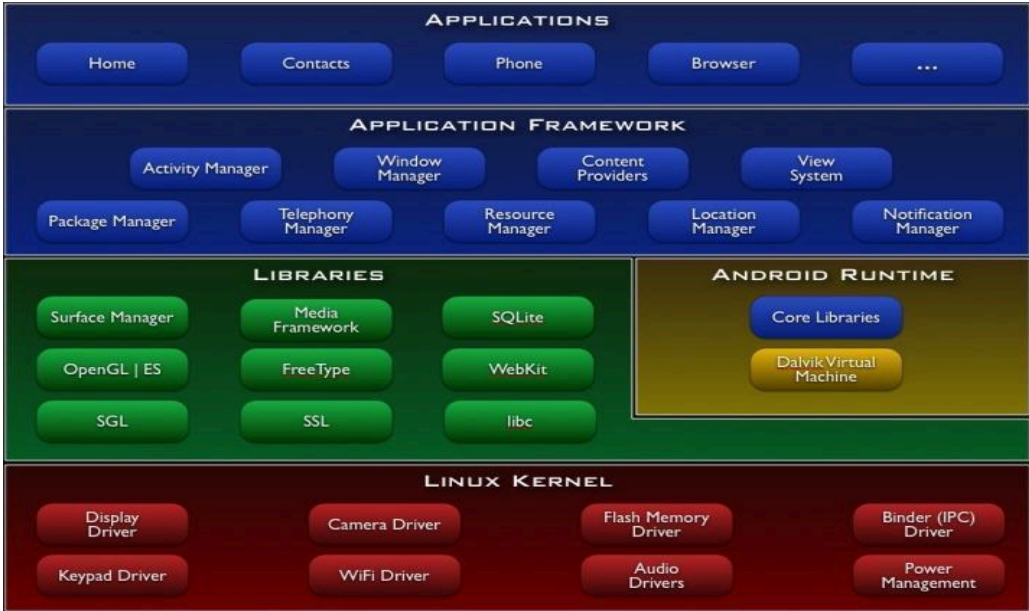


--	--	--	--	--	--	--	--	--	--	--

Internal Assessment Test –II, September 2024

Sub:	Mobile Application Development	Code:	22MCA263	
Answer Key		Marks	OBE	
			CO	RBT
1	<p>Explain the Android Software Stack in detail with a neat diagram.</p> <p>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.</p>  <p>1) Linux kernel</p> <p>It is the heart of android architecture that exists at the root of android architecture.</p> <p>Linux kernel is responsible for device drivers, power management, memory management, device management and resource access.</p> <p>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.</p>	10	CO2	L3

2) Native 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:

- 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.

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].

- a. 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.

