UNIT: 2 INTRODUCTION TO ANDROID & ANDROID APPLICATION DESIGN

CS-31 MOBILE APPLICATION DEVELOPMENT IN ANDROID USING KOTLIN

- **The Android Platform**
- **Android SDK**
- **Building a sample Android application**
- **Anatomy of an Android applications**
- **Android terminologies**
- **Application Context, Activities, Services, Intents**
- **Receiving and Broadcasting Intents**
- **Android Manifest File and its common settings**
- **Using Intent Filter**
- **Permissions**
- **Managing Application resources in a hierarchy**
- **Working with different types of resources**

Assignment 2

1. **AVD stands for**
2. **DVM stands for**
3. **OHA stands for**
4. **SDK stands for**
5. **Usage of permission in android application**
6. **Usage of intent filter in android application**
7. **Explain Manifest file**
8. **Explain Types of Android Resources**
9. **Write note on Activity Life Cycle with Example**
10. **Write note on Android Architecture**
11. **Write note on OHA**
12. **Write steps to create an android application**
13. **Explain toast in android with suitable example**
14. **Explain various terms in android**

PREPARED BY: PROF. N.K.PANDYA (PHD*, MCA, BCA, BPP)
KAMANI SCIENCE COLLEGE, AMRELI(BCA/BSC)

The Open Handset Alliance

- The Open Handset Alliance (OHA) is a business alliance that was created for the purpose of developing open mobile device standards. The OHA has approximately 80 member companies, including HTC, Dell, Intel, Motorola, Qualcomm and Google. The OHA's main product is the Android platform - the world's most popular smartphone platform.
- OHA members are primarily mobile operators, handset manufacturers, software development firms, semiconductor companies and commercialization companies. Members share a commitment to expanding the commercial viability of open platform development.

OHA member companies back the open platform concept for a number of reasons, as follows:

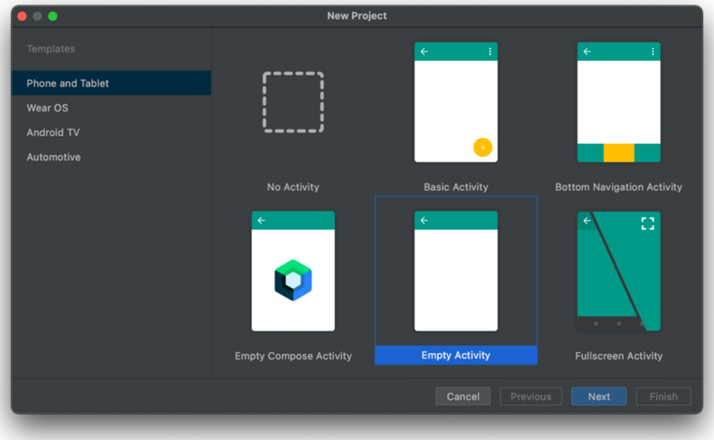
- **Lower overall handset costs**: Opens up resources, which facilitates the focus on creating innovative applications, solutions and services.
- **Developer-friendly environment**: In the open-source community, developers share notes to expedite application development.
- **Post-development**: Provides an ideal channel for application marketing and distribution.

