CBCS SCHEME

USN | 1 | C | R | 2 | 2 | M | C | 0 | H | 9 | 22MCA263

Second Semester MCA Degree Examination, June/July 2023
Mobile Application Development

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

			M	L	C
		Module – 1			
Q.1	a.	With a neat diagram explain GSM architecture.	10	L1	CO1
	b.	Explain system service architecture of mobile application.	10	L2	CO1
		OR			
Q.2	a.	What is protocol? Explain protocol architecture in mobile computing with a neat diagram.	10	L2	CO1
	b.	What is handover? Explain its different types.	10	L1	CO1
		Module – 2			
Q.3	a.	What is android? Explain its architecture with a neat diagram.	10	L1	CO2
	b.	Explain steps involved in installing android.	5	L1	CO2
	c.	Explain steps involved in creating android virtual device.	5	L1	CO2
		OR			
Q.4	a.	How you will create and execute first android project? Explain with program steps.	10	L2	CO2
	b.	Write a android program to print "Hello welcome" on emulator with a image in the background.	10	L3	CO2
		Module – 3			
Q.5	a.	What is activity? Explain the activity Life cycle in android with a neat diagram.	10	L2	CO3
	b.	Explain content providers in android.	10	L2	CO3
		OR			
Q.6	a.	Explain table layout and its features.	5	L1	CO3
	b.	Write a program to illustrate different drawing graphics in android.	10	L2	CO3
	c.	How animation is created in Android? Explain.	5	L1	CO3
		Module – 4			
Q.7	a.	With an example for each, explain common views in Android.	10	L1	CO4
	b.	Explain different layouts in android.	10	L1	CO4
		OR			
Q.8	a.	How multimedia can be implemented in Android? Explain.	10	L2	CO4
	b.	Explain how to check internet connection in Android programmatically.	10	L3	CO4
		Module – 5			
Q.9	a.	Write a program to demonstrate GPS connection in Android.	10	L3	CO
	b.	Explain the procedure of sending email in Android project.	10	L3	CO
		OR			
Q.10	a.	Explain services in android.	10	L3	CO
	b.	Explain procedure involved in publishing android application in Google play store.	10	L3	CO

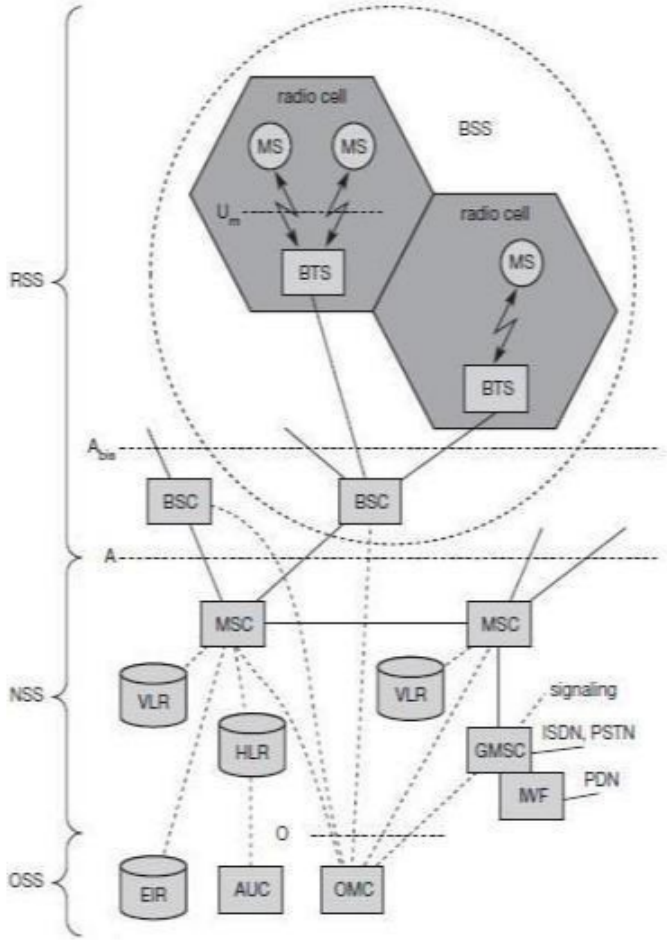
* * * * *

1.a. With a neat diagram explain GSM architecture.

A GSM system consists of three subsystems, the radio sub system (RSS), the network and switching subsystem (NSS), and the operation subsystem (OSS).

Network Switching Subsystem: The NSS is responsible for performing call processing and subscriber related functions. The switching system includes the following functional units:

- Home location register (HLR): The HLR has a database that used for storage and management of subscriptions. HLR stores all the relevant subscriber data including a subscribers service profile such as call forwarding, roaming, location information and activity status.
- Visitor location register (VLR): It is a dynamic real-time database that stores both permanent and temporary subscribers data which is required for communication b/ w the coverage area of MSC and VLR.
- Authentication center (AUC): A unit called the AUC provides authentication and encryption parameters that verify the users identity and ensure the confidentiality of each call.
- Equipment identity register (EIR): It is a database that contains information about the identity of mobile equipment that prevents calls from stolen, unauthorized or defective mobile stations.
- Mobile switching center (MSC): The MSC performs the telephony switching functions of the system. It has various other functions such as 1. Processing of signal. 2. Control calls to and from other telephone and data systems. 3. Call changing, multi-way calling, call forwarding, and other supplementary services. 4. Establishing and terminating the connection b/w MS and a fixed line phone via GMSC.



Radio Subsystem (RSS): The **radio subsystem (RSS)** comprises all radio specific entities, i.e., the **mobile stations (MS)** and the **base station subsystem (BSS)**. The figure shows the connection between the RSS and the NSS via the A **interface** (solid lines) and the connection to the OSS via the **O interface** (dashed lines).

- Base station subsystem (BSS): A GSM network comprises many BSSs, each controlled by a base station controller (BSC). The BSS performs all functions necessary to maintain radio connections to an MS, coding/decoding of voice, and rate adaptation to/from the wireless network part. Besides a BSC, the BSS contains several BTSs.
- Base station controllers (BSC): The BSC provides all the control functions and physical links between the MSC and BTS. It is a high capacity switch that provides functions such as handover, cell configuration data, and control of radio frequency (RF) power levels in BT". A number of BSC's are served by and MSC.
- Basetransceiver station(BTS):The BTS handles the radio interface to the mobile station. A BTS can form a radio cell or, using sectorized antennas, several and is connected to MS via the Um interface, and to the BSC via the A BTS interface. The Um interface contains all the mechanisms necessary for wireless transmission (TDMA, FDMA etc.). The BTS is the radio equipment (transceivers and antennas) needed to service each cell in the network. A group of BTS's are controlled by an BSC.

Operation Service Subsystem (OSS): The OSS facilitates the operations of MSCs. The OSS also handles the Operation and maintenance (OMC) of the entire network.

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. The purpose of OSS is to offer the customer cost-effective support for centralized, regional and local operational and maintenance activities that are required for a GSM network. OSS provides a network overview and allows engineers to monitor, diagnose and troubleshoot every aspect of the GSM network.

The **mobile station (MS)** consists of the mobile equipment (the terminal) and a smart card called the Subscriber Identity Module (SIM). The SIM provides personal mobility, so that the user can have access to subscribed services irrespective of a specific terminal. By inserting the SIM card into another GSM terminal, the user is able to receive calls at that terminal, make calls from that terminal, and receive other subscribed services.

The mobile equipment is uniquely identified by the International Mobile Equipment Identity (IMEI). The SIM card contains the International Mobile Subscriber Identity (IMSI) used to identify the subscriber to the system, a secret key for authentication, and other information. The IMEI and the IMSI are independent, thereby allowing personal mobility. The SIM card may be protected against unauthorized use by a password or personal identity number.

