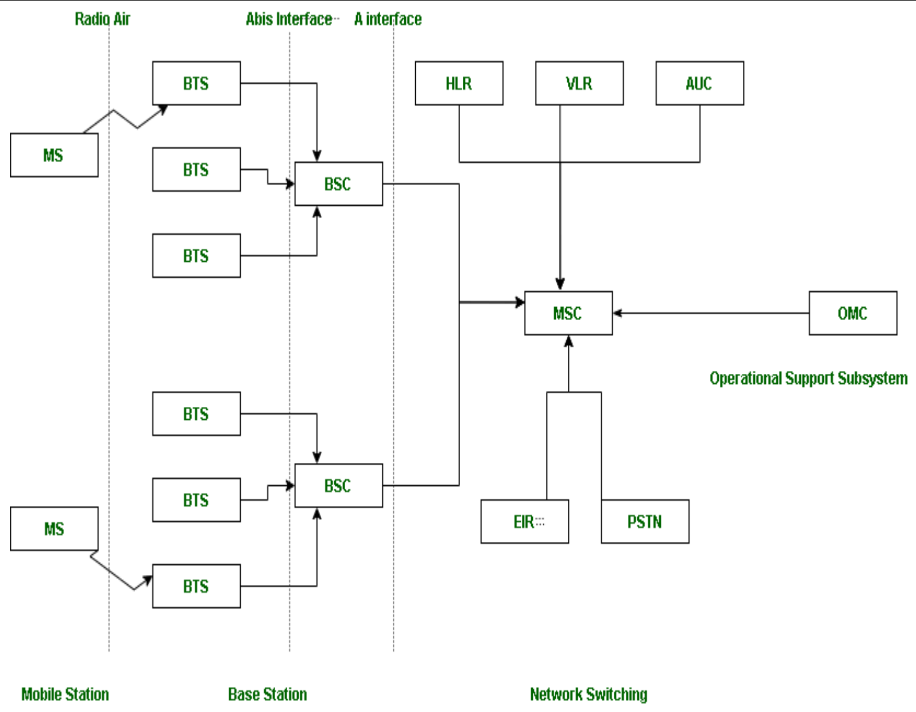
CMR INSTITUTE OF TECHNOLOGY	USN	

VTU November 2024				
Sub:	**Mobile Application Development**	**Code :**	**22MCA263**	
Answer Key		**Marks**	**OBE**	
			RBT	CO
Q1(a)	**What is Mobile Computing? Explain important mobile computing functionalities.** Mobile Computing Mobile computing(computer technology) is a technology that allows anytime, anywhere and everywhere computing. It allows user to perform any task from anywhere using a computing device. Communication is spread over both Wired and wireless media. Set of distributed computing systems participate, connect and synchronize through mobile communication protocol. Introduction of Mobile Computing Mobile Computing is a technology that provides an environment that enables users to transmit data from one device to another device without the use of any physical link or cables. In other words, you can say that mobile computing allows transmission of data, voice and video via a computer or any other wireless-enabled device without being connected to a fixed physical link. In this technology, data transmission is done wirelessly with	10	L2	CO1

	the help of wireless devices such as mobiles, laptops etc. The concept of Mobile Computing can be divided into three parts: ○ Network Infrastructure ○ Mobile Hardware ○ Mobile Software Mobile Hardware Mobile hardware consists of mobile devices or device components that can be used to receive or access the service of mobility. Examples of mobile hardware can be smart phones, laptops, portable PCs, tablet PCs, Personal Digital Assistants, etc. Mobile Software Mobile software is a program that runs on mobile hardware. This is designed to deal capably with the characteristics and requirements of mobile applications. This is the operating system for the appliance of mobile devices. Network Infrastructure Mobile computing relies heavily on network infrastructure to facilitate communication and data exchange. Cellular networks, including 3G, 4G, and the emerging 5G technology, provide the backbone for mobile connectivity. These networks enable voice calls, text messaging, and high-speed internet access, allowing users to stay connected virtually anywhere. Additionally, Wi-Fi and Bluetooth technologies offer local connectivity options, supporting data transfer, device pairing, and internet access			
Q1(b)	**With a neat diagram, explain GSM architecture.**	10	L2	CO1



GSM Architecture

The GSM (Global System for Mobile Communications) system architecture outlines the framework and components that enable mobile
communication services. It defines how mobile devices communicate with each other and with external networks. The following
are the primary components of the GSM architecture:

● The network switching system (NSS)
● The mobile station (MS)
● The base station system (BSS)
● The operations and support system (OSS)

1. Mobile Station (MS)

○ Definition: The mobile device used by subscribers, consisting of the physical device (phone) and the Subscriber Identity Module
(SIM) card.

○ Functions:

■ Communicates with the Base Station Subsystem (BSS) via the air interface.
■ Manages voice and data communication.

2. Base Station Subsystem (BSS)

○ Components:

■ Base Transceiver Station (BTS): Transmits and receives radio signals to/from mobile devices within its coverage area.
■ Base Station Controller (BSC): Manages multiple BTSs, handles call setup, handovers, and radio channel allocation.

○ Functions:

	■ Controls radio frequency (RF) resources. ■ Manages handovers between BTSs. ■ Handles encryption and decryption of voice and data. 3. Network Switching Subsystem (NSS) ○ Components: ■ Mobile Switching Center (MSC): Manages call processing and switching between different network interfaces. ■ Home Location Register (HLR): Central database storing permanent subscriber information, including location and service profiles. ■ Visitor Location Register (VLR): Temporarily stores information about subscribers currently located in the area served by the MSC. ■ Authentication Center (AuC): Provides authentication and encryption parameters to ensure secure communication. 4. Operation and Support Subsystem (OSS) ○ Components: ■ Equipment Identity Register (EIR): Database storing information about mobile devices' identities (IMEI) to prevent theft and unauthorized use. ■ Operation and Maintenance Centre (OMC): An OMC monitors and controls all other network entities through the 0 interface. The OMC also includes management of status reports, traffic monitoring, subscriber security management, and accounting and billing. Additional Elements ○ Short Message Service Center (SMSC): Handles the storage and delivery of SMS messages. ○ Gateway MSC (GMSC): Interfaces with other networks (e.g., PSTN, other mobile networks) for call routing. ○ Interworking Function (IWF): Interfaces with external networks (e.g., ISDN) to support data services.			
Q2(a)	**Discuss the different mobile services provided by mobile computing.** **1. Communication Services** These services enable seamless communication between devices and users. Examples include: ● **Voice Calls**: Traditional telephony services over cellular networks. ● **Video Calls**: Real-time video communication via mobile networks or apps like Zoom and Skype.	10	L2	CO1

- **Messaging Services**: Text-based communication through SMS, MMS, or instant messaging platforms like WhatsApp, Telegram, and iMessage.
- **Email Services**: Access to email accounts through mobile applications or web interfaces.

2. Internet Access Services

Mobile computing enables continuous internet access for a range of activities:

- **Mobile Browsing**: Browsing the internet using web browsers optimized for mobile devices.
- **Social Media**: Accessing platforms like Facebook, Instagram, and Twitter for interaction and content sharing.
- **Streaming Services**: Using apps like YouTube, Netflix, and Spotify to stream video or audio content.
- **Online Gaming**: Real-time gaming experiences enabled through mobile networks.

3. Data Synchronization Services

These services ensure that data is consistent across devices:

- **Cloud Storage**: Services like Google Drive, Dropbox, and iCloud allow users to store and access files from anywhere.
- **Backup Services**: Automatically backing up data, such as photos, contacts, and messages, to cloud servers.
- **Cross-Device Sync**: Synchronizing data like calendars, emails, and tasks across multiple devices.

4. Location-Based Services (LBS)

These services leverage GPS or cellular triangulation to provide location-specific solutions:

- **Navigation**: Real-time maps and directions through apps like Google Maps and Waze.
- **Geotagging**: Associating geographic information with photos, videos, or posts.
- **Ride-Hailing**: Services like Uber and Lyft that connect drivers and passengers based on their locations.
- **Local Recommendations**: Apps like Yelp and TripAdvisor offer location-based suggestions for restaurants, hotels, and events.

5. Entertainment Services

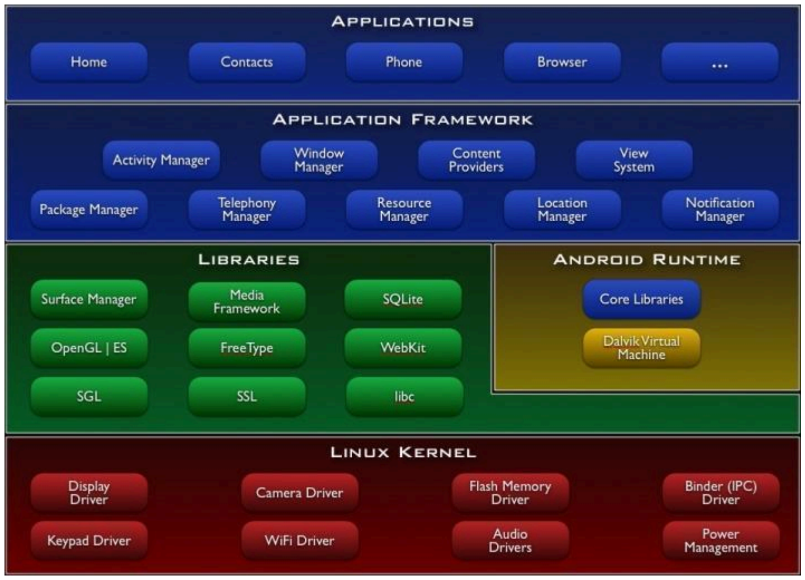
Mobile computing provides a host of entertainment options:

- **Mobile Gaming**: Platforms like PUBG Mobile, Candy Crush, and Clash of Clans.