The Android Platform

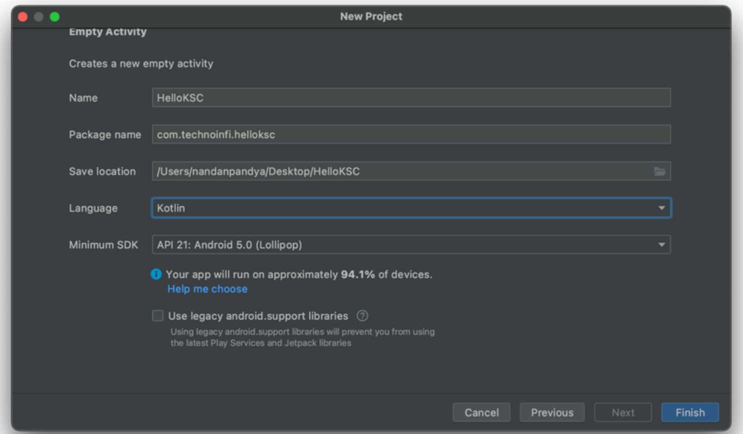
- The Android platform is a platform for mobile devices that uses a modified Linux kernel. The Android Platform was introduced by the Open Handset Alliance in November of 2007. Most applications that run on the Android platform are written in the Java programming language.
- The Android Platform was launched in 2007 by the Open Handset Alliance, an alliance of prominent companies that includes Google, HTC, Motorola, Texas Instruments and others. Although most of the applications that run on the Android Platform are written in Java, there is no Java Virtual Machine. Instead, the Java classes are first compiled into what are known as Dalvik Executables and run on the Dalvik Virtual Machine.
- Android is an open development platform. However, it is not open in the sense that everyone can contribute while a version is under development. This is all done behind closed-doors at Google. Rather, the openness of Android starts when its source code is released to the public after it is finalized. This means once it is released anyone interested can take the code and alter it as they see fit.

- To create an application for the platform, a developer requires the Android SDK, which includes tools and APIs. To shorten development time, Android developers typically integrate the SDK into graphical user IDEs (Integrated Development Environments). Beginners can also make use of the App Inventor, an application for creating Android apps that can be accessed online.

Building a sample Android application



1. Install the latest version of Android Studio.
2. In the Welcome to Android Studio window, click **Create New Project**.
3. In the Select a Project Template window, select **Empty Activity** and click Next.
4. In the Configure your project window, complete the following:
   - Enter "**Hello KSC**" in the **Name** field.
   - Enter "**com.example.helloksc**" in the **Package name** field.
   - If you'd like to place the project in a different folder, change its Save location.
   - Select **Kotlin** from the Language drop-down menu.
   - Select the **lowest** version of Android you want your app to support in the **Minimum SDK** field.


New Project

Empty Activity

Creates a new empty activity

Name: HelloKSC

Package name: com.technoinfi.helloksc

Save location: /Users/nandanpandya/Desktop/HelloKSC

Language: Kotlin

Minimum SDK: API 21: Android 5.0 (Lollipop)

Your app will run on approximately **94.1%** of devices.
Help me choose

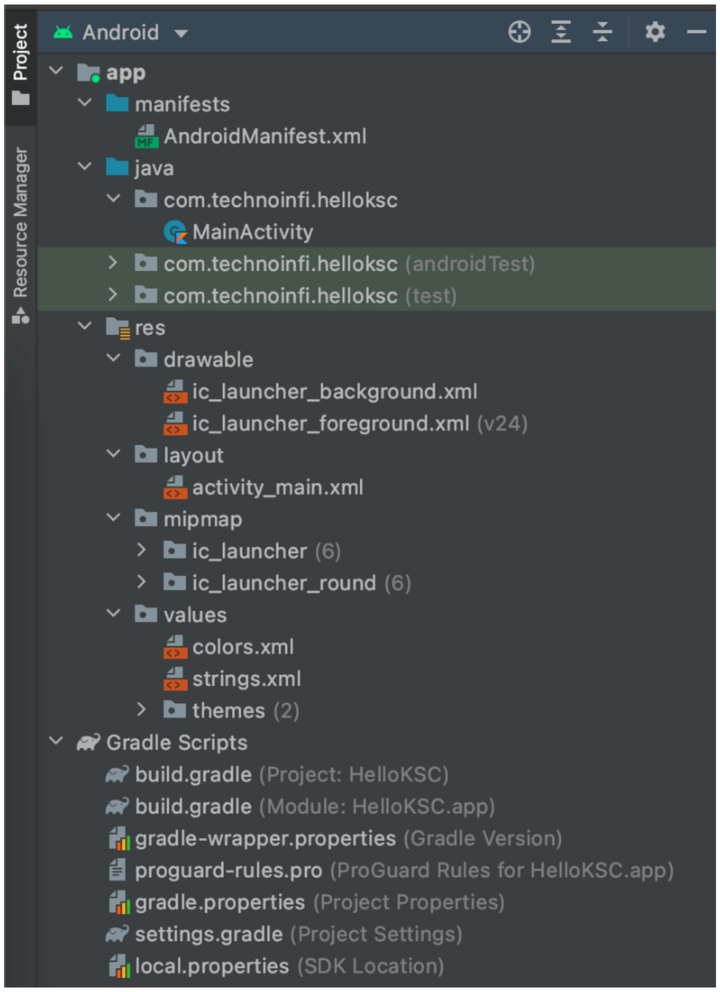
☐ Use legacy android.support libraries
Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries

