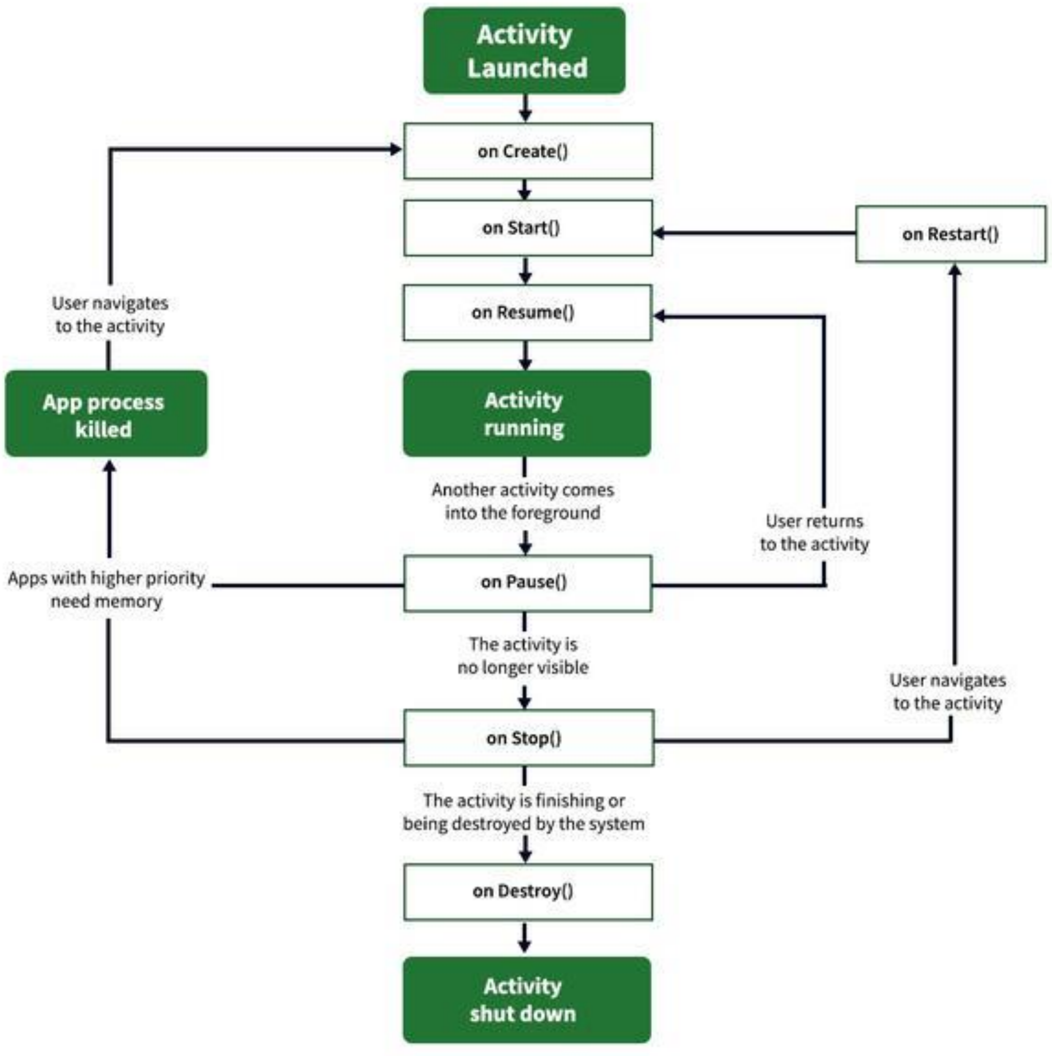
Stream: B.SC (computer science)

Subject : Fundamentals of Mobile Programming

Unit - 2 | Android Application Design Essentials

Android Life Cycle :

- In Android, an activity is referred to as one screen in an application. It is very similar to a single window of any desktop application.
- An Android app consists of one or more screensor activities. Each activity goes through various stages or a lifecycle and is managed by activity stacks.
- So when a new activity starts, the previous one always remains below it. There are four stages of an activity.
- Android Activity Lifecycle is controlled by 7 methods of android.app.Activity class.


Activity Launched
on Create()
on Start()
on Restart()
on Resume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
on Pause()
The activity is no longer visible
User navigates to the activity
on Stop()
The activity is finishing or being destroyed by the system
on Destroy()
Activity shut down

Activity Lifecycle in Android

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 Example :**

- It provides the details about the invocation of life cycle methods of activity. In this example, we are displaying the content on the logcat.