1.b. Explain system service architecture of mobile application.

It represents the architectural requirements for programming a mobile device. The requirements are Programming Languages, Functions of OS, and Functions of middleware for mobile systems.

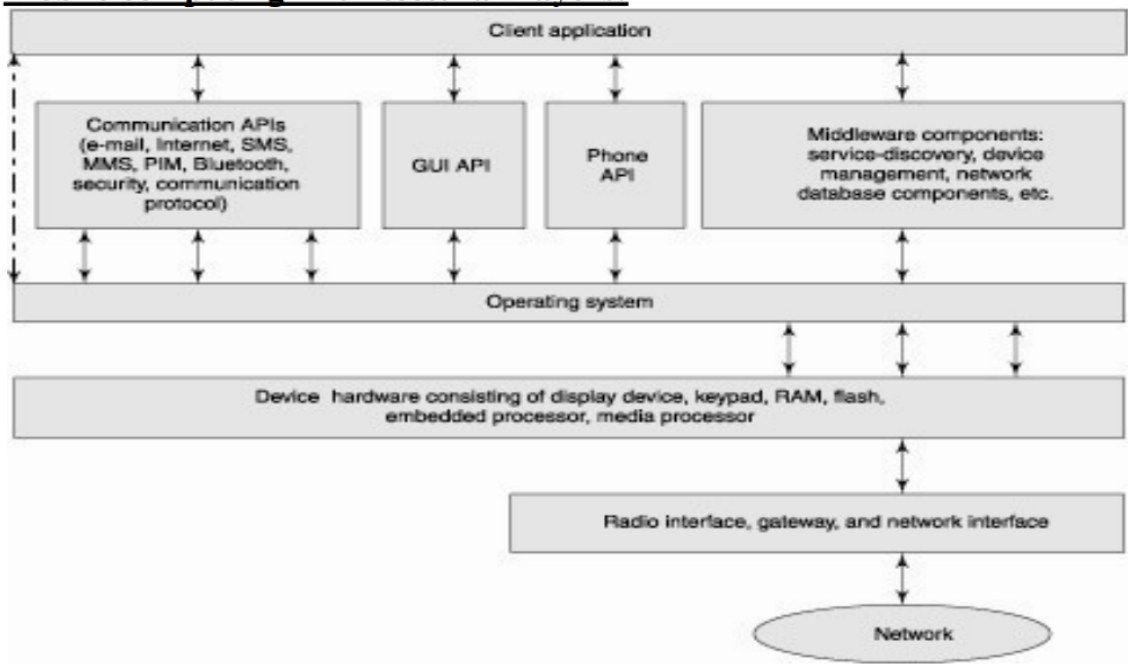
Mobile computing Architectural Layers, protocols and Layers

i) Programming Languages: A variety of programming languages are used in mobile computing architecture. Popular language used is Java J2ME and Javacard (Java for smartcard) J2EE is used for web and enterprise server-based applications of mobile services. DOTNET and Python 2.7 are also used.

ii) Functions of OS: An OS enables the user to run an application without considering the hardware specifications and functionalities. Scheduling multiple tasks. Management functions for tasks and memory. Interfaces for communication. Configurable libraries for the GUI in the device. Middleware.

iii) Functions of middleware for mobile systems: Middleware are the software components that link the application components with the network-distributed components. Mobile OS also provides middleware components. Examples are -to discover the nearby Bluetooth device, discover the nearby hotspot. -to retrieve data from a network database. -for service discovery

Mobile computing Architectural Layers:



It refers to defining various layers between the user applications, interfaces, devices and network hardware.

A well-defined architecture is necessary for systematic computations and access to data

and software objects in the layers.

Protocols:

GSM900, GSM900/1800/1900, CDMA, WCDMA, HSPA, UMTS, i-Mode, LTE, and WiMax.
WPAN protocols such as Bluetooth, IrDA, and Zigbee.
WLAN protocols such as 802.11a, and 802.11b and WAP
Layers: The OSI (open standard for interchange) seven-layer format is
-Physical for sending and receiving signals (TDMA or CDMA coding)
-Data link (Multiplexing), Networking (for linking to the destination)
-Wireless transport layer security (for establishing end-to-end connectivity)
-Wireless transaction protocol, Wireless session protocol, and Wireless application environment (for running a mob.appln., e.g., mobile e-business)

2.a. What is protocol? Explain protocol architecture in mobile computing with a neat diagram.

GSM Protocols: The signaling protocol in GSM is structured into three general layers depending on the interface, as shown below.

Layer 1 is the physical layer that handles all radio-specific functions. This includes the creation of bursts according to the five different formats, multiplexing of bursts into a TDMA frame, synchronization with the BTS, detection of idle channels, and measurement of the channel quality on the downlink.

The main tasks of the physical layer contain channel coding and error detection/ correction, which are directly combined with the coding mechanisms. Channel coding using different forward error correction (FEC) schemes.

Signaling between entities in a GSM network requires higher layers. For this purpose, the **LAPDm**

protocol has been defined at the Um interface for **layer two**. LAPDm has been derived from link access procedure for the D-channel (**LAPD**) in ISDN systems, which is a version of HDLC.

The **network layer** in GSM contains several sub-layers. The lowest sub-layer is the radio resource management (RR). The functions of RR' are supported by the BSC via the BTS management (BTSM). The maintasksofRRare setup, maintenance, and release of radio channels. Mobility management (MM) contains functions for registration, authentication, identification, location updating, and the provision of a temporary mobile subscriber identity (TMSI).

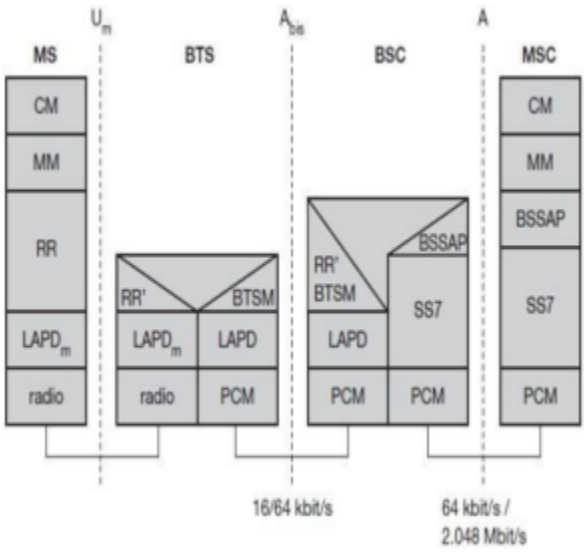
Finally, the call management (CM) layer contains three entities: call **control (CC), short message service (SMS)**, and **supplementary service (SS).**

SMS allows for message transfer using the control channels SDCCH and SACCH, while SS offers the services like user identification, call redirection, or forwarding of ongoing calls.

CC provides a point-to-point connection between two terminals and is used by higher layers for call establishment, call clearing and change of call parameters.

Data transmission at the physical layer typically uses pulse code modulation (PCM) systems. LAPD is used for layer two at Abis, BTSM for BTS management.

Signaling system No. 7 (SS7) is used for signaling between an MSC and a BSC. This protocol also transfers all management information between MSCs, HLR, VLRs, AuC, EIR, and OMC. An MSC can also control a BSS via a BSS application part (**BSSAP**).