5. Click **Finish**.

Note: After some processing time, the Android Studio main window appears.(5-10 mins or more depending on computer)

Anatomy of an Android applications/Project overview/directory structure

- Android Studio is the official IDE (Integrated Development Environment) developed by JetBrains community which is freely provided by Google for android app development.
- After completing the setup of Android Architecture we can create android application in the studio



```
app
  manifests
    AndroidManifest.xml
  java
    com.technoinfi.helloksc
      MainActivity
    com.technoinfi.helloksc (androidTest)
    com.technoinfi.helloksc (test)
  res
    drawable
      ic_launcher_background.xml
      ic_launcher_foreground.xml (v24)
    layout
      activity_main.xml
    mipmap
      ic_launcher (6)
      ic_launcher_round (6)
    values
      colors.xml
      strings.xml
      themes (2)
Gradle Scripts
  build.gradle (Project: HelloKSC)
  build.gradle (Module: HelloKSC.app)
  gradle-wrapper.properties (Gradle Version)
  proguard-rules.pro (ProGuard Rules for HelloKSC.app)
  gradle.properties (Project Properties)
  settings.gradle (Project Settings)
  local.properties (SDK Location)
```

- The android project contains different type of app modules, source code files and resource files. We will explore all the folders and files in android app.

1. Manifests Folder
2. Java Folder
3. res (Resources) Folder
4. Drawable Folder
5. Layout Folder
6. Mipmap Folder
7. Values Folder
8. Gradle Scripts

- **Manifests folder** contains AndroidManifest.xml for our creating the android application. This file contains information about our application such as android version, metadata, states package for Kotlin file and other application components. It acts as an intermediator between android OS and our application.
- **Java folder** contains all the java and Kotlin source code (.java) files which we create during the app development, including other Test files. If we create any new project using Kotlin, by default the class file MainActivity.kt file will create automatically under the package name “com.technoinfi.helloksc”.

- **Resource folder** is the most important folder because it contains all the non-code sources like images, XML layouts, UI strings for our android application.
  - **res/drawable folder**: It contains the different type of images used for the development of the application. We need to add all the images in drawable folder for the application development.
  - **res/layout folder**: Layout folder contains all XML layout files which we used to define the user Interface of our application. It contains the activity_main.xml file.
  - **res/midmap folder**: This folder contains launcher.xml files to define icons which are used to show on the home screen. It contains different density type of icons depends upon the size of the device such as hdpi, mdpi, xhdpi.
  - **res/values folder**: Values folder contains a number of XML files like strings, dimens, colors and styles definitions. One of the most important file is strings.xml file which contains the resources.
  - **colors.xml**: colors.xml file contains color resources of the Android application. Different color values are identified by a unique name that can be used in the Android application program.
  - **strings.xml**: The strings.xml file contains string resources of the Android application. The different string value is identified by a unique name that can be used in the Android application program. This file also stores string array by using XML language.
- **Gradle Scripts folder** Gradle means automated build system and it contains number of files which are used to define a build configuration which can be apply to all modules in our application. In build.gradle (Project) there are buildscripts and in build.gradle (Module) plugins

and implementations are used to build configurations that can be applied to all our application modules.