MainActivity.java

```
package com.example.myapplication1;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.Toast;


public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toast toast = Toast.makeText(getApplicationContext(), "onCreate Called",
Toast.LENGTH_LONG);
        toast.show();
    }
    protected void onStart() {
        super.onStart();
        Toast toast = Toast.makeText(getApplicationContext(), "onStart Called",
Toast.LENGTH_LONG);
        toast.show();
    }
    @Override
    protected void onRestart() {
        super.onRestart();
        Toast toast = Toast.makeText(getApplicationContext(), "onRestart Called",
Toast.LENGTH_LONG);
        toast.show();
    }

    protected void onPause() {
        super.onPause();
        Toast toast = Toast.makeText(getApplicationContext(), "onPause Called",
Toast.LENGTH_LONG);
        toast.show();
    }
    protected void onResume() {
        super.onResume();
        Toast toast = Toast.makeText(getApplicationContext(), "onResume Called",
Toast.LENGTH_LONG);
        toast.show();
    }
    protected void onStop() {
        super.onStop();
        Toast toast = Toast.makeText(getApplicationContext(), "onStop Called",
Toast.LENGTH_LONG);
        toast.show();
    }
    protected void onDestroy() {
        super.onDestroy();
        Toast toast = Toast.makeText(getApplicationContext(), "onDestroy Called",
Toast.LENGTH_LONG);
        toast.show();
    }
}
```

Logcat Window in Android Studio :

- **LogCat Window** is the place where various messages can be printed when an application runs. Suppose, you are running your application and the program crashes, unfortunately.
- Then, Logcat Window is going to help you to **debug** the output by collecting and viewing all the messages that your emulator throws. So, this is a very useful component for the app development because this Logcat dumps a lot of system messages and these messages are actually thrown by the emulator.

v(String, String)	verbose
d(String, String)	debug
i(String, String)	information
w(String, String)	warning
e(String, String)	error

The different LogCat methods are:
Log.v(); // Verbose
Log.d(); // Debug
Log.i(); // Info
Log.w(); // Warning
Log.e(); // Error

Example : used log cart

```
package com.example.myapplication1;

import androidx.appcompat.app.AppCompatActivity;

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


public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("msg","onCreate() event");
    }

    /** Called when the activity is about to become visible. */
    @Override
    protected void onStart() {
        super.onStart();
        Log.d("msg","onStart() event");
    }

    /** Called when the activity has become visible. */
    @Override
    protected void onResume() {
        super.onResume();
        Log.d("msg","onResume() event");
    }

    /** Called when another activity is taking focus. */
    @Override
    protected void onPause() {
        super.onPause();
        Log.d("msg","onPause() event");
    }

    /** Called when the activity is no longer visible. */
    @Override
    protected void onStop() {
        super.onStop();
        Log.d("msg","onStop() event");
    }

    /** Called just before the activity is destroyed. */
    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d("msg","onDestroy() event");
    }

}
```

Output :

it will display Emulator window and you should see following log messages in **LogCat** window in Android studio –

```
08-23 10:32:07.682 4480-4480/com.example.helloworld D/Android :: The onCreate() event
08-23 10:32:07.683 4480-4480/com.example.helloworld D/Android :: The onStart() event
08-23 10:32:07.685 4480-4480/com.example.helloworld D/Android :: The onResume() event
```

All these methods contain two parameters in them. The first and second both are the string.

Android terminologies :

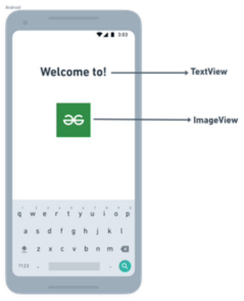
Activity:

- Activity is a key component of an Android application that represents a single screen with a user interface (UI). It serves as an entry point for interacting with the user. Each activity is typically tied to a user interaction, such as displaying a screen or handling user inputs.
- Activities in Android follow a specific life cycle, which is a series of states an activity goes through during its existence, such as being created, started, paused, resumed, or destroyed.

Layout :

- Layouts in Android is used to define the user interface that holds the UI controls or widgets that will appear on the screen of an android application or activity screen. Generally, every application is a combination of View and ViewGroup.
- So, each activity contains multiple user interface components and those components are the instances of the View and ViewGroup.
- All the elements in a layout are built using a hierarchy of View and ViewGroup objects.

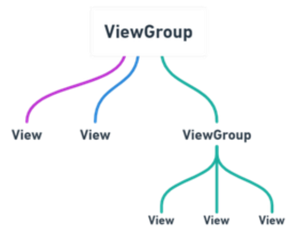
View :



- A **View** is defined as the user interface which is used to create interactive UI components such as TextView, ImageView, EditText, RadioButton, etc., and is responsible for event handling and drawing.
- They are Generally Called **Widgets**.

ViewGroup :

- The ViewGroup class is a subclass of the View class. And also it will act as a base class for layouts and layouts parameters.
- Following are the commonly used ViewGroup subclasses used in android applications.
- They are Generally Called **Layout.**



```
<?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:id="@+id/text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="This is a TextView" />

</LinearLayout>
```

- **Android Layout Types :**

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

Sr.No	Layout & Description
1	Linear Layout LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	Relative Layout RelativeLayout is a view group that displays child views in relative positions.
3	Table Layout TableLayout is a view that groups views into rows and columns.
4	Frame Layout The FrameLayout is a placeholder on screen that you can use to display a single view.
5	List View ListView is a view group that displays a list of scrollable items.
6	Grid View GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

Layout Attributes

- Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and their are other attributes which are specific to that layout.
- Following are common attributes and will be applied to all the layouts:
- You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height –

  - **android:layout_width=wrap_content** tells your view to size itself to the dimensions required by its content.
  - **android:layout_width=fill_parent** tells your view to become as big as its parent view.

