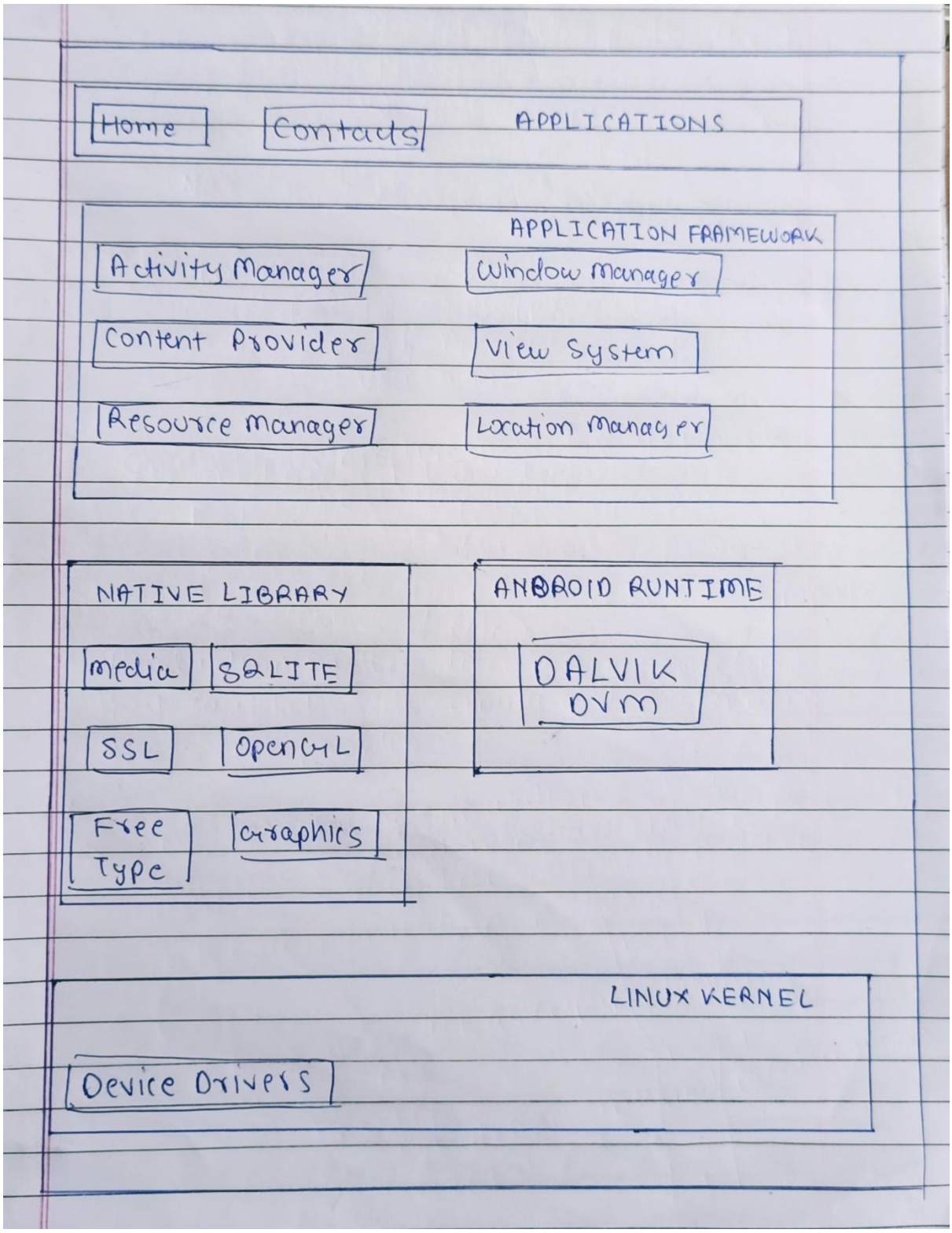
NAME: PARTH N PATEL

ID: 19DCS098

SUBJECT: MOBILE APPLICATION DEVELOPMENT

SUBJECT CODE: CS377

Q: **Write a note on android architecture.**



- **android architecture** is categorized into five parts:

    1. linux kernel
    2. native libraries (middleware),
    3. Android Runtime
    4. Application Framework
    5. Applications

LINUX KERNEL:

- It is the heart of android architecture that exists at the root of android architecture.
- **Linux kernel** is responsible for device drivers, power management, memory management, device management and resource access.
- The Linux Kernel will provide an abstraction layer between the device hardware and the other components of android architecture.