	● **Music Streaming**: Apps like Spotify, Apple Music, and Pandora. ● **Video Streaming**: Platforms like Netflix, Amazon Prime Video, and Hulu. ● **Augmented Reality (AR)**: Apps like Pokémon Go or AR-based shopping and gaming experiences. **6. Financial and Payment Services** Mobile computing has revolutionized financial transactions: ● **Mobile Banking**: Accessing bank accounts and performing transactions via mobile apps. ● **Mobile Wallets**: Digital payment systems like Google Pay, Apple Pay, and PayPal. ● **E-Commerce**: Shopping online through platforms like Amazon, eBay, and Alibaba. ● **Cryptocurrency Apps**: Managing digital currencies like Bitcoin and Ethereum. **7. Healthcare and Fitness Services** These services promote health monitoring and management: ● **Telemedicine**: Remote consultations with healthcare professionals via mobile apps. ● **Fitness Tracking**: Apps like Fitbit, MyFitnessPal, and Strava to monitor activity, nutrition, and health metrics. ● **Medication Reminders**: Apps that notify users to take prescribed medicines. **8. Business and Productivity Services** Mobile computing enhances workplace efficiency: ● **Collaboration Tools**: Apps like Slack, Microsoft Teams, and Zoom for teamwork and communication. ● **Task Management**: Tools like Trello, Asana, and Notion to organize tasks and projects. ● **Document Editing**: Apps like Microsoft Word, Google Docs, and Adobe Acrobat for creating and editing documents on the go.			
Q2(b)	**Discuss the merit and demerits of Smartphone operating system.** Merits of Smartphone Operating Systems 1. Enhanced Functionality ● **Rich Ecosystem of Apps**: Smartphone OS platforms like Android and iOS offer access to millions of apps for productivity, entertainment, health, and more. ● **User-Friendly Interfaces**: Most OSes provide intuitive and visually appealing interfaces, making devices easy to use for all age groups.	10	L2	CO1

	● **Multi-Tasking**: Operating systems allow users to run multiple applications simultaneously, enhancing productivity. 2. Regular Updates ● **Feature Enhancements**: Frequent updates bring new features and improvements, keeping the system relevant. ● **Security Improvements**: Regular patches protect devices against the latest security threats. 3. Seamless Integration ● **Cross-Device Compatibility**: Many OS platforms (e.g., iOS with macOS) enable seamless integration between devices like smartphones, tablets, laptops, and smartwatches. ● **IoT Support**: Smartphones can act as central controllers for Internet of Things (IoT) devices like smart homes and wearables. 4. Customization and Personalization ● **Android**: Offers extensive customization options, including widgets, themes, and ROMs. ● **iOS**: Provides limited but polished customization features that ensure a consistent and smooth experience. 5. Accessibility Features ● Built-in tools like screen readers, magnifiers, and voice control make smartphones accessible to people with disabilities. 6. Advanced Hardware Optimization ● iOS is tightly integrated with Apple's hardware, ensuring optimized performance and efficient battery usage. ● Android supports a wide range of devices, offering flexibility for different hardware specifications. Demerits of Smartphone Operating Systems 1. Fragmentation ● **Android**: Due to the large variety of manufacturers, Android devices often suffer from fragmented updates, with many users not receiving timely security patches or OS updates. ● **iOS**: While less fragmented, older iPhone models eventually stop receiving updates. 2. Security Concerns ● **Open Nature of Android**: Allows third-party app stores, which can lead to the installation of malicious apps. ● **iOS Jailbreaking**: Although rare, jailbreaking can expose iOS devices to security vulnerabilities. 3. Cost and Accessibility			

	● **Expensive Hardware for iOS**: The exclusive nature of iOS means users must invest in costly Apple devices. ● **Resource-Intensive Features**: Newer versions of some OSes may be too demanding for older hardware, leading to performance issues. 4. Limited Customization (for Some OSes) ● **iOS**: While stable and polished, it has limited customization options compared to Android. ● **Windows Phone (Discontinued)**: Historically offered minimal app support and lacked customization. 5. Dependence on Ecosystem ● Users often feel locked into a specific ecosystem (e.g., Apple or Google) due to cross-platform dependencies, making it hard to switch to a different OS. 6. App Compatibility Issues ● **Android**: Some apps may not run optimally on all devices due to hardware diversity. ● **iOS**: Proprietary app guidelines can limit developers' flexibility, leading to fewer experimental apps compared to Android. 7. Privacy Concerns ● **Data Collection**: Both iOS and Android have been criticized for collecting user data, sometimes leading to concerns over privacy. ● **Third-Party Apps**: Permissions required by apps can sometimes compromise user privacy. 8. Performance Limitations ● **Battery Drain**: Background processes and poorly optimized apps can lead to excessive battery consumption. ● **Lag and Crashes**: In low-end devices, the OS may struggle to handle newer apps and features effectively.			
Q3(a)	**Discuss the android software stack.** Understanding the Android Software Stack Android is structured in the form of a software stack comprising applications, an operating system, run-time environment, middleware, services and libraries. Each layer of the stack, and the corresponding elements within each layer, are tightly integrated and carefully tuned to provide the optimal application development and execution environment for mobile devices.	10	L2	CO1

	The Android software stack consists of four main layers:  APPLICATIONS Home Contacts Phone Browser ... APPLICATION FRAMEWORK Activity Manager Window Manager Content Providers View System Package Manager Telephony Manager Resource Manager Location Manager Notification Manager LIBRARIES Surface Manager Media Framework SQLite OpenGL | ES FreeType WebKit SGL SSL libc ANDROID RUNTIME Core Libraries Dalvik Virtual Machine LINUX KERNEL Display Driver Camera Driver Flash Memory Driver Binder (IPC) Driver Keypad Driver WiFi Driver Audio Drivers Power Management **1) Linux kernel** It is the heart of android architecture that exists at the root of android architecture. Linux kernel is responsible for device drivers, power management, memory management,device management and resource access. The kernel on which Android is based contains device drivers for various hardware components of an Android device, including Display, Camera, Keypad, Wifi, Memory, and Audio. **2) Native Libraries** The next layer on top of the Linux kernel is the libraries that implement different Android features.: A. Freetype library-Responsible for font support. B. SQLite library-Provides database support C. Surface Manager library-Provides graphics libraries that include SGL and OpenGL. D. Open GL(graphics library):This cross-language, cross-platform application program interface (API) is used to produce 2D and 3D computer graphics. e. WebKit: This open source web browser engine provides all the functionality to display web content and to simplify page loading. f. Media frameworks: These libraries allow you to play and record audio and video. g. Secure Socket Layer (SSL): These libraries are there for Internet security. **3) Android Runtime** it provides a set of core Android libraries and a Dalvik virtual machine that enable developers to write Android applications using java and (the Android RunTime). In android runtime, there are core libraries and DVM (Dalvik Virtual Machine) which is responsible to run android application. DVM is like JVM but it is optimized for mobile devices. It consumes less memory and provides fast performance. **4) Android Framework** On the top of Native libraries and android runtime, there is android framework. Android framework includes Android API's such as UI (User Interface), telephony, resources,			

locations, Content Providers (data) and package managers. It provides a lot of classes and interfaces for android application development.
provides the classes that enable application developers to develop[android applications].
Activity Manager:
It manages the activity lifecycle and the activity stack.
B. Telephony Manager:
It provides access to telephony services as related subscriber information, such as
phone numbers.
C. View System:
It builds the user interface by handling the views and layouts.
D. Location manager:
It finds the device's geographic location.

5). Application layer

Displays the application developed and downloaded by users.
On the top of android framework, there are applications.
All applications such as home,contact, settings, games, browsers are using android framework that uses androidruntime and libraries.
Android runtime and native libraries are using linux kernal.

Q3(b)	**Illustrate the different components of android application design.**	10	L2	CO1

1. Activities

An **Activity** is a single, focused piece of user interface (UI) in an Android app. It represents a screen or window where the user interacts with the app. Every app in Android consists of at least one activity.

- **MainActivity**: The entry point to your application, usually the first screen a user sees when the app is launched.
- **Multiple Activities**: An app can have multiple activities for different user interactions (e.g., one for the main screen, another for settings, etc.).

```
public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
}
```

2. Services

A **Service** is a component that runs in the background to perform long-running tasks without interacting with the user interface. Services continue to run even if the user switches to another app or the screen turns off.

- **Types of Services**:

- **Started Service**: A service that runs until it completes or is stopped.
- **Bound Service**: A service that provides a client-server interface to communicate with other components.