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the view.
2	**android:layout_width** This is the width of the layout.
3	**android:layout_height** This is the height of the layout
4	**android:layout_marginTop** This is the extra space on the top side of the layout.
5	**android:layout_marginBottom** This is the extra space on the bottom side of the layout.
6	**android:layout_marginLeft** This is the extra space on the left side of the layout.
7	**android:layout_marginRight** This is the extra space on the right side of the layout.
8	**android:layout_gravity** This specifies how child Views are positioned.
9	**android:layout_weight** This specifies how much of the extra space in the layout should be allocated to the View.
10	**android:paddingLeft** This is the left padding filled for the layout.
11	**android:paddingRight** This is the right padding filled for the layout.

12	**android:paddingTop** This is the top padding filled for the layout.
13	**android:paddingBottom** This is the bottom padding filled for the layout.

- Here width and height are the dimension of the layout/view which can be specified in terms of **dp (Density-independent Pixels), sp (Scale-independent Pixels), pt (Points which is 1/72 of an inch), px(Pixels), mm (Millimeters)** and finally **in (inches).**

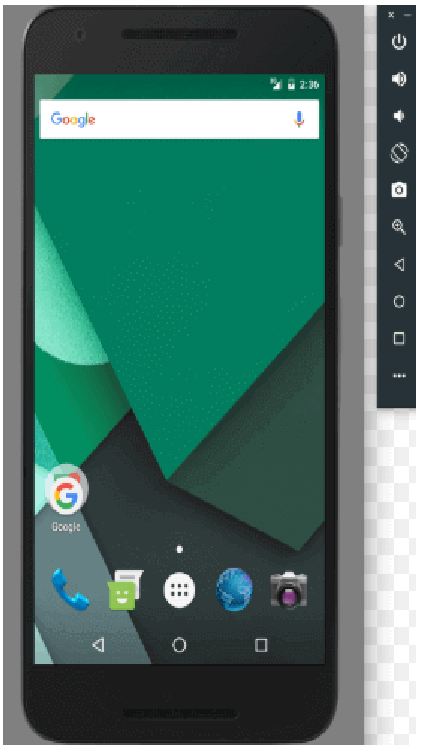
  View Identification

  - A view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is −

    android:id="@+id/my_button"

    Following is a brief description of @ and + signs −
    - The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
    - The plus-symbol (+) means that this is a new resource name that must be created and added to our resources.

Android Emulator



- The **Android emulator** is an **Android Virtual Device (AVD),** which represents a specific Android device. We can use the Android emulator as a target device to execute and test our Android application on our PC.
- The Android emulator provides almost all the functionality of a real device. We can get incoming phone calls and text messages. It also gives the location of the device and simulates different network speeds.
- Android emulator simulates rotation and other hardware sensors. It accesses the Google Play store, and much more.
- Testing Android applications on emulators is sometimes faster and easier than doing it on a real device. For example, we can transfer data faster to the emulator than to a real device connected through USB.
- The Android emulator comes with predefined configurations for several Android phones, Wear OS, tablet, Android TV devices.

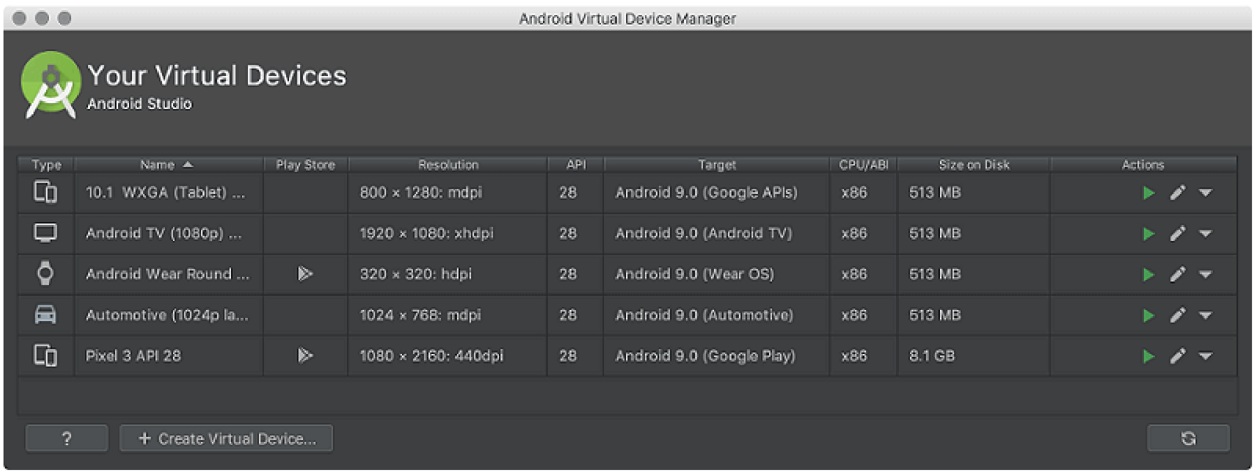
Install the emulator

- The Android emulator is installed while installing the Android Studio. However some components of the emulator may or may not be installed while installing Android Studio. To install the emulator component, select the **Android Emulator** component in the **SDK Tools** tab of the **SDK Manager.**