NATIVE LIBRARIES:

- On the top of linux kernel, their are **Native libraries**
- It includes various C/C++ core libraries and Java based libraries such as Media, Graphics, Surface Manager, OpenGL etc. to provide a support for android development.

ANDROID RUNTIME:

- Android Runtime environment is one of the most important part of Android.
- It contains components like core libraries and the Dalvik virtual machine(DVM).
- It provides the base for the application framework and powers our application with the help of the core libraries.
- DVM is like JVM but it is optimized for mobile devices. It consumes less memory and provides fast performance.

ANDROID FRAMEWORK:

- On the top of Native libraries and android runtime, there is android framework.
- It provides a generic abstraction for hardware access and also helps in managing the user interface with application resources.
- 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.

APPLICATIONS:

- On the top of android framework, there are applications.
- The pre-installed applications like home, contacts, camera, gallery etc and third party applications downloaded from the play store like chat applications, games etc. will be installed on this layer only.
- All applications such as home, contact, settings, games, browsers are using android framework that uses android runtime and libraries.
- Android runtime and native libraries are using linux kernal.

Q: Write a note on anatomy of an android application.

- There are four building blocks to an Android application:
    - Activity
    - Intent Receiver
    - Service
    - Content Provider
- It is not mandatory to have all of four in the application.

Activity:

- Activities are the most common of the four Android building blocks.
- An activity is usually a single screen in your application.
- Each activity is implemented as a single class that extends the Activity base class.
- Your class will display a user interface composed of Views and respond to events.
- Most applications consist of multiple screens.
- For example, a text messaging application might have one screen that shows a list of contacts to send messages to, a second screen to write the message to the chosen contact, and other screens to review old messages or change settings. Each of these screens would be implemented as an activity.
- When a new screen opens, the previous screen is paused and put onto a history stack.

- The user can navigate backward through previously opened screens in the history.
- Screens can also choose to be removed from the history stack when it would be inappropriate for them to remain.
- Android retains history stacks for each application launched from the home screen.

Intent and Intent Filters:

- Android uses a special class called Intent to move from screen to screen.
- Intent describe what an application wants done.
- The two most important parts of the intent data structure are the action and the data to act upon.
- Typical values for action are MAIN (the front door of the application), VIEW, PICK, EDIT, etc.
- The data is expressed as a Uniform Resource Indicator (URI).
- For example, to view a website in the browser, you would create an Intent with the VIEW action and the data set to a Website-URI.
- There is a related class called an IntentFilter.
- While an intent is effectively a request to do something, an intent filter is a description of what intents an activity (or intent receiver, see below) is capable of handling. Activities publish their IntentFilters in the AndroidManifest.xml file.

Intent Receiver:

- You can use an Intent Receiver when you want code in your application to execute in reaction to an external event, for example, when the phone rings, or when the data network is available, or when it's midnight.
- Intent receivers do not display a UI, although they may display Notifications to alert the user if something interesting has happened.
- Intent receivers are also registered in AndroidManifest.xml, but you can also register them from code using Context.registerReceiver().

Service:

- A Service is code that is long-lived and runs without a UI.
- A good example of this is a media player playing songs from a play list.
- In a media player application, there would probably be one or more activities that allow the user to choose songs and start playing them.
- However, the music playback itself should not be handled by an activity because the user will expect the music to keep playing even after navigating to a new screen.
- In this case, the media player activity could start a service using Context.startService() to run in the background to keep the music going.
- The system will then keep the music playback service running until it has finished.
- Note that you can connect to a service with the Context.bindService() method.