3. Broadcast Receivers

A **BroadcastReceiver** is a component used to listen for and handle broadcast messages from other applications or the system itself. These broadcasts can notify the app about events such as incoming messages, connectivity changes, or system updates.

- **Example**: You can use a broadcast receiver to listen for a change in network connectivity or to receive messages.

```
public class MyBroadcastReceiver extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
      // Handle the broadcast
   }
}
```

4. Content Providers

A **ContentProvider** allows apps to share data with other apps. It acts as an intermediary between an app's data (stored in a database or files) and other apps that need access to that data.

- **Example**: A contact content provider allows apps to access and manipulate contact information.

5. Views & ViewGroups

Android UI is built using **Views** and **ViewGroups**.

- **View**: A single UI element, like a button, text field, or image. It represents a single visual object.
- **ViewGroup**: A container that holds and arranges multiple views. Examples include LinearLayout, RelativeLayout, and FrameLayout.

6. Resources (XML, Drawable, Strings, etc.)

Resources in Android include various types of data that the app uses, such as:

- **Layouts** (res/layout/): XML files describing the UI layout.
- **Drawables** (res/drawable/): Image files or XML files that define shapes, colors, etc.
- **Strings** (res/values/strings.xml): String resources used throughout the app.
- **Colors** (res/values/colors.xml): Color definitions for UI elements.

	7. AndroidManifest.xml The **AndroidManifest.xml** is a crucial file that defines essential information about the app, including: ● The **app's components** (activities, services, broadcast receivers, etc.). ● Required **permissions** for accessing system features like the internet, camera, etc. ● **App configurations** like theme, package name, and version. **8. Intents** An **Intent** is used to perform actions or communicate between components. Intents can start activities, services, or broadcast receivers. ● **Explicit Intent**: Specifies the component to start. ● **Implicit Intent**: Specifies the action to be performed, and the system decides which component to use. **9. Fragments** A **Fragment** is a modular section of an activity that can be used in multiple activities. Fragments represent portions of a UI or behavior, and they can be reused within different activities. ● A fragment can exist within an activity and have its own lifecycle. ● It can be dynamically added, replaced, or removed at runtime. **10. App Widgets** **Widgets** are small views that can be added to the home screen or lock screen. They provide quick access to app content and functionality. ● Widgets can show data like weather, time, or news. ● You can define a widget layout, update it at intervals, and handle user interaction.			
Q4(a)	**Explain the process of installing Android SDK.** Installing the Android SDK For developing Android applications that you can publish on the Google play marketplace, you need to install the following four applications ● The Java Development kit(JDK) can be downloaded from www.oracle.com/technetwork/java/javase/downloads/index.html ● The Eclipse IDE can be downloaded from https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2024-06/R/eclipse-inst-jre-win64.exe ● The Android platform SDK Starter Package can be download from http://developer.android.com/sdk/index.html ● The Android Development Tools(ADT) Plug-in can be downloaded from ● The Java Development kit(JDK)	06	L2	CO2

Note that, The Android SDK makes use of the Java SE Development Kit (JDK). Hence, before installing Eclipse etc, install JDK from www.oracle.com/technetwork/java/javase/downloads/index.html

1. **Eclipse IDE**

The first step towards developing any applications is obtaining the integrated development environment (IDE). For Android, the recommended IDE is Eclipse, a multi-language software development environment featuring an extensible plug-in system.

It can be used to develop various types of applications, using languages such as Java, Ada, C, C++, COBOL, Python, etc. For Android development, you should download the Eclipse IDE for Java EE Developers (www.eclipse.org/downloads/packages/eclipse-ide-java-eedevelopers/heliossr1). Once the Eclipse IDE is downloaded, unzip its content (the eclipse folder) into a folder, say C:\Android\.

1. **Android SDK**

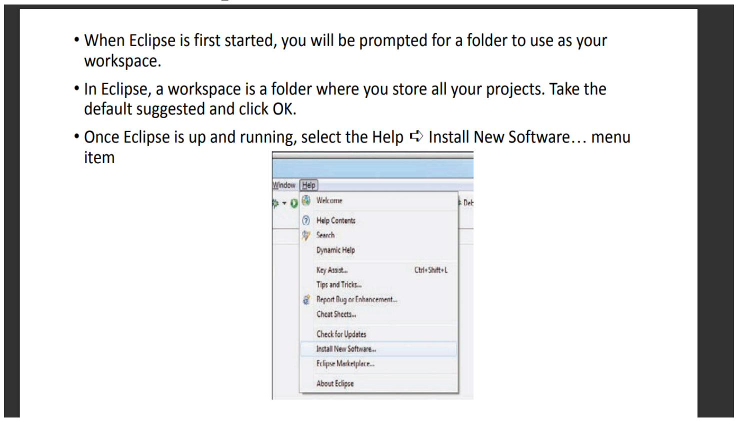
The Android SDK contains a debugger, libraries, an emulator, documentation, sample code, and tutorials. You can download the Android SDK from http://developer.android.com/sdk/index.html. Once the SDK is downloaded, unzip its content (the android-sdk-windows folder) into the C:\Android\ folder, or whatever name you have given to the folder you just created.

2. **Android Development Tools (ADT)**

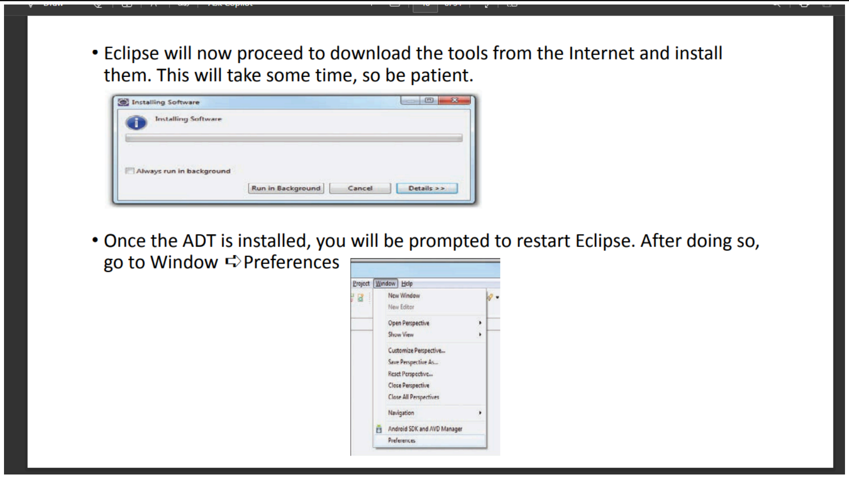
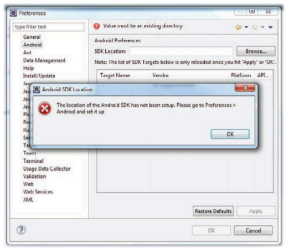
The ADT plug-in for Eclipse is an extension to the Eclipse IDE that supports the creation and debugging of Android applications. Using the ADT, you will be able to do the following in Eclipse:

- Create new Android application projects.
- Access the tools for accessing your Android emulators and devices.
- Compile and debug Android applications.
- Export Android applications into Android Packages (APK).
- Create digital certificates for code-signing your APK.

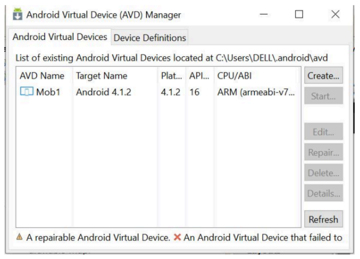
To install the ADT, first launch Eclipse by double-clicking on the eclipse.exe file located in the eclipse folder.



- When Eclipse is first started, you will be prompted for a folder to use as your workspace.
- In Eclipse, a workspace is a folder where you store all your projects. Take the default suggested and click OK.
- Once Eclipse is up and running, select the Help ➪ Install New Software… menu item

	 • Eclipse will now proceed to download the tools from the Internet and install them. This will take some time, so be patient. Installing Software Installing Software Always run in background Run in Background · Cancel · Details >> • Once the ADT is installed, you will be prompted to restart Eclipse. After doing so, go to Window ➪ Preferences Project · Window · Help New Window New Editor Open Perspective Show View Customize Perspective... Save Perspective As... Reset Perspective... Close Perspective Close All Perspectives Navigation Android SDK and AVD Manager Preferences • In the Preferences window that appears, select Android. You will see an error message saying that the SDK has not been set up. Click OK to dismiss it. • Enter the location of the Android SDK folder. In this example, it would be C:\Android\android-sdk-windows. Click OK.  Preferences Value must be an existing directory Android Preferences SDK Location: Note: The list of SDK Targets below is only reloaded once you hit 'Apply' or 'OK'. Android SDK Location The location of the Android SDK has not been setup. Please go to Preferences > Android and set it up OK			
Q4(b)	**Write the steps to create Android Virtual Device(AVD).** 1. **Creating Android Virtual Devices (AVD)** An AVD is an emulator instance that enables you to model an actual device. Each AVD consists of a hardware profile, a mapping to a system image, as well as emulated storage, such as a secure digital (SD) card. One can create many AVDs in order to test your applications with several different configurations. This testing is important to confirm the behavior of your application when it is run on different devices with varying capabilities. To create an AVD, go to Windows Menu and choose "Android Virtual Device Manager". Then give appropriate name, device, memory required etc. for the application. Once the new AVD is created, it can be used for different applications further.	06	L1	CO2