Run an Android app on the Emulator

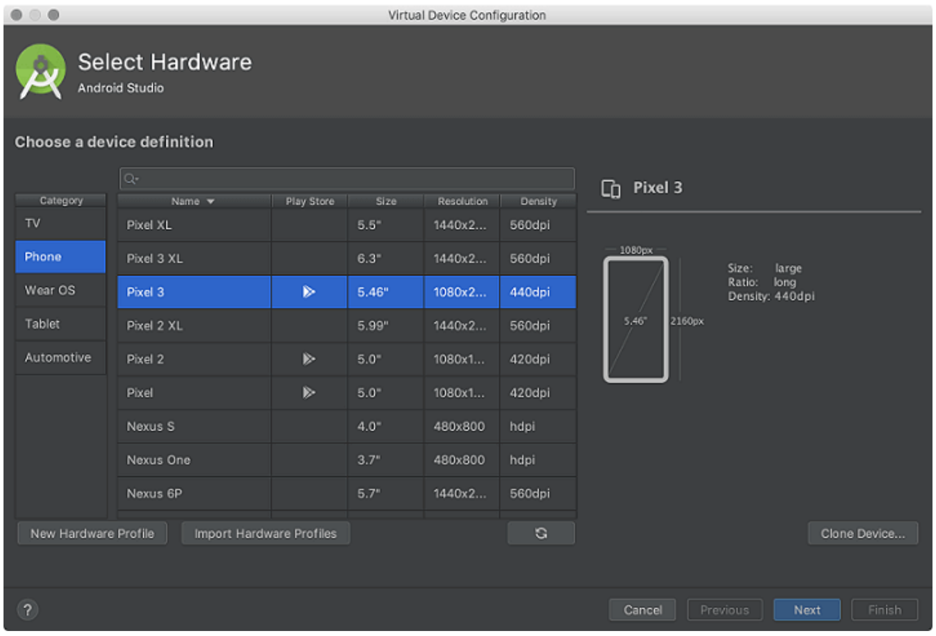
- We can run an Android app from the Android Studio project, or we can run an app which is installed on the Android Emulator as we run any app on a device.
- To start the Android Emulator and run an application in our project:

1. In **Android Studio,** we need to create an Android Virtual Device (AVD) that the emulator can use to install and run your app. To create a new AVD:-
1.1 Open the AVD Manager by clicking **Tools > AVD Manager.**



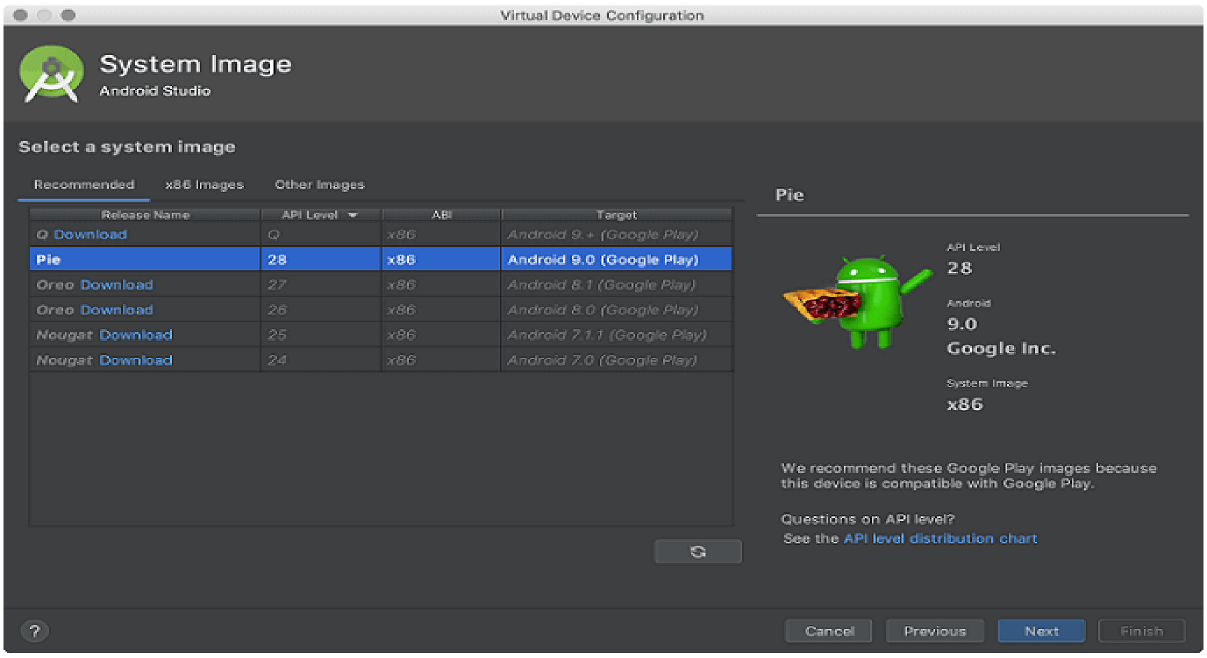
Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	10.1 WXGA (Tablet) ...		800 × 1280: mdpi	28	Android 9.0 (Google APIs)	x86	513 MB	
	Android TV (1080p) ...		1920 × 1080: xhdpi	28	Android 9.0 (Android TV)	x86	513 MB	
	Android Wear Round ...	▶	320 × 320: hdpi	28	Android 9.0 (Wear OS)	x86	513 MB	
	Automotive (1024p la...		1024 × 768: mdpi	28	Android 9.0 (Automotive)	x86	513 MB	
	Pixel 3 API 28	▶	1080 × 2160: 440dpi	28	Android 9.0 (Google Play)	x86	8.1 GB	