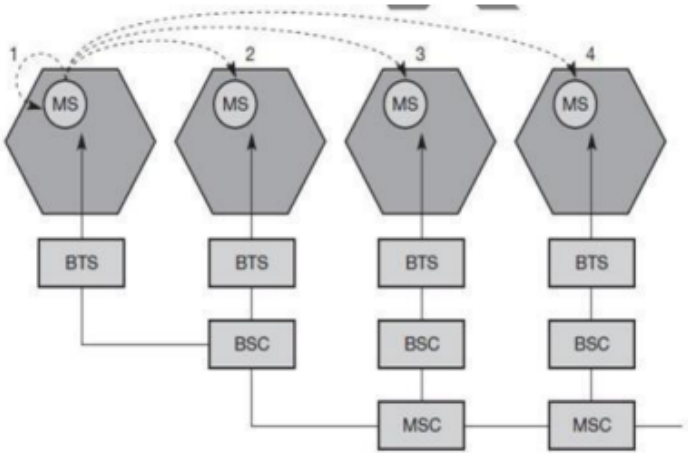


2.b. What is handover? Explain its different types.

Handover: Cellular systems require handover procedures, as single cells do not cover the whole service area. However, a handover should not cause a cut-off, also called call drop.

GSM aims at maximum handover duration of 60 ms. There are two basic reasons for a handover:

1. The mobile station moves out of the range of a BTS, decreasing the received signal level increasing the error rate thereby diminishing the quality of the radio link.
2. Handover may be due to load balancing, when an MSC/BSC decides the traffic is too high in one cell
   and shifts some MS to other cells with a lower load.

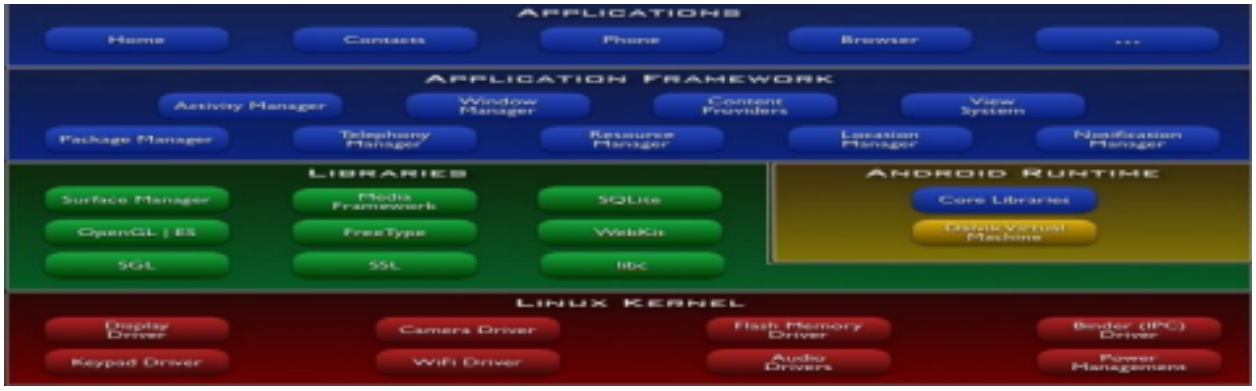


The four possible handover scenarios of GSM are shown below:

- **Intra-cell handover:** Withinacell, narrow-band interference could maketransmission at a certain frequency impossible. The BSC could then decide to change the carrier frequency (scenario
- **Inter-cell, intra-BSC handover:** This is a typical handover scenario. The mobile station moves from one cell to another, but stays within the control of the same BSC. The BSC then performs a handover, assignsanewradiochannelinthenewcellandreleasesthe old one.
- **Inter-BSC, intra-MSC handover:** As a BSC only controls a limited number of cells; GSM also has to perform handovers between cells controlled by different BSCs. This handover then has to be controlled by the MSC.
- **Inter MSC handover:** A handover could be required between two cells belonging to different MSCs. Now both MSCs perform the handover together

3.a. What is an android? Explain its architecture with a neat diagram.

The Android software stack consists of a Linux kernel and a collection of C/C++ libraries that are exposed through an application framework for application development.
The Android software stack consists of four main layers as shown in diagram.

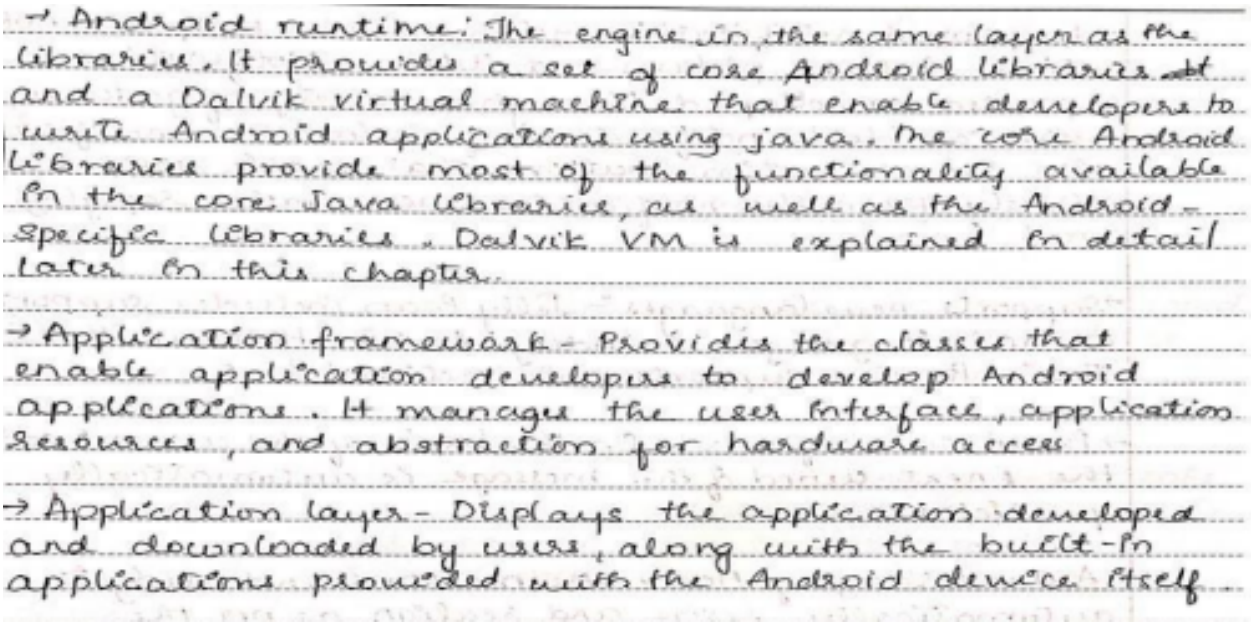


The following list gives a brief description of each layer in the software stack:

→ Linux kernel: The kernel on which Android is based contains device drivers for various hardware components of an Android device, including Display, Camera, keypad, wi-fi, Flash Memory and Audio.

→ Libraries: The next layer on top of the Linux kernel is the libraries that implement different Android features. A few of these libraries are listed here:

- Webkit library - Responsible for browser support
- Freetype library - Responsible for font support
- SQLite library - Provides database support
- Media library - Responsible for recording and playback of audio and video formats
- Surface Manager library - Provides graphics libraries that include SGL and OpenGL for 2D and 3D graphics support.



3.b. Explain steps involved in installing android.

One of the main factors determining the success of a smartphone platform is the applications that support it. Applications play a very vital role in determining whether a new platform swims or sinks. Making these applications accessible to the general user is extremely important. In August 2008, Google announced the Android Market, an online application store for Android devices, and made it available to users in October 2008. Using the Market application that is preinstalled on their Android device, users can simply download third-party applications directly onto their devices. Both paid and free applications are supported on the Android Market, though paid applications are available only to users in certain countries due to legal issues. **Google Playstore** is the current name of Android Market!!

OBTAINING THE REQUIRED TOOLS

To develop an android application, we need to install certain sofwares. It involves installing Eclipse, Android SDK, Android Development Tools (ADT) and creating Android Virtual Devices (AVD).

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

Eclipse

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\.

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.

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.