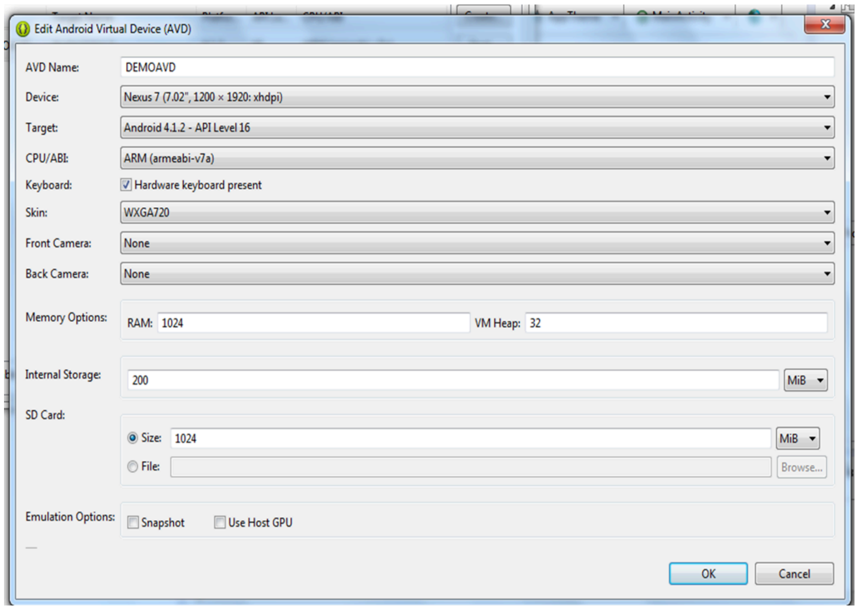
Creating Android Virtual Devices


Android Virtual Device (AVD) Manager

List of existing Android Virtual Devices located at C:\Users\DELL\.android\avd

AVD Name	Target Name	Plat...	API...	CPU/ABI
Mob1	Android 4.1.2	4.1.2	16	ARM (armeabi-v7...

⚠ A repairable Android Virtual Device. ✗ An Android Virtual Device that failed to


Edit Android Virtual Device (AVD)

AVD Name: DEMOAVD
Device: Nexus 7 (7.02", 1200 × 1920: xhdpi)
Target: Android 4.1.2 - API Level 16
CPU/ABI: ARM (armeabi-v7a)
Keyboard: ✓ Hardware keyboard present
Skin: WXGA720
Front Camera: None
Back Camera: None
Memory Options: RAM: 1024 VM Heap: 32
Internal Storage: 200 MiB
SD Card: ● Size: 1024 MiB ○ File:
Emulation Options: ☐ Snapshot ☐ Use Host GPU

Fields of ADT

- Name-used to specify the name of AVD
- Target-used to specify target API
- CPU/ABI-determines the processor that we want to emulate on our device
- SD Card-used for extending the storage capacity of the device
- Snapshot-start it from the last saved snapshot
- Skin-used for setting screen size
- Hardware-used to set properties of hardware in target device.

Q4(c)	**Describe the steps required to develop the "Hello world" application using source code.** 1. **Creating Your First Android Applications** Following are the steps involved in creating any Android application: 1. Using Eclipse, create a new project by selecting File ➪ New ➪ Android Applicationproject. 164421840. Name the Android project suitably, say *HelloWorld*.	08	L2	CO2

	In the Package Explorer (located on the left of the Eclipse IDE), expand the HelloWorld project by clicking on the various arrows displayed to the left of each item in the project. In the **res/layout** folder, double-click the **activity_main.xml** file. The **activity_main.xml** file defines the user interface (UI) of your application. The default view is the ***Layout view***, which lays out the activity graphically. To modify the UI, click the **activity_main.xml** tab located at the bottom.
	4. Add the following code in bold to the activity_main.xml file.
	`<LinearLayout xmlns:android=”http://schemas.android.com/apk/res/android”`
	`android:orientation=”vertical”`
	`android:layout_width=”fill_parent”`
	`android:layout_height=”fill_parent” >`
	`<TextView android:layout_width=”fill_parent”`
	`android:layout_height=”wrap_content”`
	`android:text=”@string/helloworld” />`
	`    </LinearLayout>`
	5. To save the changes made to your project, press Ctrl+s.
	6. You are now ready to test your application on the Android Emulator. Select the project name in Eclipse and press F11. You will be asked to select a way to debug the application. Select
	required Android Application and click OK.
	7. The Android Emulator will now be started (if the emulator is locked, you need to slide the unlock button to unlock it first).
	8. Click the Home button (the house icon in the lower-left corner above the keyboard) so that it
	now shows the Home screen.
	9. Click the application Launcher icon to display the list of applications installed on the device.
	
	Note that the HelloWorld application is now installed in the application launcher.
	(Note: In Step 6, include the steps of creating AVD).
	In Android, an Activity is a window that contains the user interface of your applications.
	An application can have zero or more activities; in this example, the application contains one activity: MainActivity. This MainActivity is the entry point of the application, which is displayed when the application is started. When you debug the application on the Android Emulator, the application is automatically
	installed on the emulator.			

Q5(a)	Define intent. Develop an android application to simple calculator to read 2 numbers and perform addition and subtraction. **Definition of Intent in Android** In Android, an **Intent** is a messaging object used to request an action from another app component. Intents can be used to: 1. Start an activity. 2. Start a service. 3. Deliver a broadcast message.	10	L1,L3	CO3

There are two main types of intents:

- **Explicit Intent**: Specifies the target component directly (e.g., starting a specific activity).
- **Implicit Intent**: Specifies an action to be performed, and the system determines which components can handle the request.

Android Application: Simple Calculator
Steps to Create the Application

The app will:

- Allow users to input two numbers.
- Perform addition and subtraction.
- Display the results.

1. Create a New Project

1. Open Android Studio.
2. Select **File > New Project**.
3. Choose **Empty Activity** and configure the project with:
    - o Name: SimpleCalculator
    - o Language: Java or Kotlin (this example uses Java).
    - o Minimum SDK: API 21 (Android 5.0 Lollipop).

2. Design the User Interface

In res/layout/activity_main.xml, design a layout for the calculator.

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  android:padding="16dp">

  <!-- Input for Number 1 -->
  <EditText
    android:id="@+id/etNumber1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Enter first number"
    android:inputType="number" />

  <!-- Input for Number 2 -->
  <EditText
```

```
        android:id="@+id/etNumber2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter second number"
        android:inputType="number" />

    <!-- Button for Addition -->
    <Button
        android:id="@+id/btnAdd"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Add" />

    <!-- Button for Subtraction -->
    <Button
        android:id="@+id/btnSubtract"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Subtract" />

    <!-- TextView to Display Result -->
    <TextView
        android:id="@+id/tvResult"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Result: "
        android:paddingTop="16dp"
        android:textSize="18sp" />

</LinearLayout>
```

3. Write the Main Activity Code

In MainActivity.java, handle user input and perform the calculations.

```
package com.example.simplecalculator;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
```

```
EditText etNumber1, etNumber2;
Button btnAdd, btnSubtract;
TextView tvResult;

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

    // Initialize UI components
    etNumber1 = findViewById(R.id.etNumber1);
    etNumber2 = findViewById(R.id.etNumber2);
    btnAdd = findViewById(R.id.btnAdd);
    btnSubtract = findViewById(R.id.btnSubtract);
    tvResult = findViewById(R.id.tvResult);

    // Set up click listeners
    btnAdd.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            performOperation("+");
        }
    });

    btnSubtract.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            performOperation("-");
        }
    });
}

private void performOperation(String operation) {
    String num1Str = etNumber1.getText().toString();
    String num2Str = etNumber2.getText().toString();

    if (!num1Str.isEmpty() && !num2Str.isEmpty()) {
        double num1 = Double.parseDouble(num1Str);
        double num2 = Double.parseDouble(num2Str);
        double result = 0;