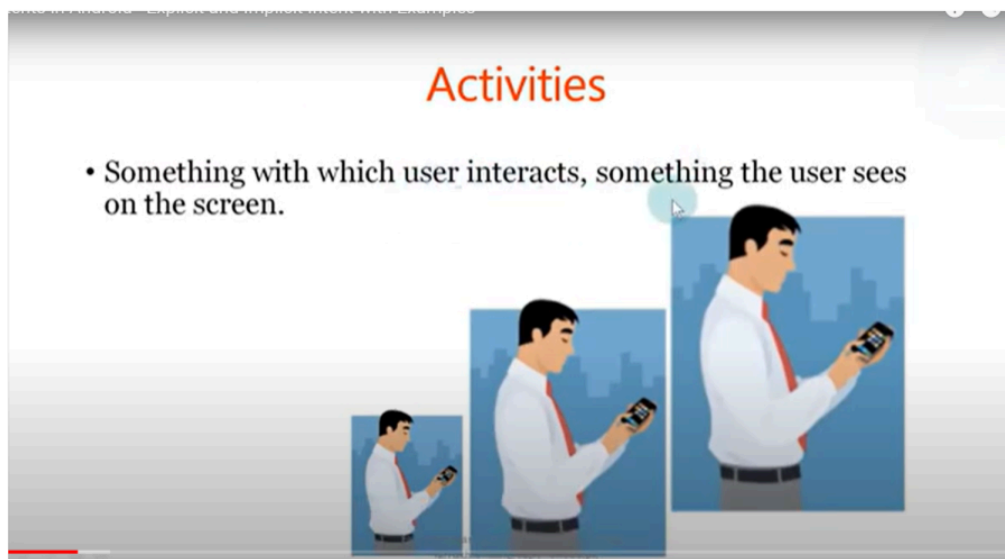
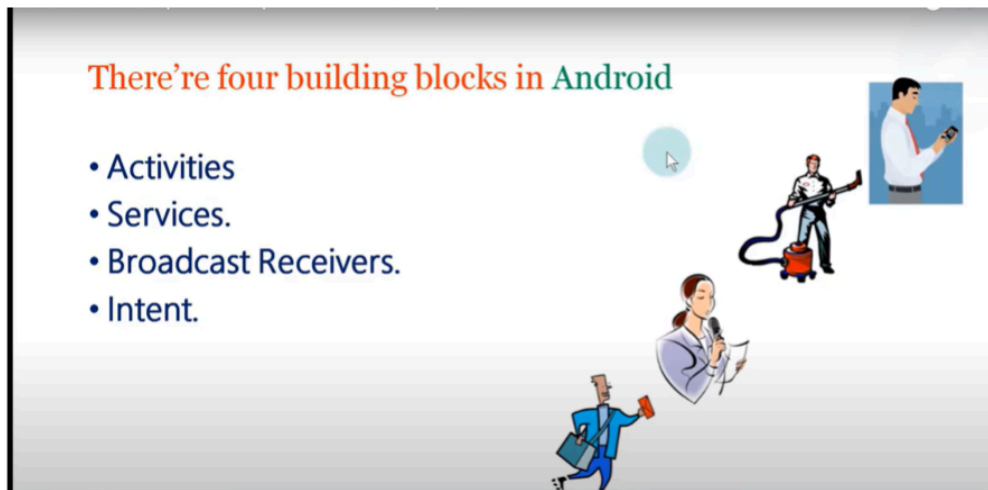
Android runtime and native libraries are using linux kernal.

2 Explain the building blocks in Android.

10

CO2

L2





Services

- Something that runs in the background.
- These are used to perform the long tasks, like downloading a file, connecting to a mail service, connecting to a server, uploading a file to a drop box and stuff like that.
- All these things happen in services.
- User can't see them, but they started by activities and such operations run independently. Even if your activity is off your services may run in the background.

Broadcast Receivers



- These are functions who sleeps all the time, but when suddenly something happens, they wake up and they do something.
- For example if your battery gets low, and you have broadcast receiver inside your app who wakes up and shut down all the downloads.
- So that's what broadcast receivers do. These are a kind of alarms that you want ring when a specific event triggered.



What an intent does?



- It is used to start a new activity from where ever you are. It is like going to a different page in html from one page.
- If you want to download something your download services begin in the background in case you're downloading a big file, or you want to play music in the background, so intents are used to start all these services.
- The third thing that intents are used register the broadcast receivers.
- Intents are also used to tell the system, which is the entry point of your application inside of your application, and which activity should appear in launcher screen, or inside the list of application installed on your android OS.

Types of Intents

1. Explicit Intents
2. Implicit Intents



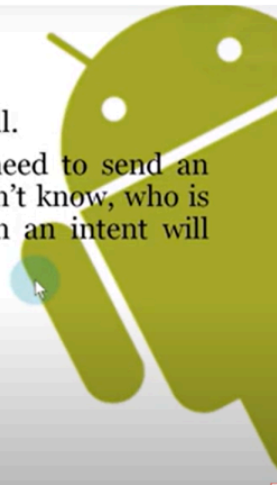
The image shows a composite of two parts. On the left is a screenshot of a mobile application titled "Desi Food Recipes" with a search bar and a grid of food images categorized by meal type (Breakfast, Appetizers, Lunch, Dinner, Desserts, Snacks). On the right is a presentation slide titled "Explicit Intents" with a bullet point: "• Explicit intents are those intents, when you know the activity you are calling." The slide also features the green Android robot logo on the right.

3 What is intent? Explain implicit intent with an example.

10 CO3 L3

Implicit Intents

- Implicit intents, when you don't know who to call.
- You know you need to send a SMS, or you need to send an Email, when know the functionality but you don't know, who is going to do that function for you, that's when an intent will become implicit intent.



- So whenever there is an event and an intent is created out of that event, it will be supplied to everybody.
- Suppose you've a **broadcast receiver**, who is actively listening to such events, then that **broadcast receiver** is going to be connected.
- There are hundreds of event in Android device that converted into an intent object, and that intent broadcast to everybody in the Android OS.