Android terminologies

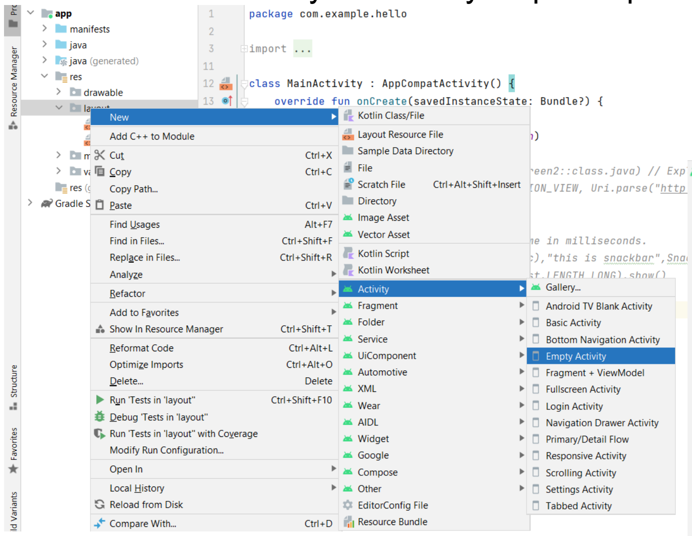
While diving into development we must know the basic things briefly, then only we can build things better. We need to setting strong foundation, then we can easily build blocks on top of it.

- **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.(In viewpager every tab shows an individual fragment)
- **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.
- **Emulator**: The emulator is the virtual device smartphone provided with an android studio. You can run your created application on the emulator and test its UI and function according to the needs.
- **Services**: It is used to run the process even in the background. There is no defined UI for service. Any component can start the service and end the services. You can easily switch between the applications even if the services are running the background.
- **Android Application Package (APK)**:The archive (package) file format used by the Android operating system to distribute applications.
- **ADB**, otherwise known as Android Debug Bridge, is a developer tool that lets you send commands to an Android device you've connected to your computer. It's a fairly advanced tool, and you run it through the command line on your PC or Mac, but if you ever want to install, say, a developer preview release of Android on your phone, you'll need to delve into ADB.

Receiving And Broadcasting Intents/ Indents In Android/Open New Activity

- Android Intent is a messaging object used to request another app component to perform an action. Intent facilitates users to communicate with app component through several ways such as starting an activity, starting a service, delivering a broadcast receiver, etc.
- Android intents are mainly used to:
  - Start the service
  - Launch an activity
  - Display a web page
  - Display a list of contacts

  - Broadcast a message
  - Dial a phone call etc.
- **Explicit Intent**: This intent satisfies the request within the application component. It takes the fully qualified class name of activities or services that we want to start.
- **Syntax**:
  - myintent = Intent(applicationContext, SecondActivity::class.java)
  - startActivity(myintent)
- **Explanation**:
  - **Intent** is a class and we created a object called **myintent** using **Intent constructor** that takes 2 parameter 1. **Application Context** and 2. **Class of activity to open**.
  - **startActivity** is a function which will open our activity this usually takes one parameter of type **intent object.**
- **Implicit Intent**: This intent does not specify the component name. It invokes the component of another app to handle it.
- **Syntax**:
  - intent= Intent(Intent.ACTION_VIEW, Uri.parse("https://www.kscpac.org/"))
  - startActivity(intent)
- **Explanation**:
  - Above code will open Kamani Science College website, here we are passing one constant from Intent class **ACTION_VIEW** instead of application Context.
- **Steps to create new activity**
  1. Click on app > res > layout > Right Click on layout. After that Select New > Activity and choose your Activity as per requirement.

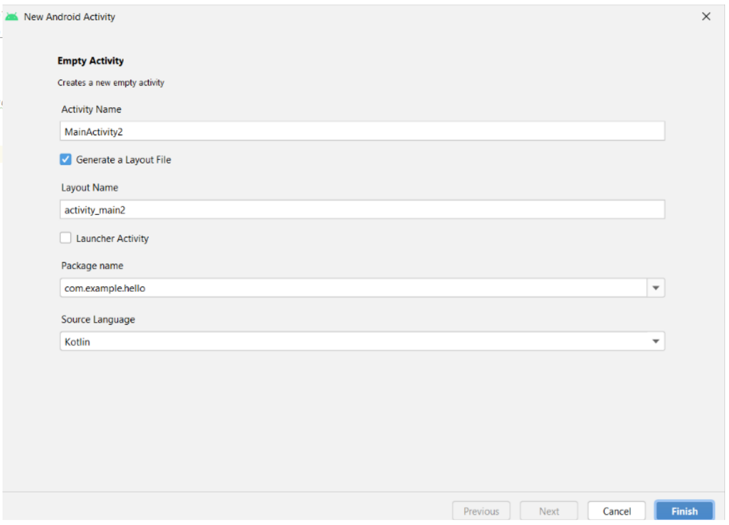


```
package com.example.hello

import ...

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
```

  2. After that Customize the Activity in Android Studio. Enter the “Activity Name” and “Package name” in the Text box and Click on Finish button.