3.c. Explain steps involved in creating an android virtual device.

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.

4.a. How you will create and execute first android project? Explain with program steps.

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 Application Project.
2. Name the Android project suitably, say *HelloWorld*.
3. 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/hello” />

</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.

4.b. Write a android program to print “Hello Welcome” on emulator with a image in the background.

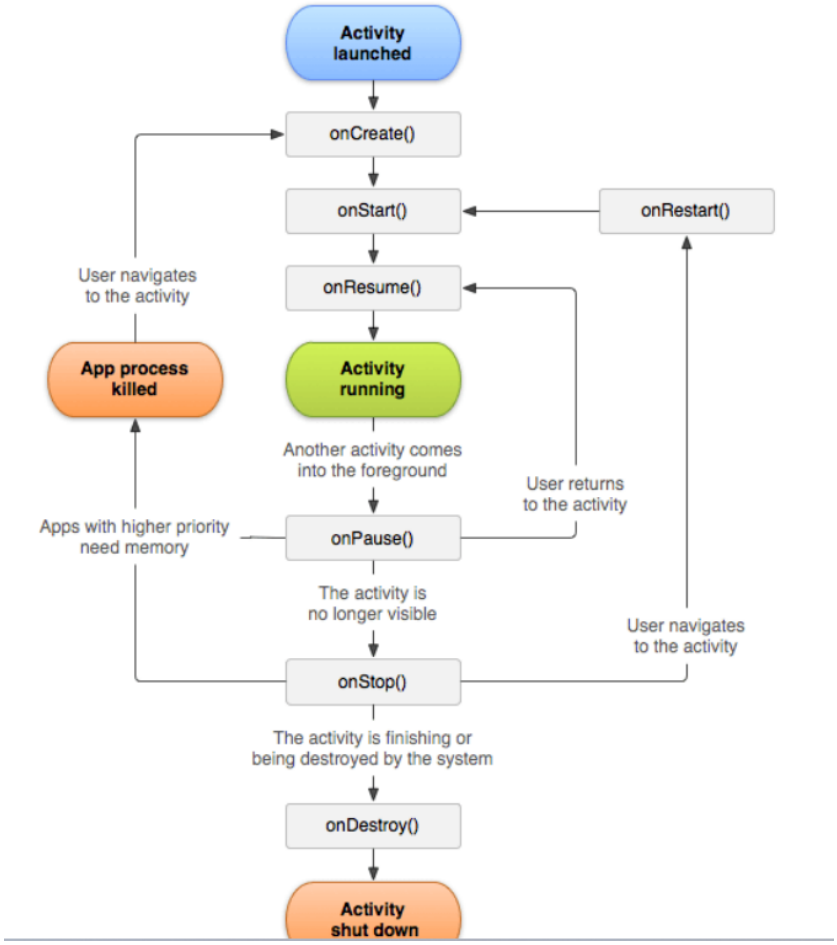
```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
```

```
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="20dp"
android:background="@drawable/background"
>
<TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="50dp"
        android:text="Hello Welcome"
        android:textSize="30sp"
        android:gravity="center"/>

</LinearLayout>
```

5.a. What is activity? Explain the activity life cycle in android with a neat diagram.


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

The Activity base class defines a series of events that governs the life cycle of an activity. Below Figure shows the life cycle of an activity and the various stages it goes through — from when the activity is started until it ends.

- onCreate() Called when the Activity is created. Setup is done here. Also provides access to any previously stored state in the form of a Bundle, which can be used to restore what the user was doing before this Activity was destroyed.
- onRestart() Called if the Activity is being restarted, if it's still in the stack, rather than starting new.
- onStart() Called when the Activity is becoming visible on the screen to the user.
- onResume() Called when the Activity starts interacting with the user. (This method is always called, whether starting or restarting.)
- onPause() Called when the Activity is pausing or reclaiming CPU and other resources. This method is where you should save state information so that when an Activity is restarted, it can start from the same state it was in when it quit.
- onStop() Called to stop the Activity and transition it to a nonvisible phase and subsequent lifecycle events.

- onDestroy() Called when an Activity is being completely removed from system memory. This method is called either because onFinish() is directly invoked or because the system decides to stop the Activity to free up resources.

5.b. Explain content providers in android.

A content provider acts as a data store and provides an interface to access its contents. Unlike a database, where information can be accessed only by the package in which it was created, information in a content provider can be shared across packages. The following lists a few characteristics of content providers:

> Like in a database, we can query, add, edit, delete, and update data in content providers.

» Data can be stored in a database, files, and over a network.

>» A content provider acts as a wrapper around the data store to make it resemble web services. That is, the data in content providers is exposed as a service.

Understanding the Android Content URI

To fetch data from a content provider, we specify the query string in the form of a URI (universal resource identifier). The syntax of the query URI appears as follows:

<standard_prefix>://<authority><data_path>/<id>

Content Providers Application

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

Activity_content_provider_app.xml

ContentProviderAppActivity.java

```
package com.androidunleashed.contentproviderapp;


import android.app.Activity;

import android.os.Bundle;

import android.net.Uri;

import android.database.Cursor;

import android.content.CursorLoader;

import android.provider.ContactsContract;

import android.widget.ListView;

import java.util.ArrayList;

import android.widget.ArrayAdapter;


public class ContentProviderAppActivity extends Activity {

   ArrayList<String> contactRows=new ArrayList<String>();

         final String[] nocontact={"No Contacts on the Device"};
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_content_provider_app);
    final ListView contactsList=(ListView) findViewById(R.id.contactslist);
    Uri contactsUri = Uri.parse("content://contacts/people");
    String[] projection = new String[] {ContactsContract.Contacts._ID,
ContactsContract.Contacts.DISPLAY_NAME };
    Cursor c;
    CursorLoader cursorLoader = new CursorLoader(this, contactsUri, projection, null, null ,
null);
    c = cursorLoader.loadInBackground();
   contactRows.clear();
   c.moveToFirst();
   while(c.isAfterLast()==false){
    String contactID = c.getString(c.getColumnIndex(ContactsContract.Contacts._ID));
                                                    String contactDisplayName =
c.getString(c.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
    contactRows.add(contactID+ " "+contactDisplayName);
    c.moveToNext();
    }
    if (c != null && !c.isClosed()) {
        c.close();
    }
   if(contactRows.isEmpty()) {
```

```
                ArrayAdapter<String> arrayAdpt=new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, nocontact);

        contactsList.setAdapter(arrayAdpt);

        }

        else {

                ArrayAdapter<String> arrayAdpt=new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, contactRows);

        contactsList.setAdapter(arrayAdpt);

        }

        }

}
```

We define a contactsuri URI for the Contacts provider. Thereafter, a projection String array is defined to specify the columns that we want to extract from the contacts database. With the help of a cursorLoader, we load rows from the Contacts provider and assign them to the cursor c. Thereafter, through a while loop, the information in the cursor is extracted. Because we want to display only the ID and contact name, the informationn the contactsContract.Contacts. ID and ContactsContract.Contacts. DISPLAY_NAME Columns is accessed and assigned to the contactRows ArrayList. If contactRows is not empty, an ArrayAdapter object called arrayAdpt is defined through it. Finally, a ListView control is filled with the information in the arrayaAdpt ArrayAdapter. To access information in the Contacts Provider in our app, we need to add the following permission into the AndroidManifest .xml file:

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

6.a Explain table layout and its features.

Table Layout
Table Layout is used to arrange the group of views into rows and columns.

Table Layout containers do not display a border line for their columns, rows or cells.

Need not specify row and column count
All elements may be of different size.

