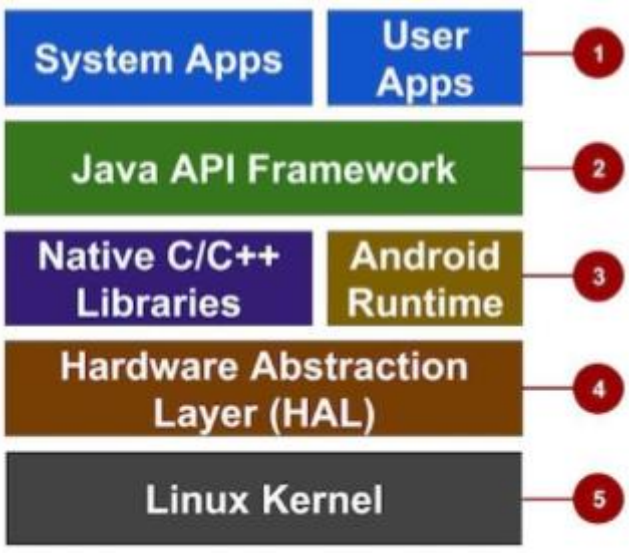
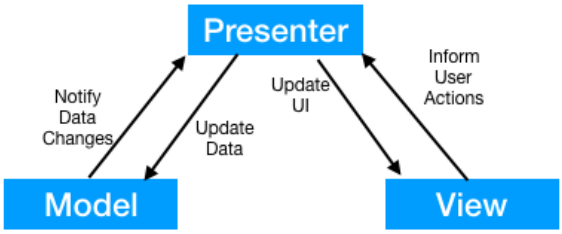
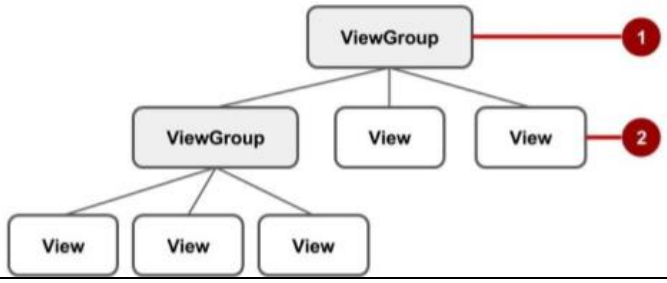
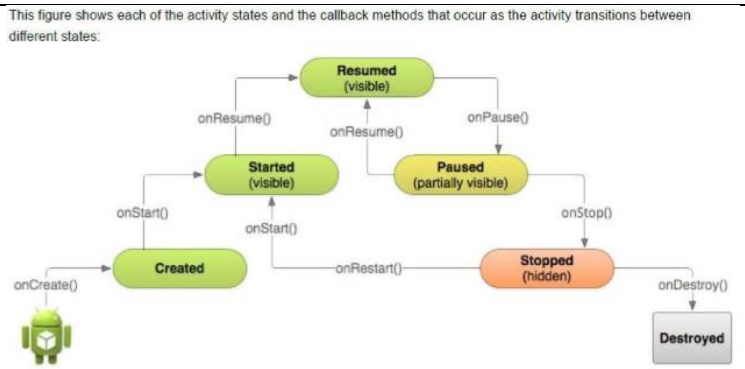
Internal Assessment Test 1 – March 2019										
Sub:	Mobile Application Development					Sub Code:	15CS661	Branch:	CSE	
Date:	07/03/19	Duration:	90 min's	Max Marks:	50	Sem / Sec:	VI B/C		OBE	

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1 (a)	**What is Android?** Answer: Android is an operating system and programming platform developed by Google for smartphones and other mobile devices (such as tablets). It can run on many different devices from many different manufacturers. Android includes a software development kit for writing original code and assembling software modules to create apps for Android users. It also provides a marketplace to distribute apps. Altogether, Android represents an ecosystem for mobile apps.	[03]	CO1	L1
(b)	**Explain Android development architecture using a diagram.** Answer: Android provides rich development architecture. You don't need to know much about the components of this architecture, but it is useful to know what is available in the system for your app to use. The following diagram shows the major components of the Android stack — the operating system and development architecture.  In the figure above: 1. Apps: Your apps live at this level, along with core system apps for email, SMS messaging, calendars, Internet browsing, or contacts. 2. Java API Framework: All features of Android are available to developers through application programming interfaces (APIs) written in the Java language. You don't need to know the details of all of the APIs to learn how to develop Android apps, but you can learn more about the following APIs, which are useful for creating apps: View System used to build an app's UI, including lists, buttons, and menus. Resource Manager used to access to	[07]	CO1	L4