- When connected to a service, you can communicate with it through an interface exposed by the service.
- For the music service, this might allow you to pause, rewind, etc.

Content Provider:

- Applications can store their data in files, a SQLite database, preferences or any other mechanism that makes sense.
- A content provider, however, is useful if you want your application's data to be shared with other applications.
- A content provider is a class that implements a standard set of methods to let other applications store and retrieve the type of data that is handled by that content provider.

Q: **Write a note on android terminology.**

Activity :

- An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map.
- Each activity is given a window in which to draw its user interface.

Fragment :

- A Fragment represents a behaviour or a portion of user interface in an Activity.
- You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.

Intent :

- Intents are an essential part of the Android ecosystem.
- They are used to express an action to be performed.
- Intents allow you to interact with components from the same applications as well as with components contributed by other applications.
- It can be classified into implicit and explicit intents.

- **Implicit intent :** It does not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.

- **Explicit Intent :** It specifies the component to start by name. You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start.

Application :

- Base class for maintaining global application state.
- The Application class, or your subclass of the Application class, is instantiated before any other class when the process for your application/package is created.

Content Provider :

- A content provider component supplies data from one application to others on request.
- Such requests are handled by the methods of the ContentResolver class.
- A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.

XML file:

- The preeminent file is used for the structure of an android project.
- It has complete information about all the components and packages.
- It initializes the API that is further used by an application

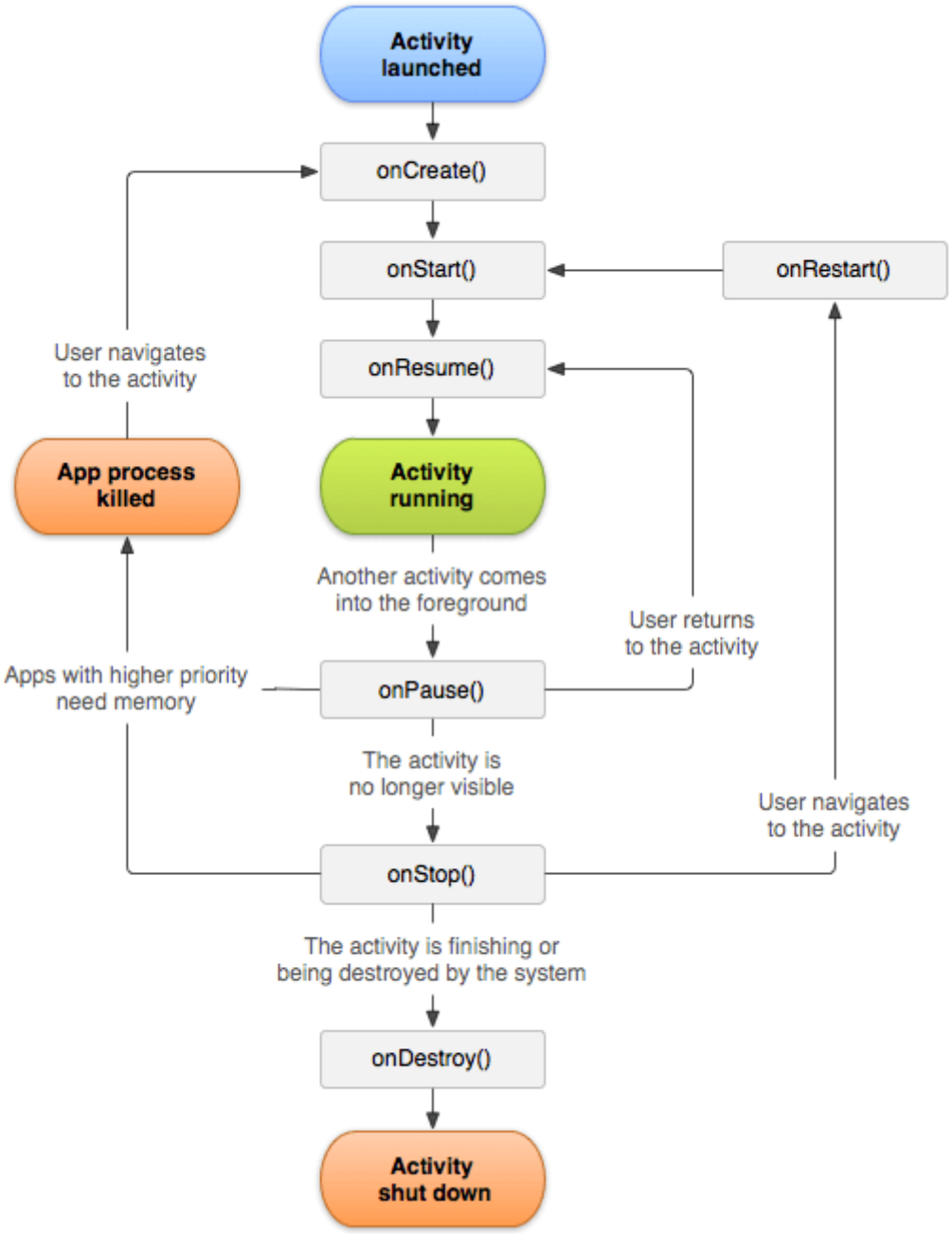
View:

- It is the component of the User Interface that occupies the rectangular area on the screen.

Layout:

- It properly aligns the views on the screen.

Q:**Write note on activity life cycle. Explain it with program.**



Lifecycle Method	Description
onCreate()	The activity is starting (but not visible to the user)
onStart()	The activity is now visible (but not ready for user interaction)
onResume()	The activity is now in the foreground and ready for user interaction
onPause()	Counterpart to onResume(). The activity is about to go into the background and has stopped interacting with the user. This can happen when another activity is launched in front of the current activity.
onStop()	Counterpart to onStart(). The activity is no longer visible to the user.
onDestroy()	Counterpart to onCreate(...). This can be triggered because finish() was called on the activity or the system needed to free up some memory.
onRestart()	Called when the activity has been stopped, before it is started again

CODE FOR EXPLAINING THE ACTIVITY CYCLE:

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

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

</androidx.constraintlayout.widget.ConstraintLayout>
```

Java Code:

```
package com.parthnpatel.lifecycle;
```

```
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.util.Log;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d("activity", "onCreate()");
    }

    @Override

    protected void onRestart(){

        super.onRestart();//call to restart after onStop

        Log.d("activity","onRestart()");

    }

    @Override

    protected void onStart() {

        super.onStart();//soon be visible
```

```
    Log.d("activity","onStarted()");
}
@Override
protected void onResume() {
    super.onResume();//visible
    Log.d("activity","onResume()");
}
@Override
protected void onPause() {
    super.onPause();//invisible
   Log.d("activity","onPause()");
}
@Override
protected void onStop() {
    super.onStop();
```

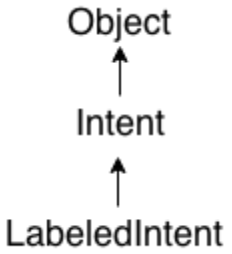
```
        Log.d("activity","onStopped()");
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d("activity","onDestroy()");
    }
}
```

OUTPUT:

```
18:39:01.231 23727-23727/com.parthnpatel.lifecycle D/activity: onCreate()
18:39:01.277 23727-23727/com.parthnpatel.lifecycle D/activity: onStarted()
18:39:01.367 23727-23727/com.parthnpatel.lifecycle D/activity: onResume()
18:39:04.736 23727-23727/com.parthnpatel.lifecycle D/activity: onPause()
18:39:05.405 23727-23727/com.parthnpatel.lifecycle D/activity: onStopped()
18:39:06.979 23727-23727/com.parthnpatel.lifecycle D/activity: onRestart()
18:39:06.992 23727-23727/com.parthnpatel.lifecycle D/activity: onStarted()
18:39:06.994 23727-23727/com.parthnpatel.lifecycle D/activity: onResume()
18:39:08.427 23727-23727/com.parthnpatel.lifecycle D/activity: onPause()
18:39:09.010 23727-23727/com.parthnpatel.lifecycle D/activity: onStopped()
18:39:09.230 23727-23727/com.parthnpatel.lifecycle D/activity: onDestroy()
```

Q:**Write a note on Intent**

- **Android Intent** is the message that is passed between components such as activities, content providers, broadcast receivers, services etc



- The LabeledIntent is the subclass of android.content.Intent class.

IMPLICIT INTENT:

- Implicit Intent doesn't specifiy the component.
- In such case, intent provides information of available components provided by the system that is to be invoked.
- It going to connect with out side application such as call, mail, phone,see any website ..etc.
- For example, you may write the following code to view the webpage.

```
Intent intent=new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.abc.com"));
startActivity(intent);
```

Explicit Intent:

- Explicit Intent specifies the component. In such case, intent provides the external class to be invoked.

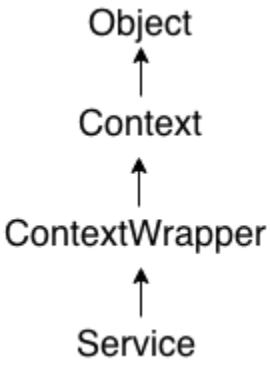
```
Intent i = new Intent(getApplicationContext(), ActivityTwo.class);