        if (operation.equals("+")) {
            result = num1 + num2;
        } else if (operation.equals("-")) {
```

	`result = num1 - num2;`
	`}`
	
	`tvResult.setText("Result: " + result);`
	`} else {`
	`tvResult.setText("Please enter valid numbers");`
	`}`
	`}`
	`}`
	4. Run the Application
	
	1. Connect an Android device or start an emulator.
	2. Click the **Run** button in Android Studio.
	3. Enter two numbers and click **Add** or **Subtract** to see the results.			

Q5(b)	Illustrate the different building blocks of android application design. **1. Activities** ● **Definition**: An Activity represents a single screen with a user interface, such as a login screen, home screen, or settings page. ● **Purpose**: To provide an entry point for user interaction. ● **Example**: A calculator app's main activity might display buttons for numbers and operations. ● **Key Methods**: o onCreate(): Initializes the activity. o onStart(), onResume(), onPause(), onDestroy(): Manage the activity lifecycle. **2. Fragments** ● **Definition**: A Fragment is a modular section of an Activity, representing a portion of the user interface. ● **Purpose**: To enable reusability of UI components across activities. ● **Example**: A weather app might use one fragment for current weather and another for forecasts. ● **Advantages**: o Allows dynamic and flexible UI designs. o Supports device compatibility for tablets and smartphones. **3. Services** ● **Definition**: A Service is a background component that performs long-running operations without a user interface. ● **Purpose**: To handle tasks like downloading files, playing music, or syncing data.	10	L2	CO3

	● **Example**: A music player app might use a service to continue playing music even when the app is minimized. ● **Types**: o **Foreground Services**: Used for tasks noticeable to the user, like media playback. o **Background Services**: Handle tasks that do not need direct user interaction. o **Bound Services**: Allow other components to bind and interact with them. **4. Broadcast Receivers** ● **Definition**: A Broadcast Receiver responds to broadcast messages from the Android system or other applications. ● **Purpose**: To handle system-wide events like network changes, battery levels, or app-specific broadcasts. ● **Example**: A weather app might use a broadcast receiver to update weather data when network connectivity changes. ● **Implementation**: o Register in the AndroidManifest.xml or dynamically in code. **5. Content Providers** ● **Definition**: Content Providers manage shared data between applications. ● **Purpose**: To provide a standard interface for accessing and modifying data, such as contacts or media files. ● **Example**: The Contacts app uses a content provider to allow other apps to access contact details. ● **Methods**: o query(): Retrieve data. o insert(), update(), delete(): Modify data. **6. Intents** ● **Definition**: An Intent is a messaging object used to request an action from another component. ● **Purpose**: To facilitate communication between components (e.g., activities, services). ● **Types**: o **Explicit Intent**: Specifies the target component explicitly. o **Implicit Intent**: Specifies an action, letting the system decide the best component to handle it. ● **Example**: Sharing an image via an email app.			
Q6(a)	Explain the use of different layout type with code snippets wherever possible. **1. LinearLayout**	10	L2	CO3

- **Definition**: Arranges child views in a single row (**horizontal**) or column (**vertical**).
- **Attributes**:
    - orientation: Determines the direction (horizontal/vertical).
    - weight: Allocates extra space to child views.

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:padding="16dp">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Name:" />

   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="Enter your name" />

   <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Submit" />
</LinearLayout>
```

2. RelativeLayout

- **Definition**: Arranges child views relative to each other or to the parent container.
- **Attributes**:
    - layout_alignParentTop, layout_centerInParent, layout_below: Used for positioning views.

```
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:padding="16dp">

   <TextView
      android:id="@+id/tvUsername"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Username" />
```

```
    <EditText
        android:id="@+id/etUsername"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/tvUsername"
        android:layout_marginTop="8dp" />

    <TextView
        android:id="@+id/tvPassword"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/etUsername"
        android:layout_marginTop="16dp"
        android:text="Password" />

    <EditText
        android:id="@+id/etPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/tvPassword"
        android:layout_marginTop="8dp"
        android:inputType="textPassword" />

    <Button
        android:id="@+id/btnLogin"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/etPassword"
        android:layout_marginTop="16dp"
        android:layout_centerHorizontal="true"
        android:text="Login" />
</RelativeLayout>
```

3. FrameLayout

- **Definition**: Used to display a single child view, overlaying multiple views if needed.
- **Use Case**: Ideal for creating overlays or displaying a single item like a video or image.
- <FrameLayout
-     xmlns:android="http://schemas.android.com/apk/res/android"
-     android:layout_width="match_parent"
-     android:layout_height="match_parent">
-
-     <ImageView

- android:layout_width="match_parent"
- android:layout_height="match_parent"
- android:src="@drawable/sample_image"
- android:scaleType="centerCrop" />
-
- <TextView
- android:layout_width="wrap_content"
- android:layout_height="wrap_content"
- android:text="Overlay Text"
- android:textColor="#FFFFFF"
- android:textSize="24sp"
- android:layout_gravity="center" />
- </FrameLayout>

4. TableLayout

- **Definition**: Arranges child views in rows and columns.
- **Use Case**: Suitable for grid-like layouts, such as forms or data tables.

```
<TableLayout

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

    android:layout_width="match_parent"

    android:layout_height="wrap_content">



    <TableRow>

        <TextView

            android:layout_width="wrap_content"

            android:layout_height="wrap_content"

            android:text="Row 1, Column 1" />

        <Button

            android:layout_width="wrap_content"

            android:layout_height="wrap_content"

            android:text="Button 1" />

    </TableRow>
```

```
<TableRow>

    <TextView

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="Row 2, Column 1" />

    <Button

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="Button 2" />

</TableRow>

</TableLayout>
```

5.GridLayout

- **Definition**: Organizes views into a grid-like structure with rows and columns.
- **Attributes**:
  - o rowCount, columnCount: Specify the number of rows and columns.
  - o layout_row, layout_column: Define the position of a view.

Example: Grid of Buttons

```
<GridLayout

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

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    android:rowCount="2"

    android:columnCount="2"

    android:padding="16dp">
    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_columnWeight="1"
        android:text="Button 1" />

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_columnWeight="1"
        android:text="Button 2" />

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_columnWeight="1"
        android:text="Button 3" />

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_columnWeight="1"
        android:text="Button 4" />
</GridLayout>
```

Q6(b)	Describe the process of creating animation with android graphics API.	10	L2	CO3

Android Animation Example

Android provides a large number of classes and interface for the animation development. Most of the classes and interfaces are given in android.animation package.

Android Animation enables you to change the object property and behavior at run time. There are various ways to do animation in android.

The *AnimationDrawable* class provides methods to start and end the animation. Even, you can use time based animation.

Let's have a look at the simple example of android animation.

activity_main.xml

You need to have a view only.

File: activity_main.xml

```
<RelativeLayout 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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <View
        />

</RelativeLayout>
```

File: logo.xml

Have a image view only.

```
<?xml version="1.0" encoding="utf-8"?>
<ImageView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/anm"
     >

</ImageView>
```

MainActivity class

File: MainActivity.java

```
package com.javatpoint.animation;

import android.os.Bundle;
import android.app.Activity;
import android.graphics.drawable.AnimationDrawable;
import android.widget.ImageView;

public class MainActivity extends Activity {

    ImageView anm;
     public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.logo);
          anm = (ImageView)findViewById(R.id.anm);

          anm.setBackgroundResource(R.drawable.animation);
       // the frame-by-frame animation defined as a xml file within the drawable folder

          /*
           * NOTE: It's not possible to start the animation during the onCreate.
           */
       }
     public void onWindowFocusChanged (boolean hasFocus) {
          super.onWindowFocusChanged(hasFocus);
          AnimationDrawable frameAnimation =
             (AnimationDrawable) anm.getBackground();
          if(hasFocus) {
             frameAnimation.start();
          } else {
             frameAnimation.stop();
          }
       }

}
```

You need to create animation.xml file inside res/drawable-hdpi directory.

You need to have many images. Here, we are using 14 images and all the 14 images are located inside res/drawable-mdpi directory.

File: animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">

    <item android:drawable="@drawable/frame0" android:duration="120" />
    <item android:drawable="@drawable/frame1" android:duration="120" />
    <item android:drawable="@drawable/frame2" android:duration="120" />
    <item android:drawable="@drawable/frame3" android:duration="120" />
    <item android:drawable="@drawable/frame4" android:duration="120" />
    <item android:drawable="@drawable/frame5" android:duration="120" />
    <item android:drawable="@drawable/frame6" android:duration="120" />
    <item android:drawable="@drawable/frame7" android:duration="120" />
    <item android:drawable="@drawable/frame8" android:duration="120" />
    <item android:drawable="@drawable/frame9" android:duration="120" />
    <item android:drawable="@drawable/frame10" android:duration="120" />
    <item android:drawable="@drawable/frame11" android:duration="120" />
    <item android:drawable="@drawable/frame12" android:duration="120" />
    <item android:drawable="@drawable/frame13" android:duration="120" />
    <item android:drawable="@drawable/frame14" android:duration="120" />
    <item android:drawable="@drawable/frame14" android:duration="120" />
    <item android:drawable="@drawable/frame13" android:duration="120" />
    <item android:drawable="@drawable/frame12" android:duration="120" />
    <item android:drawable="@drawable/frame11" android:duration="120" />
    <item android:drawable="@drawable/frame10" android:duration="120" />
    <item android:drawable="@drawable/frame9" android:duration="120" />
    <item android:drawable="@drawable/frame8" android:duration="120" />
    <item android:drawable="@drawable/frame7" android:duration="120" />
    <item android:drawable="@drawable/frame6" android:duration="120" />
    <item android:drawable="@drawable/frame5" android:duration="120" />
    <item android:drawable="@drawable/frame4" android:duration="120" />
    <item android:drawable="@drawable/frame3" android:duration="120" />
    <item android:drawable="@drawable/frame2" android:duration="120" />
    <item android:drawable="@drawable/frame1" android:duration="120" />
    <item android:drawable="@drawable/frame0" android:duration="120" />