	non-code resources such as localized strings, graphics, and layout files. Notification Manager used to display custom alerts in the status bar. Activity Manager that manages the lifecycle of apps. Content Providers that enable apps to access data from other apps. All framework APIs that Android system apps use. 3. Libraries and Android Runtime: Each app runs in its own process and with its own instance of the Android Runtime, which enables multiple virtual machines on low-memory devices. Android also includes a set of core runtime libraries that provide most of the functionality of the Java programming language, including some Java 8 language features that the Java API framework uses. Many core Android system components and services are built from native code that require native libraries written in C and C++. These native libraries are available to apps through the Java API framework. 4. Hardware Abstraction Layer (HAL): This layer provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework. The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or bluetooth module. 5. Linux Kernel: The foundation of the Android platform is the Linux kernel. The above layers rely on the Linux kernel for underlying functionalities such as threading and low-level memory management. Using a Linux kernel enables Android to take advantage of key security features and allows device manufacturers to develop hardware drivers for a well-known kernel.			
2 (a)	**What is MVP pattern? Demonstrate the steps to create an app that implement the MVP pattern.** Answer: **Model View Presenter** divides our application into three layers namely the Model, View and Presenter. 1. **Views.** Views are user interface elements that display data and respond to user actions. Every element of the screen is a view. The Android system provides many different kinds of views. 2. **Presenters.** Presenters connect the application's views to the model. They supply the views with data as specified by the model, and also provide the model with user input from the view. 3. **Model.** The model specifies the structure of the app's data and the code to access and manipulate the data. Some of the apps you create in the lessons work with models for accessing data. Example: • An example to explain the MVP pattern is the Login app. • Create the Views required for the app such as TextViews for Username and Password, Buttons for performing actions. • Build the model component by creating the .java file which will define the working actions of the view elements in the user interface. Such as the onClick attribute helps to perform the action of the view button. • The data passing from view to model and inferring data changes from model to view takes place through Presenter. Such as the login credentials updating and user actions such as button clicks take place via presenter.	[07]	CO1	L1,L3



(b)	**Define View groups.** Answer: Views can be grouped together inside a view group (ViewGroup), which acts as a container of views. The relationship is parent-child, in which the parent is a view group, and the child is a view or view group within the group. The views for a screen are organized in a hierarchy. At the root of this hierarchy is a ViewGroup that contains the layout of the entire screen. The view group's child screens can be other views or other view groups as shown in the following figure. 1. The root view group. 2. The first set of child views and view groups whose parent is the root.  	[03]	CO1	L1
3 (a)	**Explain different Activity states using a scenario and draw a block diagram and explain activity lifecycle callback methods.** Answer: **Scenario:** Say, a chat app is created and initialized to open at the first time in the android system, this invokes **onCreate()** method. Now if the app is started for second time it calls **onStart()** method. The app now is in visible state. If a call comes while the usage the app goes to hidden state and once the call ends the app resumes back using **onResume()** method. Now the app is in running state and if the phone charge is going to die, the notification pops up and the chat app goes to pause state calling **onPause()**. The app is closed or stopped using **onStop()** method with pressing of back button that is the app is in foreground. Again if the app is started the **onRestart()** method is called. The app is totally destroyed to free the memory resource using **ondestroy()** method.	[10]	CO1	L4



Activity lifecycle callback methods:

Activity created (onCreate() method)

When an activity is first created the system calls the onCreate() method to initialize that activity. For example, when the user taps your app icon from the Home screen to start that app, the system calls the onCreate() method for the activity in your app that you've declared to be the "launcher" or "main" activity.

Activity started (onStart() method)

After your activity is initialized with onCreate(), the system calls the onStart() method, and the activity is in the started state. The onStart() method is also called if a stopped activity returns to the foreground, such as when the user clicks the back or up buttons to navigate to the previous screen. While onCreate() is called only once when the activity is created, the onStart() method may be called many times during the lifecycle of the activity as the user navigates around your app.