New Android Activity

Empty Activity

Creates a new empty activity

Activity Name: MainActivity2

[✓] Generate a Layout File

Layout Name: activity_main2

[ ] Launcher Activity

Package name: com.example.hello

Source Language: Kotlin

  3. After that your new Activity in Layout will be created. Your XML Code is in Text and your Design Output is in Design.

Android Manifest File and its common settings

- 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 information's. These information's 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.
- **<manifest>:** manifest is the root element of the AndroidManifest.xml file. It has package attribute that describes the package name of the activity class.
- **<application>:** application is the subelement of the manifest. It includes the namespace declaration. This element contains several subelements that declares the application component such as activity etc.The commonly used attributes are of this element are icon, label, theme etc.
    - **android:icon** represents the icon for all the android application components.
    - **android:label** works as the default label for all the application components.
    - **android:theme** represents a common theme for all the android activities.
- **<activity>:** activity is the subelement of application and represents an activity that must be defined in the AndroidManifest.xml file. It has many attributes such as label, name, theme, launchMode etc.
    - **android:label** represents a label i.e. displayed on the screen.
    - **android:name** represents a name for the activity class. It is required attribute.
- **<intent-filter>:** intent-filter is the sub-element of activity that describes the type of intent to which activity, service or broadcast receiver can respond to.
- **<action>:** It adds an action for the intent-filter. The intent-filter must have at least one action element.
- **<category>:** It adds a category name to an intent-filter.

Using Intent Filter

- In android, Intent Filter is an expression in the app's manifest file (ActivityMainfest.xml) and it is used to specify the type of intents that the component would like to receive. In case if we create Intent Filter for an activity, there is a possibility for other apps to start our activity by sending a certain type of intent otherwise the activity can be started only by an explicit intent.
- Generally, the Intent Filters (<intent-filter>) whatever we define in the manifest file can be nested in the corresponding app components and we can specify the type of intents to accept using these three elements.

<action>: It defines the name of an intended action to be accepted and it must be a literal string value of an action, not the class constant.
<category>: It defines the name of an intent category to be accepted and it must be the literal string value of an action, not the class constant.
<data>: It defines the type of data to be accepted and by using one or more attributes we can specify various aspects of the data URI (scheme, host, port, path) and MIME type.