1.2 Click on Create **Virtual** Device, at the bottom of the AVD Manager dialog. Then **Select Hardware** page appears.



Name	Play Store	Size	Resolution	Density
Pixel XL		5.5"	1440x2...	560dpi
Pixel 3 XL		6.3"	1440x2...	560dpi
Pixel 3	▶	5.46"	1080x2...	440dpi
Pixel 2 XL		5.99"	1440x2...	560dpi
Pixel 2	▶	5.0"	1080x1...	420dpi
Pixel	▶	5.0"	1080x1...	420dpi
Nexus S		4.0"	480x800	hdpi
Nexus One		3.7"	480x800	hdpi
Nexus 6P		5.7"	1440x2...	560dpi

1.3 Select a hardware profile and then click **Next.** If we don't see the hardware profile we want, then we can create or import a hardware profile. The **System Image** page appears.


Virtual Device Configuration

System Image
Android Studio

Select a system image

Recommended | x86 Images | Other Images

Release Name	API Level	ABI	Target
Q Download	Q	x86	Android 9.+ (Google Play)
Pie	28	x86	Android 9.0 (Google Play)
Oreo Download	27	x86	Android 8.1 (Google Play)
Oreo Download	26	x86	Android 8.0 (Google Play)
Nougat Download	25	x86	Android 7.1.1 (Google Play)
Nougat Download	24	x86	Android 7.0 (Google Play)

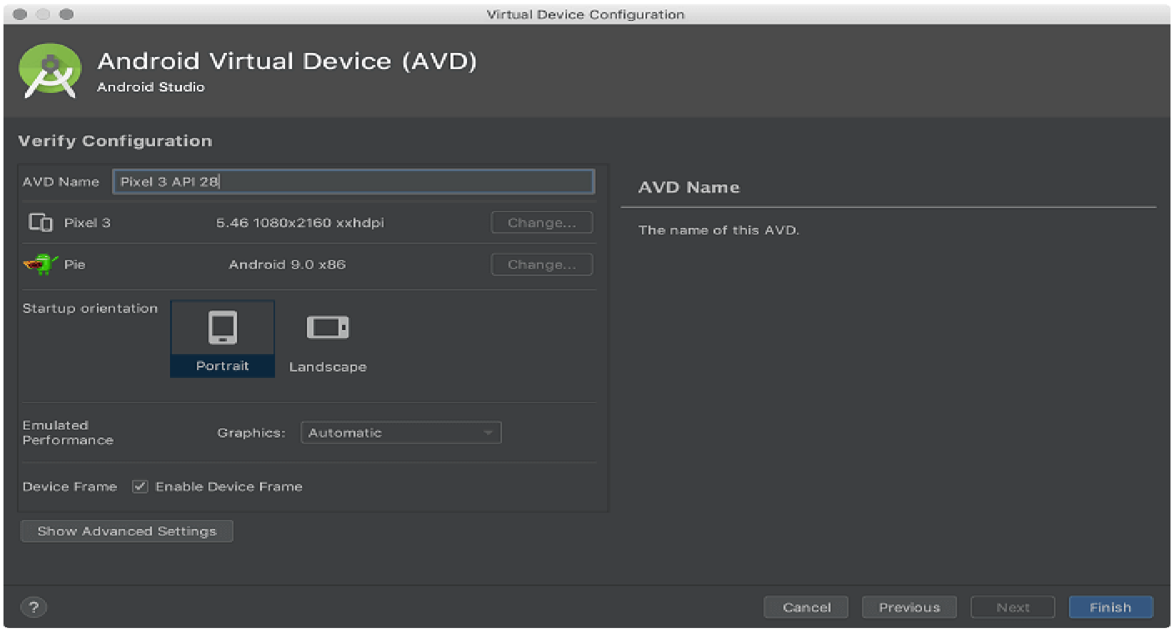
Pie

API Level
28
Android
9.0
Google Inc.
System Image
x86

We recommend these Google Play images because this device is compatible with Google Play.

Questions on API level?
See the API level distribution chart

1.4 Select the system image for the particular API level and click **Next.** This leads to open a **Verify Configuration** page.


Virtual Device Configuration

Android Virtual Device (AVD)
Android Studio

Verify Configuration

AVD Name: Pixel 3 API 28

Pixel 3 — 5.46 1080x2160 xxhdpi — Change...

Pie — Android 9.0 x86 — Change...