</animation-list>
```

Q7(a)	With and XML code, demonstrate the creation of user interface making use of commonly used view types.	10	L3	CO4

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:background="#FAFAFA">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="16dp">

        <!-- Header Section -->
        <TextView
            android:id="@+id/tvHeader"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="User Registration"
            android:textSize="24sp"
            android:textStyle="bold"
            android:layout_gravity="center"
            android:paddingBottom="16dp" />

        <!-- ImageView -->
        <ImageView
            android:id="@+id/ivProfile"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:src="@drawable/ic_user_placeholder"
            android:contentDescription="Profile Image"
            android:layout_gravity="center"
            android:layout_marginBottom="16dp"
            android:scaleType="centerCrop"
            android:background="@drawable/circle_shape" />

        <!-- EditText for Name -->
        <EditText
            android:id="@+id/etName"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
    android:hint="Enter your name"
    android:inputType="textPersonName"
    android:padding="8dp"
    android:layout_marginBottom="8dp" />

<!-- EditText for Email -->
<EditText
    android:id="@+id/etEmail"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Enter your email"
    android:inputType="textEmailAddress"
    android:padding="8dp"
    android:layout_marginBottom="8dp" />

<!-- RadioGroup for Gender -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Gender"
    android:layout_marginTop="8dp"
    android:layout_marginBottom="4dp" />

<RadioGroup
    android:id="@+id/rgGender"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <RadioButton
        android:id="@+id/rbMale"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Male" />

    <RadioButton
        android:id="@+id/rbFemale"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Female" />
</RadioGroup>

<!-- CheckBox for Subscription -->
<CheckBox
```

```
        android:id="@+id/cbSubscribe"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Subscribe to Newsletter"
        android:layout_marginTop="8dp" />

    <!-- Button Section -->
    <Button
        android:id="@+id/btnSubmit"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Submit"
        android:backgroundTint="#6200EE"
        android:textColor="#FFFFFF"
        android:layout_marginTop="16dp" />

    <!-- Divider -->
    <View
        android:layout_width="match_parent"
        android:layout_height="1dp"
        android:background="#CCCCCC"
        android:layout_marginTop="16dp" />

    <!-- ListView -->
    <TextView
        android:id="@+id/tvListHeader"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Recent Users"
        android:textStyle="bold"
        android:layout_marginTop="16dp" />

    <ListView
        android:id="@+id/lvUsers"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:divider="#CCCCCC"
        android:dividerHeight="1dp"
        android:layout_marginTop="8dp" />

  </LinearLayout>
</ScrollView>
```

Q7(b)	**Develop an android application media player. Discuss the use of layout types used in application design.**	10	L3	CO4

1. Application Overview

The media player application will include:

- A LinearLayout for a vertical structure.
- An ImageView to display album art.
- A SeekBar to show playback progress.
- Buttons for Play, Pause, and Stop.
- A TextView to display the current media name.

2. Layout Types and Their Use in Media Player

- **LinearLayout**: Used to stack views vertically or horizontally (e.g., arrange media controls in a row).
- **RelativeLayout**: Positioned controls relative to the album art or other components.
- **ConstraintLayout**: For flexible and responsive UI design, keeping elements aligned and adaptable to different screen sizes.
- **FrameLayout**: Used if overlays, such as captions or controls over album art, are needed.

3. Code Implementation

```
<?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"
    android:padding="16dp"
    android:background="#FAFAFA">

    <!-- Media Title -->
    <TextView
        android:id="@+id/tvMediaTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Media Title"
        android:textSize="20sp"
        android:layout_gravity="center_horizontal"
        android:layout_marginBottom="16dp"
        android:textStyle="bold" />

    <!-- Album Art -->
```

```
<ImageView
    android:id="@+id/ivAlbumArt"
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:layout_gravity="center_horizontal"
    android:src="@drawable/ic_album_art_placeholder"
    android:contentDescription="Album Art"
    android:scaleType="centerCrop"
    android:layout_marginBottom="16dp" />

<!-- SeekBar for Progress -->
<SeekBar
    android:id="@+id/seekBar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="16dp" />

<!-- Playback Controls -->
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:gravity="center"
    android:layout_marginTop="16dp">

    <!-- Play Button -->
    <Button
        android:id="@+id/btnPlay"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Play" />

    <!-- Pause Button -->
    <Button
        android:id="@+id/btnPause"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Pause"
        android:layout_marginStart="16dp" />

    <!-- Stop Button -->
    <Button
        android:id="@+id/btnStop"
        android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="Stop"
            android:layout_marginStart="16dp" />
    </LinearLayout>
</LinearLayout>
```

Java Code for Media Player Functionality

```
package com.example.mediaplayer;

import android.media.MediaPlayer;
import android.os.Bundle;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private MediaPlayer mediaPlayer;
    private SeekBar seekBar;
    private Button btnPlay, btnPause, btnStop;
    private TextView tvMediaTitle;
    private boolean isPlaying = false;

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

        // Initialize Views
        tvMediaTitle = findViewById(R.id.tvMediaTitle);
        seekBar = findViewById(R.id.seekBar);
        btnPlay = findViewById(R.id.btnPlay);
        btnPause = findViewById(R.id.btnPause);
        btnStop = findViewById(R.id.btnStop);

        // Media Player setup
        mediaPlayer = MediaPlayer.create(this, R.raw.sample_audio); // Replace with your
audio file
        tvMediaTitle.setText("Sample Audio");

        // Play Button
        btnPlay.setOnClickListener(v -> {
            if (!isPlaying) {
                mediaPlayer.start();
            isPlaying = true;
            updateSeekBar();
        }
    });

    // Pause Button
    btnPause.setOnClickListener(v -> {
        if (isPlaying) {
            mediaPlayer.pause();
            isPlaying = false;
        }
    });

    // Stop Button
    btnStop.setOnClickListener(v -> {
        if (mediaPlayer.isPlaying() || isPlaying) {
            mediaPlayer.stop();
            mediaPlayer = MediaPlayer.create(this, R.raw.sample_audio); // Reset
            isPlaying = false;
            seekBar.setProgress(0);
        }
    });

    // SeekBar Listener
    seekBar.setMax(mediaPlayer.getDuration());
    seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
        @Override
        public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
            if (fromUser) {
                mediaPlayer.seekTo(progress);
            }
        }

        @Override
        public void onStartTrackingTouch(SeekBar seekBar) {}

        @Override
        public void onStopTrackingTouch(SeekBar seekBar) {}
    });

    // Release resources on completion
    mediaPlayer.setOnCompletionListener(mp -> {
        isPlaying = false;
        seekBar.setProgress(0);
        });
    }

    private void updateSeekBar() {
        new Thread(() -> {
            while (mediaPlayer != null && mediaPlayer.isPlaying()) {
                seekBar.setProgress(mediaPlayer.getCurrentPosition());
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }

    @Override
    protected void onDestroy() {
        if (mediaPlayer != null) {
            mediaPlayer.release();
            mediaPlayer = null;
        }
        super.onDestroy();
    }
}
```

Discussion of Layout Types Used

1. **LinearLayout**:
   - o Used for vertical stacking of the UI components, making it simple to organize the UI in a logical order.
   - o Suitable for aligning playback controls horizontally.
2. **RelativeLayout/ConstraintLayout** (not used here but useful):
   - o Could be used for positioning components relative to each other, like aligning buttons below the SeekBar.
3. **ScrollView**:
   - o Useful if the UI expands beyond the screen height.
4. **FrameLayout**:
   - o Could be used for overlaying controls on the ImageView for advanced designs.

Q8(a)	**Discuss the use of Dialogs displaying information in android application.**	10	L2	CO4

Types of Dialogs in Android

1. **AlertDialog**

   - The most commonly used dialog to display simple information or get feedback from the user (e.g., Yes/No choices, warnings, etc.).
   - Allows customization with buttons, lists, and icons.
2. **ProgressDialog** (Deprecated in API 26)
   - Previously used for showing loading or progress status. It's now recommended to use ProgressBar within a layout or a DialogFragment for a more flexible solution.
3. **Custom Dialogs**
   - A completely customizable dialog where you can use your own layout to display any UI components.
4. **DatePickerDialog / TimePickerDialog**
   - Specialized dialogs for selecting a date or time.
5. **MultiChoice Dialog**
   - Allows users to select multiple items from a list.

1. AlertDialog

AlertDialog is the most versatile and widely used dialog type in Android. It allows you to display messages, options, or custom views.

Explanation:

- **setTitle()**: Sets the title of the dialog.
- **setMessage()**: Sets the message content.
- **setPositiveButton()**: Adds a button that performs an action when clicked (e.g., "Yes").
- **setNegativeButton()**: Adds another button with an alternate action (e.g., "No").
- **setCancelable()**: Specifies whether the dialog can be canceled by pressing the back button or tapping outside the dialog.

2. ProgressDialog (Deprecated)

While ProgressDialog was used to show a loading progress, it is now deprecated as of **API 26**. The recommended approach is to use ProgressBar in a DialogFragment or directly in the layout.