Syntax:
`<TableLayout`
 	
`android:layout_width="match_pa rent"`
 	
`android:layout_height="wrap_co ntent"`
 	
`android:id="@+id/table01"`
`android:shrinkColumns="1"`
`android:collapseColumns="2"`
`android:stretchColumns="3"> <TableRow>`
`</TableRow>`
`</TableLayout>`

Table Layout is slower compared to Grid Layout
Table Layout takes more memory than Grid Layout
Needs to add views by using <tableRow>

Q6b) write a program to illustrate different drawing graphics in android

Activity_main.xml:

```
<?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">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/imageView" />
</RelativeLayout>
```

MainActivity.java:

```
package com.example.ashwini.program4;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.drawable.BitmapDrawable;
import android.os.Bundle;
import android.widget.ImageView;

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

        //Creating a Bitmap
        Bitmap bg = Bitmap.createBitmap(720, 1280, Bitmap.Config.ARGB_8888);

        //Setting the Bitmap as background for the ImageView
        ImageView i = (ImageView) findViewById(R.id.imageView);
        i.setBackgroundDrawable(new BitmapDrawable(bg));

        //Creating the Canvas Object
        Canvas canvas = new Canvas(bg);
```

```
        //Creating the Paint Object and set its color & TextSize
        Paint paint = new Paint();
        paint.setColor(Color.BLUE);
        paint.setTextSize(50);

        //To draw a Rectangle
        canvas.drawText("Rectangle", 420, 150, paint);
        canvas.drawRect(400, 200, 650, 700, paint);

        //To draw a Circle
        canvas.drawText("Circle", 120, 150, paint);
        canvas.drawCircle(200, 350, 150, paint);

    }
}
```

Q6c) How animation is created in Android? Explain

Programmatically creating an animation

This example's animation technique uses an image bound to a sprite. In general, sprite refers to a two-dimensional image or animation that is overlaid onto a background or more complex graphical display. For this example, you'll move the sprite around the screen to give the appearance of a bouncing ball. To get started, create a new project called BouncingBall with a BounceActivity. You can copy and paste the code in the following listing for the BounceActivity.java file.

Listing 9.7 BounceActivity.java

```
public class BounceActivity extends Activity {
     protected static final int GUIUPDATEIDENTIFIER = 0x101;
     Thread myRefreshThread = null;
     BounceView myBounceView = null;
     Handler myGUIUpdateHandler = new Handler() {
          public void handleMessage(Message msg) {
               switch (msg.what) {
                    case BounceActivity.GUIUPDATEIDENTIFIER:
                         myBounceView.invalidate();
                         break;
               }
               super.handleMessage(msg);
          }
     };
     @Override
     public void onCreate(Bundle icicle) {

          super.onCreate(icicle);
          this.requestWindowFeature(Window.FEATURE_NO_TITLE);
          this.myBounceView = new BounceView(this);
          this.setContentView(this.myBounceView);
          new Thread(new RefreshRunner()).start();
     }
     class RefreshRunner implements Runnable {
          public void run() {
               while (!Thread.currentThread().isInterrupted()) {
                    Message message = new Message();
                    message.what = BounceActivity.GUIUPDATEIDENTIFIER;
                    BounceActivity.this.myGUIUpdateHandler
.sendMessage(message);
                    try {
                         Thread.sleep(100);
                    } catch (InterruptedException e) {
                         Thread.currentThread().interrupt();
                    }
               }
          }
     }
}
```

1 Create unique identifier

2 Create handler

3 Create view

4 Run animation

First we import the `Handler` and `Message` classes, and then we create a unique identifier to allow us to send a message back to our program to update the `View` in the main thread. We need to send a message telling the main thread to update the `View` each time the child thread has finished drawing the ball. Because different messages can be thrown by the system, we need to guarantee the uniqueness of our message to our handler by creating a unique identifier called `GUIUPDATEIDENTIFIER` (1). Next, we create the `Handler` that will process our messages to update the main `View` (2). A `Handler` allows us to send and process `Message` classes and `Runnable` objects associated with a thread's message queue.

`Handlers` are associated with a single thread and its message queue, but their methods can be called from any thread. Thus we can use the `Handler` to allow objects running in another thread to communicate changes in state back to the thread that spawned them, or vice versa.

We set up a `View` (3) and create the new thread. Finally, we create a `RefreshRunner` inner class implementing `Runnable` that will run unless something interrupts the thread, at which point a message is sent to the `Handler` to call `BounceView`'s `invalidate()` method (4). The `invalidate()` method invalidates the `View` and forces a refresh.

Listing 9.8 BounceView.java

```
public class BounceView extends View {
     protected Drawable mySprite;
     protected Point mySpritePos = new Point(0,0);
     protected enum HorizontalDirection {LEFT, RIGHT} ;
     protected enum VerticalDirection {UP, DOWN} ;
     protected HorizontalDirection myXDirection =
HorizontalDirection.RIGHT;
     protected VerticalDirection myYDirection = VerticalDirection.UP;
     public BounceView(Context context) {
          super(context);
this.setBackground(this.getResources().getDrawable(R.drawable.android));
this.mySprite =
     this.getResources().getDrawable(R.drawable.world);
     }
     @Override
     protected void onDraw(Canvas canvas) {
this.mySprite.setBounds(this.mySpritePos.x,
     this.mySpritePos.y,
     this.mySpritePos.x + 50, this.mySpritePos.y + 50);
          if (mySpritePos.x >= this.getWidth() -
mySprite.getBounds().width()) {
               this.myXDirection = HorizontalDirection.LEFT;
          } else if (mySpritePos.x <= 0) {
               this.myXDirection = HorizontalDirection.RIGHT;
          }
          if (mySpritePos.y >= this.getHeight() -
mySprite.getBounds().height()) {
               this.myYDirection = VerticalDirection.UP;
          } else if (mySpritePos.y <= 0) {
               this.myYDirection = VerticalDirection.DOWN;
          }
          if (this.myXDirection ==
HorizontalDirection.RIGHT) {
               this.mySpritePos.x += 10;
          } else {
               this.mySpritePos.x -= 10;
          }
          if (this.myYDirection ==
                                   VerticalDirection.DOWN) {
               this.mySpritePos.y += 10;
          } else {
               this.mySpritePos.y -= 10;
          }
          this.mySprite.draw(canvas);
     }
}
```

1 **Get image file and map to sprite**

2 **Set bounds of globe**

3 **Move ball left or right, up or down**

4 **Check if ball is trying to leave screen**

In this listing, we do all the real work of animating the image. First, we create a `Drawable` to hold the globe image and a `Point` that we use to position and track the globe as we animate it. Next, we create enumerations (`enums`) to hold directional values for horizontal and vertical directions, which we'll use to keep track of the moving globe. Then we map the globe to the `mySprite` variable and set the Android logo as the background for the animation ❶.

Now that we've done the setup work, we create a new `View` and set all the boundaries for the `Drawable` ❷. After that, we create simple conditional logic that detects whether the globe is trying to leave the screen; if it starts to leave the screen, we change its direction ❸. Then we provide simple conditional logic to keep the ball moving in the same direction if it hasn't encountered the bounds of the `View` ❹. Finally, we draw the globe using the `draw()` method.

Q7a) With an example for each, explain common views in Android.

Views are the building blocks of Android application's UI. Activities contain views, and View classes represent elements on the screen and are responsible for interacting with users through events. Every Android screen contains a hierarchical tree of View elements. These views come in a variety of shapes and sizes. Many of the views you'll need on a day-to-day basis are provided as part of the platform—text elements, input elements, images, buttons, and the like. In addition, you can create your own composite views and custom views when the need arises. You can place views into an Activity (and thus on the screen) either directly in code or by using an XML resource that's later inflated at runtime. Android provides a generous set of View classes in the android.view package. These classes range from familiar constructs such as the EditText, Spinner, and TextView that you've already seen in action, to more specialized widgets such as AnalogClock, Gallery, DatePicker, TimePicker, and VideoView.