Program1.xml

```

<Button
    android:id="@+id/btn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerVertical="true"
    android:text="Search" />

<EditText
    android:id="@+id/editText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="21dp"
    android:layout_marginTop="63dp"
    android:layout_toRightOf="@+id/btn"
    android:ems="10" >

    <requestFocus />
</EditText>

```

Program1.java

```

EditText editText;
Button button;
button = (Button)findViewById(R.id.btn);
editText = (EditText) findViewById(R.id.editText);

button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String url=editText.getText().toString();
        Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
        startActivity(intent);
    }
});

```

4	<p>What is ViewGroup? Explain linear layout with properties.</p> <p>VIEW GROUPS(Layout Managers)</p> <p>One or more views can be grouped together into a ViewGroup. A ViewGroup (which is itself a special type of view) provides the layout in which you can order the appearance and sequence of views. Examples of ViewGroups include LinearLayout and FrameLayout. A ViewGroup derives from the base class android.view.ViewGroup.</p> <p>4 LinearLayout</p> <p>The LinearLayout arranges views in a single column or a single row.</p> <p>LinearLayout:-</p> <p>The LinearLayout is the most basic layout, and it arranges its elements sequentially, either horizontally or vertically. To arrange controls within a linear layout, the following attributes are used:</p> <ol style="list-style-type: none"> 1. android:orientation—Used for arranging the controls in the container in horizontal or vertical order. android:orientation="vertical" 2. android:layout_width—Used for defining the width of a control. android:layout_width="20px" 3. android:layout_height—Used for defining the height of a control. android:layout_height="20px" 4. android:padding—Used for increasing the whitespace between the boundaries of the control and its actual content. android:padding="5dip" android:paddingLeft="5dip" 5. android:layout_weight—Used for shrinking or expanding the size of the control to consume the extra space relative to the other controls in the container.the value of the weight attribute range from 0.0 to 1.0, where 1.0 is the highest value. android:layout_weight="0.0" 6. android:gravity—Used for aligning content within a control. Android:gravity includes left,center,right,top,bottom,center_horizontal,center_vertical,fill_horizontal, and fill_vertical. android:gravity="center" <ol style="list-style-type: none"> a. center_vertical—Places the object in the vertical center of its container, without changing its size. b. fill_vertical—Grows the vertical size of the object, if needed, so it completely fills its container. c. center_horizontal—Places the object in the horizontal center of its container, without changing its size. d. fill_horizontal—Grows the horizontal size of the object, if needed, so it completely fills its container e. center—Places the object in the center of its container in both the vertical and horizontal axis,without changing its size. 	10	CO4	L3
---	---	----	-----	----

7. android:layout_gravity—Used for aligning the control within the container(left,center,right).

android:layout_gravity="center"

- Child views can be arranged either vertically or horizontally.
- To see how LinearLayout works, consider the following elements typically contained in the activity_main.xml file:

```
<?xml version="1.0" encoding="Utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="Vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="string/hello" />
</LinearLayout>
```

5 Discuss the lifecycle of activity with a diagram.

10

CO2

L1

Android Activity Lifecycle

Android Activity Lifecycle is controlled by 7 methods of android.app.Activity class.

An activity is the single screen in android. It is like window or frame of Java. By the help of activity, you can place all your UI components or widgets in a single screen.

The 7 lifecycle method of Activity describes how activity will behave at different states.



Android Activity Lifecycle methods

Let's see the 7 lifecycle methods of android activity.

Method	Description
onCreate	called when activity is first created.
onStart	called when activity is becoming visible to the user.
onResume	called when activity will start interacting with the user.
onPause	called when activity is not visible to the user.
onStop	called when activity is no longer visible to the user.
onRestart	called after your activity is stopped, prior to start.
onDestroy	called before the activity is destroyed.

Android Activity Lifecycle methods

1. onCreate(Bundle savedInstanceState)

•State: Created

•Description: Called when the activity is first created. This is where you should initialize your activity, including setting up the UI (using setContentView) and initializing data. This method is only called once during the entire lifecycle of the activity.

2. onStart()

•State: Started

•Description: Called when the activity becomes visible to the user. At this point, the activity is not yet interactive.

3. onResume()

•State: Resumed

•Description: Called when the activity starts interacting with the user. The activity is now at the top of the activity stack, and the user can interact with it. This is where the activity enters the foreground.