Example:
```
<activity android:name=".MainActivity">
   <!-- This activity is the main entry, should appear in app launcher -->
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
      <data android:mimeType="text/plain"/>
   </intent-filter>
</activity>
```

```
<activity android:name=".ResultActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

Explanation:

- If you observe above code snippet the activity “MainActivity” will act as an entry point for our app because we defined an activity using **MAIN** action and **LAUNCHER** category attributes in intent filters (<intent-filter>).
- **MAIN** : It indicates the app’s main entry point that means it starts the activity which defines with the MAIN action when the user initially launches the app with a launcher icon.
- **LAUNCHER** : It indicates that this activity icon should be placed on the home screen list of apps. In case if the <activity> element doesn’t specify an icon with icon, then the system uses the icon from the <application> element.
- These two (MAIN, LAUNCHER) elements must be paired together in order for the activity to appear in the app launcher.
- The second activity “ResultActivity” is intended to help us to share the text. The user might enter this activity by navigating from MainActivity and **they can also enter directly from another app using Implicit Intent which is matching one of the two activity filters**.

Permissions

- Whenever we are developing an Android application, we need various components of Android devices like camera, GPS, etc. So, to use these features of our Android device, we need to take permission from the user to use something present on their phone. You can’t directly use any of those features. Also, there are various protection levels in permission i.e. if the protection level of permission is very low then you need not ask the user to use that permission. You can directly use that. But for dangerous permissions, you need to explicitly take permission from the user.
- Normally, if we want to add take some permission from the user, we write the below code in our AndroidManifest.xml file
    - <uses-permission android:name="android.permission.INTERNET" />: grants permission to use the internet.
- Using permission is not a simple task, you have to decide if you need to explicitly ask the user for permission or you can directly take permission. This decision is made based on the protection level of permission. Following are the three protection levels of permissions in Android:
- **Normal Permissions**: If there is a very little or no risk of the user privacy then the permission comes under the Normal Permission category. For example, if you want to get the data and time, then these things involve no user privacy and in this case, you need not ask the user to use date or time. You can directly use this facility by adding permission in the AndroidManifest.xml file. At the installation time, the system will automatically grant permission to your app.
- Following are the permission that comes under Normal Permissions:
    - ACCESS_LOCATION_EXTRA_COMMANDS

  - **ACCESS_NETWORK_STATE**
  - CHANGE_NETWORK_STATE
  - ACCESS_WIFI_STATE
  - CHANGE_WIFI_STATE
  - CHANGE_WIFI_MULTICAST_STATE
  - BLUETOOTH
  - BLUETOOTH_ADMIN
  - **INTERNET**
  - SET_ALARM
  - SET_WALLPAPER
  - VIBRATE
  - WAKE_LOCK
- **Signature Permissions:** The android system grants these permissions at the installation time but there is one condition. The app that is asking for some permission must be signed with the same signature as that of the app that defines the required permission.
- Following are some of the Signature permissions:
  - BIND_ACCESSIBILITY_SERVICE
  - BIND_AUTOFILL_SERVICE
  - BIND_CARRIER_SERVICE
  - BIND_DEVICE_ADMIN
  - BIND_INPUT_METHOD
  - BIND_NFC_SERVICE
  - BIND_TV_INPUT
  - BIND_WALLPAPER
  - READ_VOICEMAIL
  - WRITE_SETTINGS
  - WRITE_VOICEMAIL
- **Runtime permissions, also known as dangerous permissions** include that permission that involve user data in some or the other way. For example, if you want to read contacts from the phone or you want to access the file storage of the phone then these permissions come under the Dangerous permission as they include user's privacy. To use Dangerous permissions, you have to explicitly ask for permission before using that by showing some alert dialog or any other dialog. If the user denies the permission then your application can't use that particular permission.
- Following are some of the Dangerous permissions:
  - READ_CALENDAR
  - WRITE_CALENDAR
  - CAMERA
  - READ_CALL_LOG
  - WRITE_CALL_LOG
  - READ_CONTACTS
  - WRITE_CONTACTS
  - GET_ACCOUNTS
  - ACCESS_FINE_LOCATION
  - ACCESS_COARSE_LOCATION
  - SEND_SMS
  - RECEIVE_SMS
- **Special Permissions**: These are those permissions that are neither Normal permissions nor Dangerous permissions. Most of the applications should not use these permissions because they are very sensitive and you need user authorization before using these permissions. To

use this permission, you must declare it in the AndroidManifest.xml file and then send an intent to request for user authorization.

- Some of the Special permissions are:
  - WRITE_SETTINGS
  - SYSTEM_ALERT_WINDOW

Managing Application resources in a hierarchy

- Resources are the additional files and static content that your code uses, such as bitmaps, layout definitions, user interface strings, animation instructions, and more.
- You should always externalize app resources such as images and strings from your code, so that you can maintain them independently. You should also provide alternative resources for specific device configurations, by grouping them in specially-named resource directories. At runtime, Android uses the appropriate resource based on the current configuration. For example, you might want to provide a different UI layout depending on the screen size or different strings depending on the language setting.