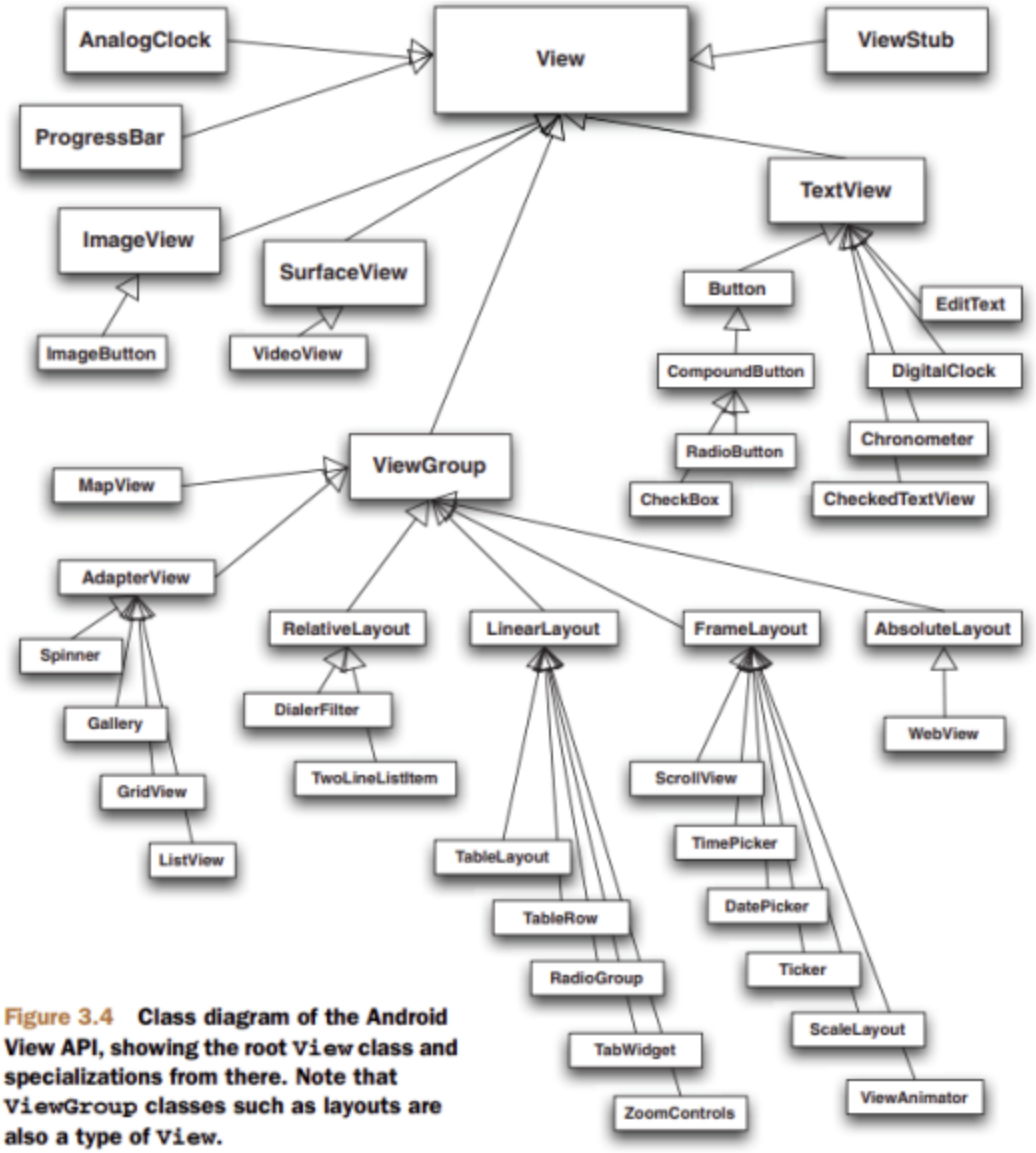


Figure 3.4 Class diagram of the Android View API, showing the root `View` class and specializations from there. Note that `ViewGroup` classes such as layouts are also a type of `View`.

View is the base class for many classes. ViewGroup is a special subclass of View related to layout, as are other elements such as the commonly used TextView. All UI classes are derived from the View class, including the layout classes (which extend ViewGroup).

Q7b) Explain different layouts in android.

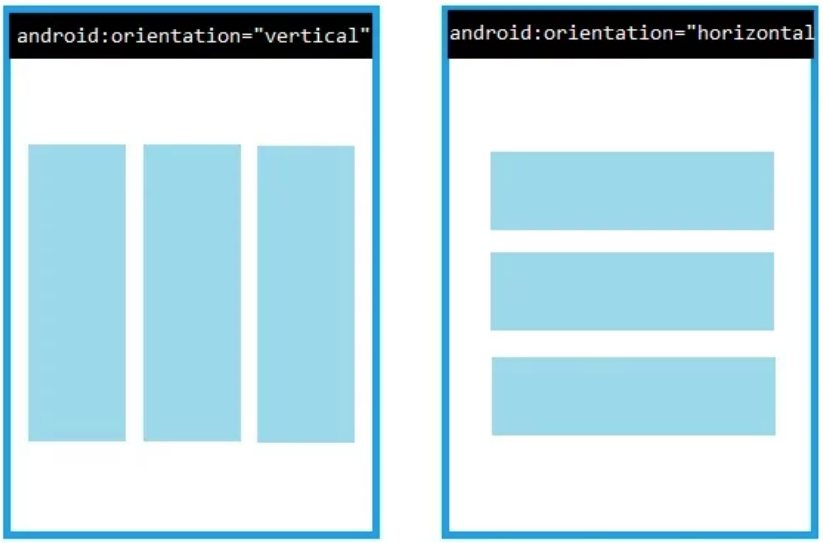
1. Linear Layout

This view group aligns all of its children in a single direction, either vertically or horizontally. The android:orientation attribute specifies the layout's direction.

The below code snippet demonstrates the inclusion of a Linear Layout in the XML file.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="horizontal">
<!-- Include other widget or layout tags here. These are considered
"child views" or "children" of the linear layout -->
</LinearLayout>
```

Setting **android:orientation** helps specify if the child views are displayed in row or column format



2. Relative Layout

RelativeLayout is a view group that places its child views relative to one another. Each view's location can be set relative to sibling components (such as to the left or below another view) or relative to the RelativeLayout area of its parent (such as aligned to the bottom, left or center).

3. Frame Layout

FrameLayout is designed to display a single object in a section of the screen. In general, FrameLayout should only be used to host a single child view, as it can be difficult to organize child views in a manner that is scalable to various screen sizes without the children overlapping. Nevertheless, you may add several children to a

FrameLayout and manage their position by providing gravity to each child using the android:layout gravity feature.

4. Constraint Layout

ConstraintLayout permits the creation of complicated layouts with a flat view hierarchy (no nested view groups). It is similar to RelativeLayout in that all views are set up according to the relationships between sibling views and the parent layout, but it is more flexible than RelativeLayout and easier to use with the Layout Editor in Android Studio. Because the layout API and the Layout Editor were purpose-built for one another, all the power of ConstraintLayout is accessible immediately from the Layout Editor's visual tools. So, developers may construct the layout with ConstraintLayout using only drag-and-drop instead of XML.

5. Android Table Layout

TableLayout is a ViewGroup that presents its children in rows and columns. TableLayout arranges its children into columns and rows. TableLayout containers do not display row, column, and cell border lines. The table will have the same number of columns as the row with the most cells. A table can leave cells vacant. As in HTML, cells can span multiple columns in Tableau. Developers can span columns using the TableRow's span field. Class layout parameters.

Q8a) How multimedia can be implemented in Android? Explain

Multimedia: Playing Audio,Playing Video and Capturing Media: Playing Audio: The audio file that we want to play must be located in the res/raw folder of our application.

The raw folder isn't created automatically, so we need to create it manually. The raw folder is a special folder that is not processed at all; hence the content of the files copied in this folder is retained. Right-click the res folder in the Package Explorer window and select New, Folder. In the dialog box that opens, enter the name of the new folder as raw and click the Finish button. In the raw folder, let's copy an audio file called song1.mp3. The Java class R. java file is automatically regenerated after the audio file is added to the application allowing us to access it.

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