3. Custom Dialog

Sometimes, the predefined dialog types are not sufficient, and you need a completely custom UI. In such cases, you can create your own dialog using a custom layout.
Creating a Custom Dialog

1. **Create a Layout for the Custom Dialog** (res/layout/custom_dialog.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
```

	`    android:orientation="vertical"`
	`    android:padding="20dp">`
	
	`    <TextView`
	`        android:id="@+id/tvMessage"`
	`        android:layout_width="match_parent"`
	`        android:layout_height="wrap_content"`
	`        android:text="This is a custom dialog"`
	`        android:textSize="16sp" />`
	
	`    <Button`
	`        android:id="@+id/btnClose"`
	`        android:layout_width="wrap_content"`
	`        android:layout_height="wrap_content"`
	`        android:text="Close" />`
	`</LinearLayout>`
	
	4. DatePickerDialog / TimePickerDialog
	
	Android provides specialized dialogs to select a date or time.
	DatePickerDialog
	`DatePickerDialog datePickerDialog = new DatePickerDialog(this,`
	`    (view, year, monthOfYear, dayOfMonth) -> {`
	`        // Handle date selection`
	`        String selectedDate = dayOfMonth + "/" + (monthOfYear + 1) + "/" + year;`
	`        Log.d("Selected Date", selectedDate);`
	`    }, 2024, 11, 25);  // Initial date`
	`datePickerDialog.show();`
	TimePickerDialog
	`TimePickerDialog timePickerDialog = new TimePickerDialog(this,`
	`    (view, hourOfDay, minute) -> {`
	`        // Handle time selection`
	`        String selectedTime = hourOfDay + ":" + minute;`
	`        Log.d("Selected Time", selectedTime);`
	`    }, 14, 30, true);  // Initial time`
	`timePickerDialog.show();`			

Q8(b)	**Develop an android application to access internet and display web pages in android application.** **Steps to Develop the Application:** 1. **Set Up the Android Project**: o Create a new Android project in Android Studio. 2. **Add Internet Permissions**:	10	L3	CO4

- o Open the AndroidManifest.xml file and add the necessary permission to access the internet.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Create the Layout with WebView:

- In your activity_main.xml, define the WebView component to display web pages.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- WebView to display web pages -->
    <WebView
        android:id="@+id/webview"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

Set Up WebView in MainActivity:

- In your MainActivity.java, initialize the WebView and load a URL.

```
package com.example.webviewexample;

import android.os.Bundle;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private WebView webView;

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

        // Initialize the WebView
        webView = findViewById(R.id.webview);

        // Enable JavaScript (optional, depending on the webpage you are loading)
        WebSettings webSettings = webView.getSettings();
        webSettings.setJavaScriptEnabled(true);

        // Set WebViewClient to open links within the WebView itself
        webView.setWebViewClient(new WebViewClient());

        // Load a URL (web page)
        webView.loadUrl("https://www.example.com");
    }

    @Override
    public void onBackPressed() {
        // Handle back press to navigate within the WebView history
        if (webView.canGoBack()) {
            webView.goBack();  // Go to the previous page
        } else {
            super.onBackPressed();  // Exit the app if no history is available
        }
    }
}
```

Manifest File (AndroidManifest.xml):

Make sure the application has the required internet permission and the main activity is properly set up in the manifest.

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

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/Theme.AppCompat.Light.DarkActionBar">
        <activity android:name=".MainActivity"
            android:label="WebView Example"
            android:theme="@style/Theme.AppCompat.Light.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Q9(a)	**Demonstrate the use of maps in Android application and develop an application to mark the pointer to Sydney, Australia using maps.**	10	L2	CO5

1. Set Up Google Maps SDK for Android

Before you can use Google Maps in your Android application, you need to set up your project to use the **Google Maps SDK**.
Step 1: Set Up Google Cloud Project and API Key

1. **Create a Google Cloud Project**:
    - o Go to the Google Cloud Console.
    - o Create a new project if you don't have one.
2. **Enable Google Maps API**:
    - o In the Google Cloud Console, search for **Google Maps SDK for Android**.
    - o Enable the API for your project.
3. **Get API Key**:
    - o Once the API is enabled, go to **Credentials** and create an **API key**.
    - o Restrict the API key to avoid misuse (e.g., restrict by package name).

Step 2: Add Dependencies to build.gradle

Add the necessary dependencies for Google Maps in your app/build.gradle file.

```
dependencies {
   implementation 'com.google.android.gms:play-services-maps:17.0.1'
   implementation 'com.google.android.gms:play-services-location:17.0.0'
}
```

Step 3: Add Permissions in AndroidManifest.xml

In your AndroidManifest.xml, you need to add the following permissions for accessing the internet and location services:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

<application
   android:allowBackup="true"
   android:icon="@mipmap/ic_launcher"
   android:label="Map Example"
   android:theme="@style/Theme.AppCompat.Light.NoActionBar">

   <meta-data
      android:name="com.google.android.maps.v2.API_KEY"
      android:value="@string/maps_api_key"/>

   <activity android:name=".MainActivity"
      android:label="Map Example"
      android:theme="@style/Theme.AppCompat.Light.NoActionBar">
      <intent-filter>
```

```
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```
Add the API key in the strings.xml file.
```
<resources>
    <string name="maps_api_key">YOUR_GOOGLE_MAPS_API_KEY</string>
</resources>
```
Step 4: Create the Layout with SupportMapFragment

In the activity_main.xml, create a FrameLayout or any layout that holds the MapFragment to display the map.
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- MapFragment to display the map -->
    <fragment
        android:id="@+id/map"
        android:name="com.google.android.gms.maps.SupportMapFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</RelativeLayout>
```

2. MainActivity Setup

Now, set up the MainActivity.java to initialize the map and mark Sydney, Australia.
Step 1: Initialize the Map and Add a Marker
```
package com.example.mapsdemo;

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

public class MainActivity extends FragmentActivity implements OnMapReadyCallback {

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

        // Initialize the map fragment
                              SupportMapFragment    mapFragment    =    (SupportMapFragment)
getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);  // Callback when map is ready
    }

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

        // Define the location of Sydney, Australia
        LatLng sydney = new LatLng(-33.8688, 151.2093);

        // Add a marker at Sydney and move the camera to it
        mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
            mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(sydney, 10));   // Zoom
level of 10
    }
}
```

Step 2: Handle Permissions for Location (Optional)

If you want to allow the user to see their location on the map, you need to request permissions to access the user's location. You can use the **FusedLocationProviderClient** for location updates.

In MainActivity, request the permission to access location:

```
import android.content.pm.PackageManager;
import android.location.Location;
import android.os.Build;
import android.widget.Toast;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

private static final int LOCATION_PERMISSION_REQUEST_CODE = 1;

@Override
protected void onResume() {
    super.onResume();
```

```
                                if      (ContextCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_FINE_LOCATION)
        != PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this,
        new String[]{android.Manifest.permission.ACCESS_FINE_LOCATION},
        LOCATION_PERMISSION_REQUEST_CODE);
  }
}
```

3. Running the Application

Now that you have set up everything, you can run the application on an emulator or a real device. The map will load and display a marker at **Sydney, Australia**.

Expected Result

When the application starts:

1. The map will be displayed.
2. A marker will be placed at Sydney, Australia (coordinates: **Latitude: -33.8688**, **Longitude: 151.2093**).
3. The camera will zoom in on Sydney, displaying the marker.

Q9(b)	**Develop an android application to send an email text message.**	10	L3	CO5

1. Set Up the Layout (activity_main.xml)

First, create a simple layout with an EditText for entering the email text message and a Button to trigger the sending of the email.

```
<?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"
  android:padding="16dp">

  <!-- EditText for Email Subject -->
  <EditText
    android:id="@+id/email_subject"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Enter Subject"
    android:padding="8dp" />

  <!-- EditText for Email Body -->
  <EditText
    android:id="@+id/email_body"
    android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:hint="Enter your message"
        android:padding="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginBottom="16dp"
        android:inputType="textMultiLine"
        android:gravity="top" />

    <!-- Send Email Button -->
    <Button
        android:id="@+id/send_email_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Send Email"
        android:layout_gravity="center" />
</LinearLayout>
```

This layout includes:

- An EditText for the subject of the email.
- An EditText for the body of the email.
- A Button to trigger the email sending action.

2. Add Internet Permissions in AndroidManifest.xml

Although you're using an external email client to send the email (not directly sending the email over a network), it is good practice to ensure your app has the necessary permissions. For sending emails via other apps, no special permissions are required, but you might still want to add these for future use.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

3. Write the Logic in MainActivity.java

In your MainActivity.java, you'll need to:

- Get the subject and body from the user inputs.
- Create an Intent to send the email using the default email client.

```
package com.example.sendemailapp;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private EditText emailSubjectEditText;
    private EditText emailBodyEditText;
    private Button sendEmailButton;

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

        // Initialize UI components
        emailSubjectEditText = findViewById(R.id.email_subject);
        emailBodyEditText = findViewById(R.id.email_body);
        sendEmailButton = findViewById(R.id.send_email_button);

        // Set an onClickListener for the Send Email button
        sendEmailButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Get subject and body from user input
                String subject = emailSubjectEditText.getText().toString();
                String body = emailBodyEditText.getText().toString();

                // Create an email Intent
                Intent emailIntent = new Intent(Intent.ACTION_SEND);
                emailIntent.setType("message/rfc822"); // Set type as email
                                                  emailIntent.putExtra(Intent.EXTRA_EMAIL,    new
String[]{"recipient@example.com"}); // Add recipient email address (optional)
                emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject); // Add email subject
                emailIntent.putExtra(Intent.EXTRA_TEXT, body); // Add email body

                try {
                    // Start an email client (app chooser)
                    startActivity(Intent.createChooser(emailIntent, "Choose an Email Client"));
                } catch (android.content.ActivityNotFoundException ex) {
                    // Handle case where no email client is found
                    // You can show a Toast or an alert to the user
                }
            }
        });
    }
}
```

Q10(a)	**Explain context providers. How they are useful in application development in Android.** **What is a Content Provider?** A **Content Provider** in Android is a component used to access and manage structured data in a standardized way. A content provider is often used to share data between different apps, but it can also be used for accessing data within the same app. It acts as a middleman that allows apps to interact with data sources, such as a database or files, through a defined interface.	10	L2	CO5

The main functionality of a content provider is to offer a **uniform interface** for accessing data from different sources. Content providers can be used for many purposes:

- Sharing data between apps (e.g., contact information or images).
- Storing data locally in an SQLite database.
- Providing access to content from cloud storage.