Startup orientation: Portrait | Landscape

Emulated Performance — Graphics: Automatic

Device Frame — ☑ Enable Device Frame

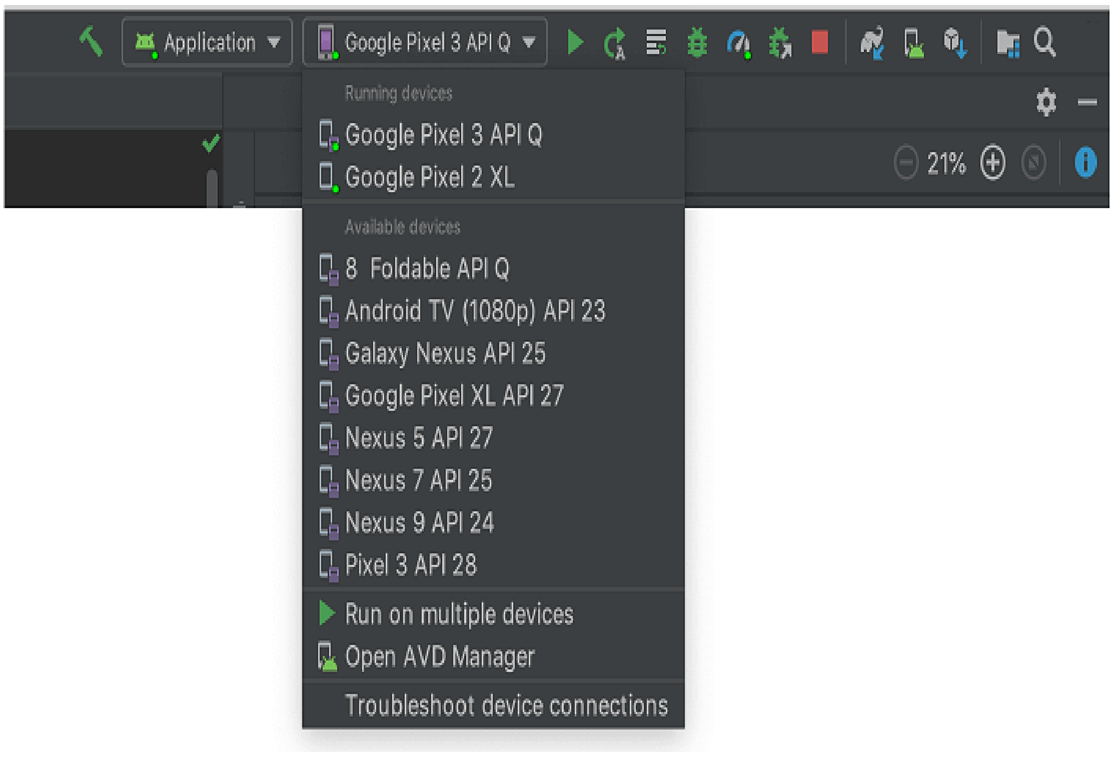
Show Advanced Settings

AVD Name

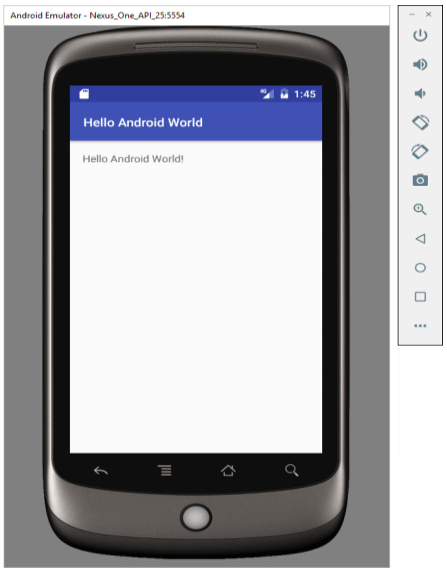
The name of this AVD.

1.5 Change AVD properties if needed, and then click **Finish.**

2. In the toolbar, choose the AVD, which we want to run our app from the target device from the drop-down menu.



3. Click **Run.**



❖ Launch the Emulator without first running an app

To start the emulator:

1. Open the AVD Manager.
2. Double-click an AVD, or click **Run**

- While the emulator is running, we can run the Android Studio project and select the emulator as the target device. We can also drag an APKs file to install on an emulator, and then run them.

Gradle :

- Gradle is a build system (open source) that is used to automate building, testing, deployment, etc. “build.gradle” are script where one can automate the tasks.
- For example, the simple task of copying some files from one directory to another can be performed by the Gradle build script before the actual build process happens.
- **Why is Gradle Needed?**
  - Every Android project needs a Gradle for generating an apk from the *.java* and *.xml* files in the project. Simply put, a Gradle takes all the source files (java and XML) and applies appropriate tools, e.g., converts the java files into dex files and compresses all of them into a single file known as apk that is actually used.
  - There are two types of build.gradle scripts.

    1. Top-level build.gradle
    2. Module-level build.gradle

Top-level build.gradle:

- It is located in the root project directory and its main function is to define the build configurations that will be applied to all the modules in the project. It is implemented as:

Example :

```
buildscript {
    repositories {
        google()     // Google's Maven repository for Android libraries for Play Services
        mavenCentral()  // A widely used repository for Java and Kotlin libraries.
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.4.3'
    }
}
allprojects {
    repositories {
        google()
        mavenCentral()
    }
}
task clean(type: Delete) {
    delete rootProject.buildDir
}
```

buildscript: This block is used to configure the repositories and dependencies for Gradle.
dependencies: This block in buildscript is used to configure dependencies that the Gradle needs to build during the project.