4. onPause()

•State: Paused

•Description: Called when the system is about to start another activity. The current activity is still partially visible but not in the foreground. You should use this method to pause any ongoing tasks that should not continue while the activity is not in the foreground.

5. onStop()

•State: Stopped

•Description: Called when the activity is no longer visible to the user. This happens when another activity has taken over the screen. You should use this method to release resources that are not needed while the activity is not visible.

6. onRestart()

•State: Restarted

•Description: Called after the activity has been stopped, just before it is started again. This is useful for refreshing any resources or data that were released in onStop().

7. onDestroy()

•State: Destroyed

•Description: Called before the activity is destroyed. This can happen when the user manually finishes the activity, or the system destroys it to free up resources. You should use this method to clean up any resources that are not automatically handled by the garbage collector.

6

What are the different attributes of relative layout? Explain with an example.

RelativeLayout

The RelativeLayout enables you to specify how child views are positioned relative to each other. Consider the following main.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:id="@+id/RLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

10

CO3

L3

```

xmlns:android="http://schemas.android.com/apk/res/android"
>
<TextView
android:id="@+id/lblComments"
android:layout_width="wrap_content"
android:layout_height="wrap_content" android:text="Comments"
android:layout_alignParentTop="true"
android:layout_alignParentLeft="true"
/>
<EditText
android:id="@+id/txtComments"
android:layout_width="fill_parent"

android:layout_height="170px"
android:textSize="18sp"
android:layout_alignLeft="@+id/lblComments"
android:layout_below="@+id/lblComments"
android:layout_centerHorizontal="true"
/>
<Button
android:id="@+id/btnSave"
android:layout_width="125px"
android:layout_height="wrap_content"
android:text="Save"
android:layout_below="@+id/txtComments"
android:layout_alignRight="@+id/txtComments"
/>
<Button
android:id="@+id/btnCancel"
android:layout_width="124px"
android:layout_height="wrap_content"
android:text="Cancel"
android:layout_below="@+id/txtComments"
android:layout_alignLeft="@+id/txtComments"
/>
</RelativeLayout>

```

The UI of the above code would look like –



- Each view is embedded within the relative layout has attributes that enable it to align with another view.
- The value for each of these attributes is the **ID** for the view that you are **referencing**.
- These **attributes** are as follows:
 - `layout_alignParentTop`
 - `layout_alignParentLeft`
 - `layout_alignLeft`

 - `layout_alignRight`
 - `layout_below`
 - `layout_centerHorizontal`

The attributes used to set the location of the control relative to a container are
android: layout _alignParentTop—The top of the control is set to align with the top of the container.

android: layout _alignParentBottom—The bottom of the control is set to align with the bottom of the container.

android: layout _alignParentLeft—The left side of the control is set to align with the left side of the container.

android: layout _alignParentRight—The right side of the control is set to align with the right side of the container.

android: layout _centerHorizontal—The control is placed horizontally at the center of the container.

android: layout _centerVertical—The control is placed vertically at the center of the container.

android: layout _centerInParent—The control is placed horizontally and vertically at the center of the container.

The attributes to control the position of a control in relation to other controls are

android: layout _above—The control is placed above the referenced control.

android: layout _below—The control is placed below the referenced control.

android:layout_toLeftof—The control is placed to the left of the referenced control.

android:layout_toRightof—The control is placed to the right of the referenced control.

The attributes that control the alignment of a control in relation to other controls are

android:layout_alignTop— The top of the control is set to align with the top of the referenced control.

android:layout_alignBottom—The bottom of the control is set to align with the bottom of the referenced control.

android:layout_alignLeft—The left side of the control is set to align with the left side of the referenced control.

android:layout_alignRight—The right side of the control is set to align with the right side of the referenced control.

android:layout_alignBaseline—The baseline of the two controls will be aligned.

android:padding—Defines the spacing of the content on all four sides of the control.

To define padding for each side individually, use
android:paddingLeft,

android:paddingRight, **android:paddingTop**, and **android:paddingBottom**.

android:paddingTop—Defines the spacing between the content and the top of the control.

android:paddingBottom—Defines the spacing between the content and the bottom of the control.

android:paddingLeft—Defines the spacing between the content and the left side of the control.

android:paddingRight—Defines the spacing between the content and the right side of the control.

android:layout_margin—Defines the spacing of the control in relation to the controls or the container on all four sides. To define spacing for each side individually, we use the

android:layout_marginTop—Defines the spacing between the top of the control and the related control or container.

android:layout_marginBottom—Defines the spacing between the bottom of the control and the related control or container.

android:layout_marginRight—Defines the spacing between the right side of the control and the related control or container.

android:layout_marginLeft—Defines the spacing between the left side of the control and the related control or container.