Activity resumed/running (onResume() method)

Your activity is in the resumed state when it is initialized, visible on screen, and ready to use. The resumed state is often called the running state, because it is in this state that the user is actually interacting with your app. The first time the activity is started the system calls the onResume() method just after onStart(). The onResume() method may also be called multiple times, each time the app comes back from the paused state.

Activity paused (onPause() method)

The paused state can occur in several situations:

• The activity is going into the background, but has not yet been fully stopped. This is the first indication that the user is leaving your activity.

• The activity is only partially visible on the screen, because a dialog or other transparent activity is overlaid on top of it.

Activity stopped (onStop() method)

An activity is in the stopped state when it is no longer visible on the screen at all. This is usually because the user has started another activity, or returned to the home screen. The system retains the activity instance in the back stack, and if the user returns to that activity it is restarted again. Stopped activities may be killed altogether by the Android system if resources are low.

Activity destroyed (onDestroy() method)

When your activity is destroyed it is shut down completely, and the Activity instance is reclaimed by the system. This can happen in several cases:

• You call finish() in your activity to manually shut it down.

• The user navigates back to the previous activity.

• The device is in a low memory situation where the system reclaims stopped activities to free more resources.

	• A device configuration change occurs. You'll learn more about configuration changes later in this chapter. Use onDestroy() to fully clean up after your activity so that no component (such as a thread) is running after the activity is destroyed. **Activity restarted (onRestart() method)** The restarted state is a transient state that only occurs if a stopped activity is started again. In this case the onRestart() method is called in between onStop() and onStart(). If you have resources that need to be stopped or started you typically implement that behavior in onStop() or onStart() rather than onRestart().			

4 (a)	**What are Resource files? How is the resource files accessed in your java and xml files?**
	
	Answer:
	There are many more items which you use to build a good Android application. Apart from coding for the application, you take care of various other **resources** like static content that your code uses, such as bitmaps, colors, layout definitions, user interface strings, animation instructions, and more. These resources are always maintained separately in various sub-directories under **res/** directory of the project.
	
	During your application development you will need to access defined resources either in your code, or in your layout XML files. Following section explains how to access your resources in both the scenarios −
	
	Accessing Resources in Code
	
	When the Android application is compiled, a **R** class gets generated, which contains resource IDs for all the resources available in your **res/** directory. We can use R class to access that resource using sub-directory and resource name or directly resource ID.
	
	Example
	
	To access *res/drawable/myimage.png* and set an ImageView you will use following code −
	
	`ImageView imageView = (ImageView) findViewById(R.id.myimageview);`
	`imageView.setImageResource(R.drawable.myimage);`
	
	Here first line of the code make use of *R.id.myimageview* to get ImageView defined with id *myimageview* in a Layout file. Second line of code makes use of *R.drawable.myimage* to get an image with name **myimage** available in drawable sub-directory under **/res**.
	
	Example
	
	Consider next example where *res/values/strings.xml* has following definition −
	
	`<?xml version="1.0" encoding="utf-8"?>`
	`<resources>`
	`<string name="hello">Hello, World!</string>`
	`</resources>`
	
	We can set the text on a TextView object with ID msg using a resource ID as follows −
		[10]	CO1	L1,L1

```
TextView msgTextView = (TextView) findViewById(R.id.msg);
msgTextView.setText(R.string.hello);
```

Accessing Resources in XML

Consider the following resource XML *res/values/strings.xml* file that includes a color resource and a string resource −

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="opaque_red">#f00</color>
  <string name="hello">Hello!</string>
</resources>
```

Now you can use these resources in the following layout file to set the text color and text string as follows −

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:textColor="@color/opaque_red"
   android:text="@string/hello" />
```

5 (a)	**Define passing of data between activities using (a) Explicit intents and**	[05]	CO1	L1

Answer:

Starting an activity with an explicit intent:-

Explicit intents specify the receiving activity (or other component) by that activity's fully-qualified class name. Use an explicit intent to start a component in your own app (for example, to move between screens in the user interface), because you already know the package and class name of that component.

Passing data between activities with intents

The intent object you use to start an activity can include intent data (the URI of an object to act on), or intent extras, which are bits of additional data the activity might need.