This line adds the plugins as a classpath dependency for gradle 3.0.1.

1. **allprojects:** This is the block where one can configure the third-party plugins or libraries. For freshly created projects android studio includes mavenCentral() and Google's maven repository by default.

2. **task clean(type:Delete):** This block is used to delete the directory every time the project is run. This way the projects keep clean when someone modifies some config files like, during settings.gradle which requires a complete clean.

Module-level build.gradle:

Located in the *project/module* directory of the project this Gradle script is where all the dependencies are defined and where the SDK versions are declared. This script has many functions in the project which include additional build types and override settings in the main/app manifest or *top-level build.gradle* file. It is implemented as:

```
Example :

apply plugin : 'com.android.application'
android
{
   compileSdkVersion 30
   buildToolsVersion "30.0.3"
   {
      applicationId "example.mehakmeet.geeksforgeeks"
      minSdkVersion 19
      targetSdkVersion 30
      versionCode 1
      versionName "1.0"
   }

}

dependencies
{
 }
```

The Module-level build.gradle supports various build configurations like:

1. **android:** This block is used for configuring the specific android build options.

    - *compileSdkVersion* – This is used to define the API level of the app and the app can use the features of this and lower level.

2. **defaultConfig:**

    - *applicationId*– This is used for identifying unique id for publishing of the app.

- *minSdkVersion*– This defines the minimum API level required to run the application.

- *targetSdkVersion*– This defines the API level used to test the app.

- *versionCode*– This defines the version code of the app. Every time an update needs to be of the app, the version code needs to be increased by 1 or more.

- *versionName*– This defines the version name for the app. this could be increased by much while creating an update.

3. **dependencies:** This specifies the dependencies that are needed to build the project.

Android Manifest File and its common settings :

- Every project in Android includes a Manifest XML file, which is AndroidManifest.xml, located in the root directory of its project hierarchy.
- The manifest file is an important part of our app because it defines the structure and metadata of our application, its components, and its requirements.
- This file includes nodes for each of the Activities, Services, Content Providers, and Broadcast Receivers that make the application, and using Intent Filters and Permissions determines how they coordinate with each other and other applications.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.geeksforgeeks"
    android:versionCode="1"
    android:versionName="1.0"
    android:installLocation="preferExternal">

      <uses-sdk
        android:minSdkVersion="18"
        android:targetSdkVersion="27" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApplication"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>

</manifest>
```

➢ **Manifest :**

- The main component of the AndroidManifest.xml file is known as manifest. Additionally, the packaging field describes the activity class’s package name. It must contain an <application> element with the xmlns:android and package attribute specified.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.geeksforgeeks">

    <!-- manifest nodes -->

    <application>

    </application>

</manifest>
```

➢ **uses-sdk :**

- It is used to define a minimum and maximum SDK version by means of an API Level integer that must be available on a device so that our application functions properly, and the target SDK for which it has been designed using a combination of minSdkVersion, maxSdkVersion, and targetSdkVersion attributes, respectively.
- It is contained within the <manifest> element.

```
<uses-sdk
    android:minSdkVersion="18"
    android:targetSdkVersion="27" />
```

➢ **uses-permission :**

- It outlines a system permission that must be granted by the user for the app to function properly and is contained within the <manifest> element.
- When an application is installed (on Android 5.1 and lower devices or Android 6.0 and higher), the user must grant the application .

```
<uses-permission
    android:name="android.permission.CAMERA"
    android:maxSdkVersion="18" />
```

- **Application :**

- A manifest can contain only one application node. It uses attributes to specify the metadata for your application (including its title, icon, and theme).
- During development, we should include a debuggable attribute set to true to enable debugging, then be sure to disable it for your release builds.
- The application node also acts as a container for the Activity, Service, Content Provider, and Broadcast Receiver nodes that specify the application components. The name of our custom application class can be specified using the android:name attribute.

```
<application
    android:name=".GeeksForGeeks"
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@drawable/gfgIcon"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@android:style/Theme.Light"
    android:debuggable="true"
    tools:targetApi="31">

    <!-- application nodes -->

</application>
```

Application node :

- **uses-library :**

- It defines a shared library against which the application must be linked.
- This element instructs the system to add the library's code to the package's class loader. It is contained within the <application> element.

```
<uses-library
    android:name="android.test.runner"
    android:required="true" />
```

- **Activity :**

- The Activity sub-element of an application refers to an activity that needs to be specified in the AndroidManifest.xml file.
- It has various characteristics, like label, name, theme, launchMode, and others. In the manifest file, all elements must be represented by <activity>.
- Any activity that is not declared there won't run and won't be visible to the system. It is contained within the <application> element.

```
<activity
    android:name=".MainActivity"
    android:exported="true">
</activity>
```

- **<u>intent-filter :</u>**

- It is the sub-element of activity that specifies the type of intent to which the activity, service, or broadcast receiver can send a response.
- It allows the component to receive intents of a certain type while filtering out those that are not useful for the component.
- The intent filter must contain at least one <action> element.

```
<intent-filter>
   <action android:name="android.intent.action.MAIN" />

   <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

- **Action :**
- It adds an action for the intent-filter. It is contained within the <intent-filter> element.