```
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Playing Audio" />
    <Button android:id="@+id/playbtn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Play" />
</LinearLayout>
```

```
package com.androidunleashed.playaudioapp;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.view.View;
import android.media.MediaPlayer;

public class PlayAudioAppActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_play_audio_app);
        Button playButton = (Button) findViewById(R.id.playbtn);
        playButton.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                MediaPlayer mp =
                    MediaPlayer.create(PlayAudioAppActivity.this,R.raw.song1);
                mp.start();
            }
        });
    }
}
```

The playbtn Button control is captured from the layout and mapped to the Button object playButton. An event handler, onclickListener, is added to the playButton object. Its callback method, onclick(), is implemented, which is executed when the playButton is clicked. In the onclick() method, we create an instance, mp, of the MediaPlayer by calling the MediaPlayer.create() method. To the MediaPlayer instance mp, we pass the

reference of our MP3 song, song1.mp3, that we placed in the raw folder. Finally, we call the MediaPlayer's start () method to play the song. After the application is run, we get a TextView showing the text Playing Audio and a Button control with the text Play When the Play button is clicked, the audio, songl.mp3, is played. Playing Video: To play video in an application, Android provides a videoview control, which, along with the MediaController, provides several buttons for controlling video play. These buttons allow us to play, pause, rewind, and fast-forward the video content displayed via the VideovView control. To understand the steps for playing a video, let's create a new Android project called PlayvideoApp. We can play a video that is available on the Internet or one that is loaded onto an SD card of our device or emulator

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

```
package com.androidunleashed.playvideoapp;

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

public class PlayVideoAppActivity extends Activity {
```

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

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

Q8 b) Explain how to check internet connection in Android programmatically

This method checks whether mobile is connected to internet and returns true if connected:

private boolean isNetworkConnected() { ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE); return cm.getActiveNetworkInfo() != null && cm.getActiveNetworkInfo().isConnected(); }
in manifest,

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

This method actually checks if device is connected to internet(There is a possibility it's connected to a network but not to internet).

public boolean isInternetAvailable() { try { InetAddress ipAddr = InetAddress.getByName("google.com"); //You can replace it with your name return !ipAddr.equals(""); } catch (Exception e) { return false; } }

Q9a) Write a program to demonstrate GPS connection in Android

AndroidManifest.xml1 for fine and coarse permissions are as follows:

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

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

We are accessing Google Maps in the application; consequently, we also need to add the following two statements in the manifest file:

<uses-library android:name="com.google.android.maps" />

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

activity_know_location_app.xml

KnowLocationAppActivity.java

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:text="Latitude: "
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:id="@+id/latitude_view" />
    <TextView android:text="Longitude: "
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:id="@+id/longitude_view" />
</LinearLayout>
```

KnowLocationAppActivity.java

```
package com.androidunleashed.knowlocationapp;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.location.LocationManager;
import android.content.Context;
import android.location.Location;
import android.location.LocationListener;

public class KnowLocationAppActivity extends Activity {
    private TextView latitudeView;
    private TextView longitudeView;
    private LocationManager locationManager;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_know_location_app);
        latitudeView = (TextView) findViewById(R.id.latitude_view);
        longitudeView = (TextView) findViewById(R.id.longitude_view);
        locationManager = (LocationManager) getSystemService(Context.LOCATION_SER-
VICE);    #1
        locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
new LocationListener(){
            public void onProviderDisabled(String provider) { }
            public void onProviderEnabled(String provider) {}
            public void onStatusChanged(String provider, int status, Bundle extras)
{}
            public void onLocationChanged(Location loc) {
                if (loc != null) {
                    int lt = (int) (loc.getLatitude());
                    int lg = (int) (loc.getLongitude());
                    latitudeView.setText("Latitude is: "+String.valueOf(lt));
                    longitudeView.setText("Longitude is: "+String.valueOf(lg));
                }
            }
        });
    }
}
```

Q9 b) Explain procedure of sending email in Android project

To send an email with Android, we use the following

Intent: Intent .ACTION_ SEND

The Intent .ACTION SEND calls an existing email client to send an email. So, for sending email through the Android emulator, we need to first configure the email client. If the email client is not configured, the emulator does not respond to the Intent. Through the ACTION SEND Intent, messages of different types, such as text or image, can be sent. The only thing we need to do is to set the type of the Intent through the setType() method. For example, the following statement declares that the text data type will be sent through the Intent:

emailIntent.setType("plain/text") ;

The emailiIntent is the Intent .ACTION_SEND object. After we launch this Intent, any application on the device that supports plain text messaging may handle this request. To let a user choose the email client to handle the Intent, we call startActivity() with the createChooser() method. As the name suggests, the createChooser() method prompts the user to choose the application to handle the Intent. This statement shows how to call startActivity() with the createChooser() method:

startActivity (Intent.createChooser(emailIntent, "Sending Email"));

This statement displays all the applications that are eligible to handle the Intent, allowing the user to choose the application to launch. If there is only one application able to handle the Intent, it is launched automatically without prompting the user. To supply data for the email message fields, we set certain standard extras for the Intent. For example, we can set the following:

· EXTRA_EMAIL—Sets the To: address (email address of the receiver)

· EXTRA_cc—Sets the cc: address (email address of the carbon copy receiver)

· EXTRA_BCC—Sets the Bcc: address (email address of the blind carbon copy receiver)

· EXTRA_SUBJECT—Sets the Subject of the email VSN DON, VOY.

· EXTRA_TEXT—Sets the body of the email

After setting the desired Intent's extras, launch the activity to initiate the sending email task.

activity_send_email_app.xml

```
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent" >
   <TextView
         android:id="@+id/email_form"
         android:text = "Email Form"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
         android:typeface="serif"
         android:textSize="18sp"
         android:textStyle="bold"
         android:padding="10dip"
         android:layout_centerHorizontal="true"/>
   <TextView
      android:id="@+id/to_addressview"
         android:text = "To:"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
         android:layout_margin="10dip"
```

```
        android:layout_below="@id/email_form" />
    <EditText
        android:id="@+id/toaddresses"
       android:layout_height="wrap_content"
       android:layout_width="match_parent"
       android:layout_below="@id/email_form"
        android:layout_toRightOf="@id/to_addressview"
        android:singleLine="true" />
    <TextView
       android:id="@+id/cc_addressview"
        android:text = "Cc:"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_below="@id/to_addressview"
        android:layout_margin="10dip" />
    <EditText
        android:id="@+id/ccaddresses"
       android:layout_height="wrap_content"
       android:layout_width="match_parent"
        android:singleLine="true"
       android:layout_below="@id/toaddresses"
       android:layout_toRightOf="@id/cc_addressview" />
    <TextView
```

```
    android:id="@+id/bcc_addressview"
      android:text = "Bcc:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/cc_addressview"
      android:layout_margin="10dip" />
<EditText
    android:id="@+id/bccaddresses"
    android:layout_height="wrap_content"
      android:layout_width="match_parent"
      android:singleLine="true"
    android:layout_below="@id/ccaddresses"
    android:layout_toRightOf="@id/bcc_addressview" />
<TextView
      android:id="@+id/subjectview"
      android:text = "Subject:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/bcc_addressview"
      android:layout_margin="10dip"
    android:paddingTop="10dip"/>
<EditText
      android:id="@+id/emailsubject"
```

```
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
      android:singleLine="true"
    android:layout_below="@id/bccaddresses"
    android:layout_toRightOf="@id/subjectview"
      android:layout_marginTop="10dip" />
<TextView
    android:id="@+id/emailtextview"
      android:text = "Message:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/subjectview"
    android:layout_margin="10dip" />
<EditText
      android:id="@+id/emailtext"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
      android:lines="5"
    android:layout_below="@id/emailsubject"
    android:layout_toRightOf="@id/emailtextview" />
<Button
      android:id="@+id/send_button"
      android:text="Send"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
          android:layout_centerHorizontal="true"
          android:paddingLeft="25dip"
          android:paddingRight="25dip"
        android:layout_marginTop="10dip"
        android:layout_below="@id/emailtext" />
</RelativeLayout>
```

