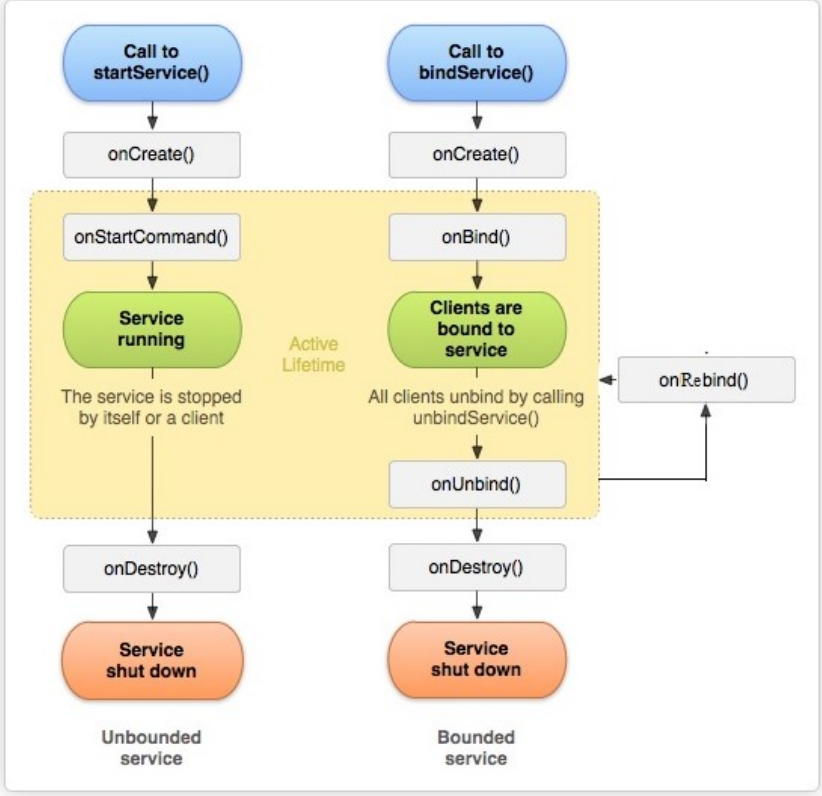
9. Illustrate following

a. Services b. options menu and context menu. ***CO4 L2 14M***

9 a: Services:

A service is an application in Android that runs in the background without needing to interact with the user. For example, while using an application, you may want to play some background music at the same time.

Life cycle of a Android Service: **<2M>**



Life cycle of a Android Service: **<2M>**

Unbounded service:

A service is "started" when an application component (such as an activity) starts it by calling startService(). Once started, a service can run in the background indefinitely (unbounded), even if the component that started it is destroyed. Usually, a started service performs a single operation and does not return a result to the caller. For example, it might download or upload a file over the network. When the operation is done, the service should stop itself.

Bounded Service:

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). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

Methods used in Bounded Service life cycle: **<3M>**

startService(Intent Service)
This you must call to start un-bounded service

onCreate()
This method is Called when the service is first created

onStartCommand(Intent intent, int flags, int startId)
This method is called when service is started

onBind(Intent intent)
This method you must call if you want to bind with activity

onUnbind(Intent intent)
This method is Called when the service will un-binded from activity

onRebind(Intent intent)
This method is called when you want to Re-bind service after calling un-bind method

onDestroy()
This method is called when The service is no longer used and is being destroyed

9 b. Illustrate options menu and context menu. **<7M>**

OptionsMenu: **<4M>**

Android Option Menus are the primary menus of android. They can be used for settings, search, delete item etc.

Here, we are going to see two examples of option menus. First, the simple option menus and second, options menus with images.

Here, we are inflating the menu by calling the **inflate()** method of **MenuInflater** class. To perform event handling on menu items, you need to override **onOptionsItemSelected()** method of Activity class.

Let's see the simple option menu example that contains three menu items.

In activity_main.xml, add:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.optionmenu.MainActivity">
```

```
    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay" />

    </android.support.design.widget.AppBarLayout>
    <include layout="@layout/content_main" />
</android.support.design.widget.CoordinatorLayout>
```

context_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context="example.javatpoint.com.optionmenu.MainActivity"
    tools:showIn="@layout/activity_main">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

menu_main.xml

It contains three items as show below. It is created automatically inside the res/menu directory.

```
 <menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="example.javatpoint.com.optionmenu.MainActivity">

    <item  android:id="@+id/item1"
        android:title="Item 1"/>
    <item  android:id="@+id/item2"
        android:title="Item 2"/>
    <item  android:id="@+id/item3"
        android:title="Item 3"
        app:showAsAction="withText"/>
</menu>
```

In MainActivity.java, add:

```
 import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
       int id = item.getItemId();
        switch (id){
            case R.id.item1:
                Toast.makeText(getApplicationContext(),"Item 1 Selected",Toast.LENGTH_LONG).show();
                return true;
            case R.id.item2:
                Toast.makeText(getApplicationContext(),"Item 2 Selected",Toast.LENGTH_LONG).show();
                return true;
            case R.id.item3:
                Toast.makeText(getApplicationContext(),"Item 3 Selected",Toast.LENGTH_LONG).show();
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

ContextMenu: **<3M>**

```
import android.graphics.Color;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.MenuItem;
import android.view.View;
import android.widget.RelativeLayout;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;
```

```
public class MainActivity extends AppCompatActivity {
    TextView textView;
    RelativeLayout relativeLayout;

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

        // Link those objects with their respective id's that we have given in .XML file
        textView = (TextView) findViewById(R.id.textView);
        relativeLayout = (RelativeLayout) findViewById(R.id.relLayout);

        // here you have to register a view for context menu you can register any view
          // like listview, image view, textview, button etc
        registerForContextMenu(textView);

    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo
menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);
        // you can set menu header with title icon etc
        menu.setHeaderTitle("Choose a color");
        // add menu items
        menu.add(0, v.getId(), 0, "Yellow");
        menu.add(0, v.getId(), 0, "Gray");
        menu.add(0, v.getId(), 0, "Cyan");
    }

    // menu item select listener
    @Override
    public boolean onContextItemSelected(MenuItem item) {
        if (item.getTitle() == "Yellow") {
            relativeLayout.setBackgroundColor(Color.YELLOW);
        } else if (item.getTitle() == "Gray") {
            relativeLayout.setBackgroundColor(Color.GRAY);
        } else if (item.getTitle() == "Cyan") {
            relativeLayout.setBackgroundColor(Color.CYAN);
        }
        return true;
    }
}
```

Signature of the Internal Examiner | **Signature of the External Examiner** | **Signature of the HOD**