7

Develop a mobile application to create a login form by using table layout.

10

CO3

L3

```

<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent">
    <TableRow>
    <TextView
        android:text="User Name:"
        android:width
        ="120px" />
    <EditText
        android:id="@+id/txtUserName" android:width="200px"
        />
    </TableRow>
    <TableRow>
    <TextView
        android:text="Password:" />
    <EditText
        android:id="@+id/txtPassword" android:password="true"
        />
    </TableRow>
    <TableRow>
    <TextView />
    <CheckBox android:id="@+id/chkRememberPassword"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Remember Password"/>
    </TableRow>
    <TableRow>
    <Button
        android:id="@+id/buttonSignIn" android:text="Log In" />
    </TableRow>
</TableLayout>

```

The above code result into following GUI:



8	Create a mobile app to change textview attributes from XML and Java code.	10	CO3	L3
9	<p>What is canvas? Explain rectangle, circle, and arc with examples.</p> <pre> public class MainActivity extends Activity { DemoView demoview; @Override protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity_main); demoview = new DemoView(this); setContentView(demoview); } private class DemoView extends View{ public DemoView(Context context){ super(context); } protected void onDraw(Canvas canvas) { super.onDraw(canvas); Paint paint = new Paint(); paint.setStyle(Paint.Style.STROKE); // make the entire canvas white paint.setColor(Color.WHITE); canvas.drawPaint(paint); // draw blue circle with anti aliasing turned off </pre>	10	CO4	L3

```

    paint.setColor(Color.BLUE);
    canvas.drawCircle(20, 20, 15, paint);
// draw red rectangle with anti aliasing turned off

    paint.setColor(Color.RED);
    paint.setStyle(Paint.Style.STROKE);
    canvas.drawRect(100, 5, 200, 30, paint);
//draw arc
    paint.setColor(Color.RED);
    RectF rect1=new RectF(200,400,300,550);
    canvas.drawArc(rect1, 90, 270, true, paint);

canvas.restore();
}
}

```

10 What is Handover? Explain possible handover scenarios of GSM.

10 CO1 L1

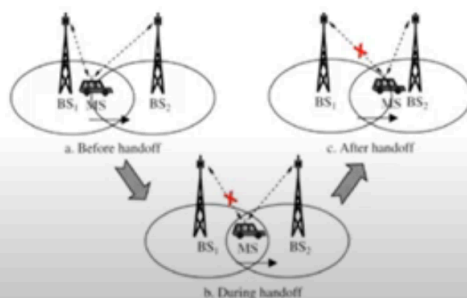
- Cellular systems require handover procedures, as single cells do not cover the whole service area, but, e.g., only up to 35 km around each antenna on the countryside and some hundred meters in cities.
- The smaller the cell size and the faster the movement of a mobile station through the cells (up to 250 km/h for GSM), the more handovers of ongoing calls are required.
- 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.

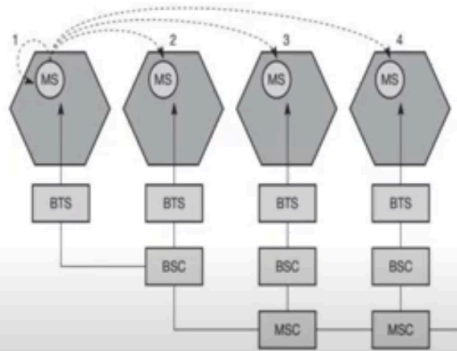
1. The mobile station moves out of the range of a BTS or a certain antenna of a BTS respectively.
2. Handover may be due to load balancing.

GSM-Handover

Hard handover - The connection to the source is broken before or 'as' the connection to the target is made—for this reason such handovers are also known as *break-before-make*.



GSM-Handover



Types of handover
in GSM

GSM-Handover

- Four possible handover scenarios in GSM:

- **Intra-cell handover:**

- Within a cell, narrow-band interference could make transmission at a certain frequency impossible.
- The BSC could then decide to change the carrier frequency.

- **Inter-cell, intra-BSC handover:**

- The mobile station moves from one cell to another, but stays within the control of the same BSC.
- The BSC then performs a handover, assigns a new radio channel in the new cell and releases the 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.