In the first (sending) activity,

1. Create the Intent object.
2. Put data or extras into that intent.
3. Start the new activity with startActivity().

In the second (receiving) activity, you:

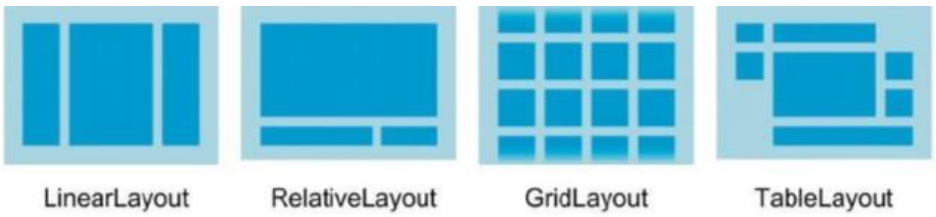
1. Get the intent object the activity was started with.

	2. Retrieve the data or extras from the Intent object.
	
	To start a specific activity from another activity, use an explicit intent and the startActivity() method. Explicit intents include the fully-qualified class name for the activity or other component in the Intent object. All the other intent fields are optional, and null by default. Ex:
	
	Intent msg= new Intent(this, ShowMessageActivity.class)
	
	The Intent constructor takes two arguments for an explicit intent. • An application context. In this example, the activity class provides the content (here, this). • The specific component to start (ShowMessageActivity.class).
	
	To add data to the intent:
	msg.setData(Uri.parse("http//www.google.com"));
	
	If the data is added then start the activity:
	startActivity(msg);
	
	Add extras:
	Use the putExtra() methods to add your key/value pairs to the intent extras.
	msg.putExtra(EXTRA_msg,"hi");
	
	Alternatively bundles can be created to add extras.
	Bundle extras=new bundle();
	extras.putString(EXTRA_msg,"hi");
	extras.putInt(EXTRA_x,50)
	
	Add it to intent:
	msg.putExtras(extras)
	
	Start the activity as usual.
	
	Retrieve the data from the intent in the started activity
	To retrieve the intent use getIntent() method.
	Intent intent=getIntent();
	
	To extract extra data:
	String message=intent.getStringExtra(MainActivity.EXTRA_msg);
	
	Extracting extras from bundle:
	Bundle extras=intent.getExtras();
	String message=extras.getString(MainActivity.EXTRA_msg);			

(b)	**Implicit Intents** Answer: Implicit intents do not specify a specific activity or other component to receive the intent. Instead you declare a general action to perform in the intent. The Android system matches your request to an activity or other component that can handle your requested action. In sending activity create a new intent object Intent msg= new Intent() Once the object is created set the action:	[05]	CO1	L1

	msg.setAction(Intent.ACTION_SEND)
	
	Before starting the activity resolve the activity
	
	if(sendIntent.resolveActivity(getPackageManager())!=null)
	{
	startActivity(chooser);
	}
	Show the app chooser
	If the android system has more than one app that performs this action then app chooser shows up and user can take up the apt app for performing the task. The createChooser() can be used to create the app chooser.
	
	Receiving implicit intents
	To retrieve the intent use getIntent() method.
	Intent intent=getIntent();
	
	To extract extra data:
	String message=intent.getStringExtra(MainActivity.EXTRA_msg);
	
	Extracting extras from bundle:
	Bundle extras=intent.getExtras();
	String message=extras.getString(MainActivity.EXTRA_msg);			

6 (a)	**What are layouts? Describe different types of layout.** Answer: Layouts ⬇ are specific types of view groups ⬇ are subclasses of ViewGroup ⬇ contain child views ⬇ can be in a row, column, grid, table, absolute • LinearLayout: A group of child views positioned and aligned horizontally or vertically. • RelativeLayout: A group of child views in which each view is positioned and aligned relative to other views within the view group. In other words, the positions of the child views can be described in relation to each other or to the parent view group. • ConstraintLayout: A group of child views using anchor points, edges, and guidelines to control how views are positioned relative to other elements in the layout. ConstraintLayout was designed to make it easy to drag and drop views in the layout editor. • TableLayout: A group of child views arranged into rows and columns. • AbsoluteLayout: A group that lets you specify exact locations (x/y coordinates) of its child views. Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning.	[10]	CO1	L1,L1