- Once you externalize your app resources, you can access them using resource IDs that are generated in your project's R class. This document shows you how to group your resources in your Android project and provide alternative resources for specific device configurations, and then access them from your app code or other XML files.

animator/	XML files that define property animations.
anim/	XML files that define tween animations. (Property animations can also be saved in this directory, but the animator/ directory is preferred for property animations to distinguish between the two types.)
color/	XML files that define a state list of colors. See Color State List Resource
drawable/	Bitmap files (.png, .9.png, .jpg, .gif) etc.
mipmap/	Drawable files for different launcher icon densities. For more information on managing launcher icons with mipmap/ folders, see Put app icons in mipmap directories.
layout/	XML files that define a user interface layout.
menu/	XML files that define app menus, such as an Options Menu, Context Menu, or Sub Menu. See Menu Resource.
raw/	Arbitrary files to save in their raw form. To open these resources with a raw InputStream, call Resources.openRawResource() with the resource ID, which is R.raw.filename. However, if you need access to original file names and file hierarchy, you might consider saving some resources in the assets/ directory (instead of res/raw/).
values/	XML files that contain simple values, such as strings, integers, and colors. Whereas XML resource files in other res/ subdirectories define a single resource based on the XML filename, files in the values/ directory describe multiple resources. For a file in this directory, each child of the <resources> element defines a single resource. For example, a <string> element creates an R.string resource and a <color> element creates an R.color resource. Because each resource is defined with its own XML element, you can name the file whatever you want and place different resource types in one file. However, for clarity, you might want to place unique resource types in different files. For example, here are some filename conventions for resources you can create in this directory:

	• arrays.xml for resource arrays (typed arrays). • colors.xml for color values • dimens.xml for dimension values. • strings.xml for string values. • styles.xml for styles. See String Resources, Style Resource, and More Resource Types.
xml/	Arbitrary XML files that can be read at runtime by calling Resources.getXML(). Various XML configuration files must be saved here, such as a searchable configuration.
font/	Font files with extensions such as .ttf, .otf, or .ttc, or XML files that include a <font-family> element. For more information about fonts as resources, go to Fonts in XML.

Working with different types of resources

Accessing your app resources

- Once you provide a resource in your application, 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.
- When your application is compiled, aapt generates the R class, which contains resource IDs for all the resources in your res/ directory. For each type of resource, there is an R subclass (for example, R.drawable for all drawable resources), and for each resource of that type, there is a static integer (for example, R.drawable.icon). This integer is the resource ID that you can use to retrieve your resource.
- Although the R class is where resource IDs are specified, you should never need to look there to discover a resource ID. A resource ID is always composed of:
  - **The resource type**: Each resource is grouped into a "type," such as string, drawable, and layout. For more about the different types, see Resource Types.
  - **The resource name**, which is either: the filename, excluding the extension; or the value in the XML android:name attribute, if the resource is a simple value (such as a string).
- **Accessing resources in code:** You can use a resource in code by passing the resource ID as a method parameter. For example, you can set an ImageView to use the res/drawable/myimage.png resource using setImageResource():
  - val imageView = findViewById(R.id.myimageview) as ImageView
  - imageView.setImageResource(R.drawable.myimage)
- **Accessing resources from XML**
- You can define values for some XML attributes and elements using a reference to an existing resource. You will often do this when creating layout files, to supply strings and images for your widgets.
- For example, if you add a Button to your layout, you should use a string resource for the button text:

```
<Button
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/submit" />
```

Toast and SnackBar

- **Toast:** Android Toast is used to display a sort time notification to the user without affecting the user interaction with UI. The message displayed using Toast class displays quickly, and it disappears after some time. The message in the Toast can be of type text, image or both.

- Toast.makeText(applicationContext," Hello KSC Students ",Toast.LENGTH_SHORT).show()
- val toast = Toast.makeText(applicationContext, "Hello KSC Students",Toast.LENGTH_LONG)
- toast.show()

- **Android Snackbar** is a material design component introduced with API 22.2.0. The functionality would resemble Android Toast, but unlike Toast, Snackbar could be dismissed by user or an action listener could be setup to respond to user actions. Can be used only with view.
- val snack = Snackbar.make(R.id.txt_view,"This is a simple Snackbar",Snackbar.LENGTH_LONG)
- snack.show()

Programs to perform:

1. Simple application and all resources
2. Create new activity/resource
3. Handling intent
4. Handling Intent Filter
5. Toast/Snackbar
6. Assessing Recourses using Code