SendEmailAppActivity.java

```
package com.androidunleashed.sendemailapp;

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

```
public class SendEmailAppActivity extends Activity {
    Button sendbutton;
    EditText toAddress, ccAddress, bccAddress, subject, emailMessage;
    String  toAdds, ccAdds, bccAdds;

    @Override
    public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_send_email_app);
          sendbutton=(Button) findViewById(R.id.send_button);
          toAddress=(EditText) findViewById(R.id.toaddresses);
          ccAddress=(EditText) findViewById(R.id.ccaddresses);
          bccAddress=(EditText) findViewById(R.id.bccaddresses);
          subject=(EditText) findViewById(R.id.emailsubject);
          emailMessage=(EditText) findViewById(R.id.emailtext);
          sendbutton.setOnClickListener(new OnClickListener(){
          @Override
          public void onClick(View v) {
                    final Intent emailIntent = new Intent(Intent.ACTION_SEND);
                    if(toAddress.getText().length() >0) {
                    toAdds = ""+toAddress.getText().toString()+"";
                   emailIntent.putExtra(Intent.EXTRA_EMAIL, new String[]{ toAdds});
```

```
                }
                if(ccAddress.getText().length() >0) {
                ccAdds = ""+ccAddress.getText().toString()+"";
                emailIntent.putExtra(Intent.EXTRA_CC, new String[]{ ccAdds});
                }
            if(bccAddress.getText().length() >0) {
                bccAdds = ""+bccAddress.getText().toString()+"";
               emailIntent.putExtra(Intent.EXTRA_BCC, new String[]{ bccAdds});
                }
            emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject.getText().toString());
            emailIntent.putExtra(Intent.EXTRA_TEXT, emailMessage.getText());
            emailIntent.setType("plain/text");
            startActivity(Intent.createChooser(emailIntent, "Sending Email"));
        }
        });
    }
}
```

Here, the toaddresses, ccaddresses, bocaddresses, emailsubject, and emailtext EditText controls are accessed and mapped to their respective EditText objects. Similarly, the Button control with the send_button ID is accessed and assigned to the sendbutton Button object. The clickListener event listener is associated with the Button control. After we click the Button, the onCclick() callback method is invoked. In the onclick () method, an Intent object called emailiIntent is created, with its action set to Intent. ACTION_SEND. Also, to auto-fill the email client's fields, the Intent extras, EXTRA EMAIL, EXTRA_CC, EXTRA BCC, EXTRA_SUBJECT, and EXTRA _TEXT, are initialized with the data entered by the user in the EditText controls. Finally, the email is

sent by launching the email client installed on the device. After running the application, we see a user interface to enter addresses of the receiver(s), subject, and email body. When the user clicks the send button, the email client is invoked. The fields of the email client are automatically filled with data entered in the EditText controls. The email is sent to the recipient after it pipes through the email client,

Q10 a) Explain services in android

CREATING YOUR OWN 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. In this case, the code that is playing the background music has no need to interact with the user, and hence it can be run as a service. Services are also ideal for situations in which there is no need to present a UI to the user.

Following are the steps involved in creating own service.

- Create a new android application.
- Add a new class file to the project and name it MyService.java. Write the following code in it:

```
package net.learn2develop.Services; import
android.app.Service;
import android.content.Intent; import
android.os.IBinder; import
android.widget.Toast;

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

        public int onStartCommand(Intent intent, int flags, int startId)
        {
              Toast.makeText(this,"ServiceStarted",Toast.LENGTH_LONG).show();
               return START_STICKY;
        }

        public void onDestroy()
        {
             super.onDestroy();
             Toast.makeText(this,"ServiceDestroyed",Toast.LENGTH_LONG).show();
        }
}
```

The onBind()method enables you to bind an activity to a service. This in turn enables an activity to directly access members and methods inside a service. The onStartCommand() method is called when you start the service explicitly using the startService() method. This method signifies the start of the service, and you code it

to do the things you need to do for your service. In this method, you returned the constant START_STICKY so that the service will continue to run until it is explicitly stopped. The onDestroy() method is called when the service is stopped using the stopService() method. This is where you clean up the resources used by your service.

- In the AndroidManifest.xml file, add the following statement :

```
<service android:name=".MyService" />
```

- In the activity_main.xml file, add the following statements in bold:

```
<Button android:id="@+id/btnStartService"
android:layout_width="fill_parent"
android:layout_height="wrap_content" android:text="Start
Service" />
<Button android:id="@+id/btnStopService"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Stop Service" />
```

- Add the following statements in bold to the MainActivity.java file:

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

public class MainActivity extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);

     Button btnStart = (Button) findViewById(R.id.btnStartService);

     btnStart.setOnClickListener(new View.OnClickListener()
     {
              public void onClick(View v)
              {
        startService(new Intent(getBaseContext(), MyService.class));
              }
     });
```

```
Button btnStop = (Button) findViewById(R.id.btnStopService);
btnStop.setOnClickListener(new View.OnClickListener()
{
    public void onClick(View v)
    {
        stopService(new Intent(getBaseContext(), MyService.class));
    }
  });
}
}
```

Q10b) Explain procedure involved in publishing android application in google play store

Following are the steps to publish an Android application:

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

Setting Versioning Information of an Application

We need three things to essentially inform users about our application:

» The features that our application requires to run—for example, whether it requires Bluetooth, multitouch screen, and so on, to operate

» The necessary hardware configuration that our application requires for running

» The permissions that our application requires to operate

To inform users about these three things, we use the three tags <uses-features>, <uses-configuration>, and <uses-permission> in our AndroidManifest.xml file.

Generating a Certificate, Digitally Signing the Android Applications, and Generating the APK

Signing Applications Using the Export Android Application Wizard

The Export Android Application Wizard simplifies the process of creating and signing our application package. To launch the wizard, follow these steps:

1. Select the Android project in the Package Explorer window and select the File, Export Option or right-click the project in the Package Explorer window and select the Export option.

2. In the Export dialog, expand the Android item and select Export Android Application. Click Next.

3. Our Android project name, createServiceApp, is displayed in the Project box. Click Next.

4. The next dialog prompts us to either select an existing keystore or create a new one. Select the create new keystore option to create a new certificate, that is, keystore, for our application. In the Location box, specify the name and path of the keystore. Assign the name as CreateService to our new keystore. In the Password and Confirm boxes, enter the password to protect the keystore. After entering the password , click Next.

5. Provide an alias for the private key and enter a password to protect the private key. Applications published on Google Play require a certificate with a validity period ending after October 22, 2033. Hence, enter a number that is greater than 2033 minus the current year in the validity box. Also fill the First and Last Name box with your name and click Next.

6. Enter a path to store the destination APK file. Click Finish.

Once you have signed your APK files, you need a way to get them onto your users' devices. Three methods are here:

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