startActivity(i);
```

- It going to connect the internal world of an application such as start activity or send data between two activities.

Q: **What is service? Explain it in detail.**

- A service is a component which runs in the background without direct interaction with the user.
- As the service has no user interface, it is not bound to the lifecycle of an activity.
- Services are used for repetitive and potentially long running operations, i.e., Internet downloads, checking for new data, data processing, updating content providers and the like.
- Services run with a higher priority than inactive or invisible activities and therefore it is less likely that the Android system terminates them.
- Services can also be configured to be restarted if they get terminated by the Android system once sufficient system resources are available again.
- It is possible to assign services the same priority as foreground activities.
- In this case it is required to have a visible notification active for the related service.
- It is frequently used for services which play videos or music
- The android.app.Service is subclass of ContextWrapper class.



- There can be two forms of a service.The lifecycle of service can follow two different paths: started or bound.

    1. Started
    2. Bound

- To carry out a downloading task in the background, the **startService()** method will be called.
- Whereas to get information regarding the download progress and to pause or resume the process while the application is still in the background, the service must be bounded with a componen**t** which can perform these tasks.

Started:

- A service is started when an application component, such as an activity, starts it by calling startService().
- Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
- Two option are available to stop the execution of service:
- By calling stopService() method,
- The service can stop itself by using stopSelf() method.

Bound:

- A service is bound when an application component binds to it by calling bindService().
- A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).
- To stop the execution of this service, all the components must unbind themselves from the service by using **unbindService()** method.

Q: **What is a resource? How we access resources.**

- Resources are the additional files and static content that your code uses, such as bitmaps, layout definitions, user interface strings, animation instructions, and more.
- **Android R.java** is *an auto-generated file by aapt* (Android Asset Packaging Tool) that contains resource IDs for all the resources of res/ directory.

ACCESSING RESOURCES:

- Once you provide a resource in your application (discussed in Providing Resources), you can apply it by referencing its resource ID.
- All resource IDs are defined in your project's R class, which the aapt tool automatically generates.
- There are two ways you can access a resource:
- In code: Using an static integer from a sub-class of your R class, such as: R.string.hello
- In XML: Using a special XML syntax that also corresponds to the resource ID defined in your R class, such as: @string/hello

Q:**What is use of android manifest file? Also explain its common settings.**

- Every application must have an AndroidManifest.xml file (with precisely that name) in its root directory.
- The **AndroidManifest.xml file** *contains information of your package*, including components of the application such as activities, services, broadcast receivers, content providers etc.
- It performs some other tasks also:
    - It is **responsible to protect the application** to access any protected parts by providing the permissions.
    - It also **declares the android api** that the application is going to use.
    - It **lists the instrumentation classes**. The instrumentation classes provides profiling and other informations. These informations are removed just before the application is published etc.
- This is the required xml file for all the android application and located inside the root directory.

Package name and application ID:

- The manifest file's root element requires an attribute for your app's package name (usually matching your project directory structure—the Java namespace).

App components:

- For each app component that you create in your app, you must declare a corresponding XML element in the manifest file:

- <activity> for each subclass of Activity.
- <service> for each subclass of Service.
- <receiver> for each subclass of BroadcastReceiver.
- <provider> for each subclass of ContentProvider.
- If you subclass any of these components without declaring it in the manifest file, the system cannot start it.

Intent filters:

- App activities, services, and broadcast receivers are activated by *intents*.
- An intent is a message defined by an Intent object that describes an action to perform, including the data to be acted upon, the category of component that should perform the action, and other instructions.
- When an app issues an intent to the system, the system locates an app component that can handle the intent based on *intent filter* declarations in each app's manifest file.

Icons and labels:

- A number of manifest elements have icon and label attributes for displaying a small icon and a text label, respectively, to users for the corresponding app component.
- In every case, the icon and label that are set in a parent element become the default icon and label value for all child elements.
- For example, the icon and label that are set in the <application> element are the default icon and label for each of the app's components

Permissions:

- Android apps must request permission to access sensitive user data (such as contacts and SMS) or certain system features (such as the camera and internet access).
- Each permission is identified by a unique label.
- You must declare all permission requests with a <uses-permission> element in the manifest.
- If the permission is granted, the app is able to use the protected features. If not, its attempts to access those features fail.
- The <uses-feature> element allows you to declare hardware and software features your app needs

Q:**List out resource directory supported by android. Explain it in detail.**

- The common resources are mostly used are enlisted following
    - Animation (example of sub-directory: **res/anim**)
    - Drawable (example of sub-directory: **res/drawable**, **res/drawable-hdpi**, **res/no-drawable**)
    - Layout (example of sub-directory: **res/layout**, **res/layout-port**, **res/layout-land**)
    - Mipmap (example of sub-directory: **res/mipmap-hdpi**, **res/mipmap-mdpi**)
    - Values (example of sub-directory: **res/values**, **res/values-en**, **res/values-jp**)
    - Raw (example of sub-directory: **res/raw**)
    - Menu (example of sub-directory: **res/menu**)
    - Xml (example of sub-directory: **res/xml**)

Animation:

- A good Android application needs animation to show different transition flows. The developers use **res/anim** directory to keep the **xml** files of animation.

Drawable:

- We need many drawable files like **.png**, **.jpg**, **.gif** or **xml** which are compiled into bitmaps, state lists, shape or animation drawable.
- Usually these files are saved into **res/drawable** directory.
- There are several types of drawable directories which maintain the files to support user interface in multiple screen.

Layout:

- This is the most important part of UI resources for an Android project.
- All the xml layout files are kept inside the **res/layout** directory.
- Also, there is various cases like **res/layout-port** or **res/layout-land** to support potrait & landscape screen.

Mipmap:

- the mipmap folders are responsible to keep the **launcher icons** only.
- Of course, any other drawable files should be kept in the relevant drawable folders as before

Values:

- **res/values** is another important directory which is used to keep several important files to store string, dimension, color, style or different attributes.
- Also, you can keep the string data to support different languages,

Menu:

- The Android applications consist of action bar which have to show menu items via popup.
- Those menu items are declared into xml files which are kept into **res/menu** directory to access