	• FrameLayout: A group of child views in a stack. FrameLayout is designed to block out an area on the screen to display one view. Child views are drawn in a stack, with the most recently added child on top. The size of the FrameLayout is the size of its largest child view. • GridLayout: A group that places its child screens in a rectangular grid that can be scrolled.  			
7 (a)	**Why is it necessary to develop apps for android?** Answer: Apps are developed for a variety of reasons: addressing business requirements, building new services, creating newbusinesses, and providing games and other types of content for users. Developers choose to develop for Android in orderto reach the majority of mobile device users. 1. Most popular platform for mobile apps As the world's most popular mobile platform, Android powers hundreds of millions of mobile devices in more than 190countries around the world. It has the largest installed base of any mobile platform and is still growing fast. Every dayanother million users power up their Android devices for the first time and start looking for apps, games, and other digital content. 2. Best experience for app users Android provides a touch-screen user interface (UI) for interacting with apps. Android's user interface is mainly based on direct manipulation, using touch gestures such as swiping, tapping and pinching to manipulate on-screen objects. In addition to the keyboard, there's a customizable virtual keyboard for text input. Android can also support game controllers and full-size physical keyboards connected by Bluetooth or USB. The Android home screen can contain several pages of app icons, which launch the associated apps, and widgets, which display live, auto-updating content such as the weather, the user's email inbox or a news ticker. Android is designed to provide immediate response to user input. Besides a fluid touch interface, the vibration capabilities of an Android device can provide haptic feedback. Internal hardware such as accelerometers, gyroscopes and proximity sensors, are used by many apps to respond to additional user actions. The Android platform, based on the Linux kernel, is designed primarily for touchscreen mobile devices such as smartphones and tablets. 3. Easy to develop apps Use the Android software development kit (SDK) to develop apps that take advantage of the Android operating system and UI. The SDK includes a comprehensive set of development tools including a debugger, software libraries of prewritten code, a device emulator, documentation, sample code, and tutorials. To develop apps using the SDK, use the Java programming language for developing the app and Extensible Markup Language (XML) files for describing data resources. By writing the code in Java and creating a single app binary, you will have an app that can run on both phone and tablet form factors. At runtime, Android applies the correct resource sets based on its screen size, density, locale, and so on. Google offers a full Java Integrated Development Environment (IDE) called Android Studio, with advanced features for developing,	[05]	CO1	L1

	debugging, and packaging Android apps. 4. Many distribution options You can distribute your Android app in many different ways: email, website or an app marketplace such as Google Play. Android users download billions of apps and games from the Google Play store each month. Google Play is a digital distribution service, operated and developed by Google, which serves as the official appstore for Android, allowing consumers to browse and download apps developed with the Android SDK and published through Google.			
(b)	Summarize the challenges of Android app development? Answer: 1. Building for a multi-screen world Android runs on billions of handheld devices around the world, and supports various form factors including wearable devices and televisions. Devices can come in different sizes and shapes that affect the screen designs for UI elements in your apps. In addition, device manufacturers may add their own UI elements, styles, and colors to differentiate their products. Each manufacturer offers different features with respect to keyboard forms, screen size, or camera buttons. The challenge for many developers is to design UI elements that can work on all devices It is also the developer's responsibility to provide an app's resources such as icons, logos, other graphics and text styles to maintain uniformity of appearance across different devices. 2. Maximizing app performance An app's performance—how fast it runs, how easily it connects to the network, and how well it manages battery and memory usage—is affected by factors such as battery life, multimedia content, and Internet access. For example, you will have to balance the background services by enabling them only when necessary; this will save battery life of the user's device. 3. Keeping your code and your users secure You need to take precautions to secure your code and the user's experience when using your app. Use tools such as ProGuard (provided in Android Studio), which detects and removes unused classes, fields, methods, and attributes, and encrypt all of your app's code and resources while packaging the app. To protect your user's critical information such as logins and passwords, you must secure the communication channel to protect data in transit (across the Internet) as well as data at rest (on the device). 4. Remaining compatible with older platform versions Consider how to add new Android platform version features to an app, while ensuring that the app can still run on devices with older platform versions. It is impractical to focus only on the most recent Android version, as not all users may have upgraded or may be able to upgrade their devices. 5. Understanding the market and the user.	[05]	CO1	L2