How Content Providers Work:

Content providers expose a data model via a **URI** (Uniform Resource Identifier), which identifies the data and the operations that can be performed on it. An app can then query the content provider, perform inserts, or retrieve information from it using a standardized API.

A content provider is built using the ContentProvider class, and it implements methods to handle the various operations like:

- query(): To retrieve data.
- insert(): To insert new data.
- update(): To update existing data.
- delete(): To delete data.
- getType(): To return the MIME type of the data.

How Are Content Providers Useful in Application Development?

1. **Data Sharing**:
   - Content providers make it possible to share data between different applications. For example, a contact list provider (ContactsContract in Android) allows third-party apps to access contact information without needing direct access to the database.
   - Example: A photo gallery app can use a content provider to allow other apps to access and manipulate photos in the gallery.
2. **Abstracting Data Access**:
   - A content provider abstracts the underlying data structure from the client (app). This allows for easier management of the data and ensures that apps do not need to know the details of where or how the data is stored.
   - Example: A contact provider abstracts the storage of contact data in a local database, ensuring that apps can access contacts without having to manage databases themselves.
3. **Data Security**:
   - Content providers provide controlled access to data. By defining permissions in the manifest file, the app can limit which apps can access the data. The access control is enforced by the content provider, which ensures that only authorized apps can read or modify the data.
   - Example: When an app accesses shared contact information, it may be required to request specific permissions like READ_CONTACTS, ensuring that sensitive information is only shared with authorized apps.
4. **Consistency**:
   - A content provider provides a consistent and standardized way to interact with data, ensuring that all apps follow the same pattern. Whether you're

	accessing shared data or data specific to an app, content providers allow you to work with data in a consistent way, regardless of its source. o Example: Whether the data is stored in a file, an SQLite database, or a cloud service, the content provider provides a consistent interface for querying, inserting, and updating the data. 5. **Support for Multiple Data Sources**: o A content provider can abstract access to data from multiple data sources (e.g., local database, remote server, shared preferences, or external storage). This allows developers to work with diverse data sources while maintaining a uniform interface. o Example: An app might use a content provider to access data stored locally in an SQLite database and remotely in a cloud-based service. 6. **Inter-App Communication**: o Content providers facilitate communication between different apps. For instance, an app can query the contacts provider to fetch a user's contacts and display them in its own interface. o Example: A messaging app might use the contacts content provider to display contact names and numbers for sending text messages or emails. 7. **Local Data Management**: o Content providers are commonly used for managing local app data. For example, an app can store its data in an SQLite database and use a content provider to provide access to that data for querying, updating, or deletion. o Example: A task management app might store tasks in an SQLite database, and the content provider can expose a standardized API to interact with that data.			
Q10(b)	**Explain the steps required to create an APK file and publish the application into play stack.** Step 1: Prepare Your App for Release Before you can create an APK for release, make sure your app is ready for production. This includes: 1. **Clean up your app:** o Remove unnecessary debug code, logs, and unused resources. o Ensure that your app follows best practices for performance, security, and user experience. 2. **Test the app thoroughly:** o Test your app on different devices and screen sizes. o Check for any crashes or performance issues. o Test your app with different network conditions and permissions. 3. **Check your app's configuration:** o Set the correct version number in build.gradle. o Ensure that your app has the correct permissions listed in the AndroidManifest.xml. o Review the app's settings for any necessary optimizations, such as adjusting for different screen densities, API levels, etc.	10	L2	CO5

Step 2: Generate a Signed APK

A signed APK is required to publish your app on the Google Play Store. This ensures the authenticity of the app and enables updates.

1. Generate a Keystore File

A keystore file is used to sign your APK. If you don't have one already, you'll need to create it. Follow these steps:

- Open a terminal (or Command Prompt on Windows).
- Run the following command to create the keystore file (replace my-release-key.jks with your desired filename and adjust other values accordingly):

bash

Copy code

```
keytool -genkeypair -v -keystore my-release-key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias my-key-alias
```

- You will be asked to enter some details (like your name, organization, and password). Make sure to remember the password as you will need it later.

2. Configure Your App for Release

- Open your project in Android Studio.
- In the **Build** menu, select **Generate Signed Bundle / APK**.
- Choose **APK** and click **Next**.
- Select the **Release** build variant.
- Click **Create new** (if you don't have a keystore file) and provide the path to your keystore, the alias, and the passwords you set during the keystore creation.

3. Build the APK

- After entering the required information (Keystore file, Key alias, Key password), click **Next**.
- Choose **Release** as the build variant and specify whether you want a **debuggable** or **non-debuggable** APK. For release, ensure that it is not debuggable.
- Click **Finish**, and Android Studio will build your signed APK. Once the build is complete, you can find the APK in the output folder (usually in app/build/outputs/apk/release/).

Step 3: Optimize Your APK (Optional but Recommended)

You can optimize the APK file size using **ProGuard** (now known as **R8**) or **Android App Bundles**.

1. **ProGuard / R8**:
   - o ProGuard is a tool that shrinks, optimizes, and obfuscates your code.
   - o To enable ProGuard, configure it in your build.gradle file:

   gradle

   Copy code

```
buildTypes {
   release {
      minifyEnabled true
      shrinkResources true
               proguardFiles  getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
   }
}
```

2. **App Bundles**:
   - o Google Play now encourages developers to publish **Android App Bundles** (.aab) instead of APKs for better app delivery optimization. An App Bundle allows Google Play to generate APKs for different device configurations, reducing the download size for users.
   - o To build an App Bundle, select **Build > Build Bundle** in Android Studio.
   - o This will generate an .aab file instead of an APK, which you can upload to the Play Store.

Step 4: Create a Developer Account on Google Play Console

To publish your app, you need a **Google Play Developer account**. Here’s how to create one:

1. Go to the Google Play Console.
2. Sign in with your Google account or create one.
3. Pay a one-time fee of $25 to register as a developer.
4. Complete the necessary forms and provide your business information.

Step 5: Prepare Store Listing

Before uploading your APK, you need to prepare a store listing for your app. This will include details about your app such as:

1. **App Name**: The name of your app.
2. **App Description**: A detailed description of your app's features and functionality.
3. **Screenshots**: High-quality screenshots of your app (at least 2–4 screenshots from different screens of your app).
4. **App Icon**: A high-resolution app icon (512x512 px).
5. **App Category**: Select the appropriate category for your app (e.g., Games, Productivity, etc.).
6. **Contact Details**: Provide contact email, website (if any), and physical address (optional).
7. **Privacy Policy**: A link to your app's privacy policy (required if your app handles sensitive user data).

Step 6: Upload APK to Google Play Console

Once you have completed the store listing and prepared your APK or App Bundle, follow these steps to upload your app to the Google Play Console:

1. Log in to the **Google Play Console**.
2. Select **Create Application** and choose the default language.
3. Enter the app's title and description.
4. In the **App releases** section, select **Production** or a different track (Alpha or Beta if you're testing).
5. Upload the signed APK or App Bundle (.apk or .aab file).
6. Complete the content rating questionnaire, privacy policy, and any necessary compliance information (e.g., for children's apps).
7. Click **Save** and then **Review and Rollout** when you are ready to publish.

Step 7: Publish Your App

Once you've uploaded your APK/App Bundle and filled out all the required fields (store listing, content rating, etc.), you can submit your app for review. The steps are:

1. **Review**: Double-check everything, especially the content rating and permissions requested by the app.
2. **Submit for Review**: Once all sections are complete and you're satisfied with your submission, click **Submit** for review.
3. **Google Review**: Google will review your app. This process typically takes a few hours to a couple of days. If there are no issues, your app will be published.

Step 8: Monitor and Update Your App

Once your app is live, you can monitor its performance and user feedback through the **Google Play Console**.

- **User Feedback**: Monitor app ratings, reviews, and user comments. Respond to users and address any issues raised.
- **Analytics**: Use Google Play Console to track the performance of your app (downloads, installs, crashes, etc.).
- **Update Your App**: When you need to make updates, follow the same steps for creating a new signed APK or App Bundle, upload it to the Play Console, and publish the update.