```
<action android:name="android.intent.action.MAIN" />
```

- **Category :**
- It adds a category name to an intent-filter. It is contained within the <intent-filter> element.

```
<category android:name="android.intent.category.LAUNCHER" />
```

- **uses-configuration :**

- The uses-configuration components are used to specify the combination of input mechanisms that are supported by our application.
- It is useful for games that require particular input controls.

```
<uses-configuration
   android:reqTouchScreen=”finger”
   android:reqNavigation=”trackball”
   android:reqHardKeyboard=”true”
   android:reqKeyboardType=”qwerty”/>

<uses-configuration
android:reqTouchScreen="finger"
android:reqNavigation="trackball"
android:reqHardKeyboard="true"
android:reqKeyboardType="twelvekey" />
```

- **uses-features :**

- It is used to specify which hardware features your application requires.
- This will prevent our application from being installed on a device that does not include a required piece of hardware such as **NFC (Near Field Communication)** hardware, as follows:

```
<uses-feature android:name=”android.hardware.nfc” android:required="true"/>
```

Resource Management in Android :

- 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.

```
MyProject/
  app/
    manifest/
      AndroidManifest.xml
  java/
    MyActivity.java
    res/
      drawable/
        icon.png
      layout/
        activity_main.xml
        info.xml
      values/
        strings.xml
```

Sr.No.	Directory & Resource Type
1	**color/** XML files that define a state list of colors. They are saved in res/color/ and accessed from the **R.color** class.
2	**drawable/** Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawable. They are saved in res/drawable/ and accessed from the **R.drawable** class.
3	**layout/** XML files that define a user interface layout. They are saved in res/layout/ and accessed from the **R.layout** class.
4	**menu/** XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in res/menu/ and accessed from the **R.menu** class.

5	**values/** XML files that contain simple values, such as strings, integers, and colors. For example, here are some filename conventions for resources you can create in this directory − • arrays.xml for resource arrays, and accessed from the **R.array** class. • integers.xml for resource integers, and accessed from the **R.integer** class. • bools.xml for resource boolean, and accessed from the **R.bool** class. • colors.xml for color values, and accessed from the **R.color** class. • dimens.xml for dimension values, and accessed from the **R.dimen** class. • strings.xml for string values, and accessed from the **R.string** class. • styles.xml for styles, and accessed from the **R.style** class.
6	**xml/** Arbitrary XML files that can be read at runtime by calling *Resources.getXML()*. You can save various configuration files here which will be used at run time.

- **Accessing Resources in Code :**

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);
```

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>
```

Now you can set the text on a TextView object with ID msg using a resource ID as follows −

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

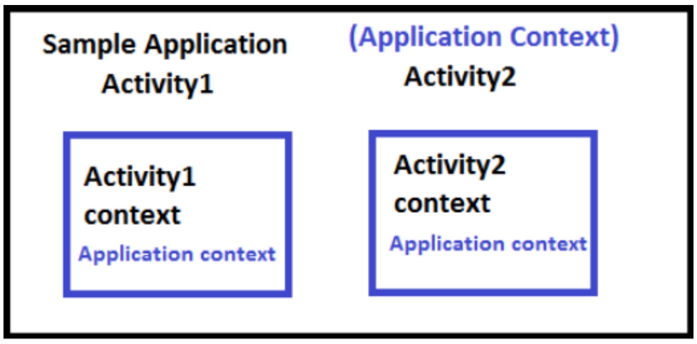
Application Context :

- A Context gives us information about the current state of an application. It allows us to interact with Android Components.
- It allows us to access files and other resources such as pictures, Activities, Fragments and Services.
- The Context Class provides access to several resources and services that are needed to build an Android application.

- **Types of Context in Android :**

  - There are mainly two types of Context that are available in Android.

    1. **Application Context and**
    2. **Activity Context**



1) Application Context :

- This Context is tied to the **Lifecycle of an Application**. Mainly it is an instance that is a singleton and can be accessed via **getApplicationContext()**.
- Some use cases of Application Context are:
  - If it is necessary to create a singleton object
  - During the necessity of a library in an activity

➢ **getApplicationContext():**

- It is used to return the Context which is linked to the Application which holds all activities running inside it.
- When we call a method or a constructor, we often have to pass a Context and often we use **"this"** to pass the activity Context or **"getApplicationContext"** to pass the application Context.
- This method is generally used for the application level and can be used to refer to all the activities.
- For example, if we want to access a variable throughout the android app, one has to use it via **getApplicationContext()**.

Example :

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
     Toast.makeText(getApplicationContext(),"Welcome to My App!",
Toast.LENGTH_SHORT).show();
   }
}
```

2) Activity Context :

- It is the activity Context meaning each and every screen got an activity.
- For example, EnquiryActivity refers to EnquiryActivity only and AddActivity refers to AddActivity only. It is tied to the life cycle of activity.
- It is used for the current Context. The method of invoking the Activity Context is **getContext()**.

- Some use cases of Activity Context are:
    - The user is creating an object whose lifecycle is attached to an activity.
    - Whenever inside an activity for UI related kind of operations like toast, dialogue, etc.,

➢ **getContext():**

- It returns the Context which is linked to the Activity from which it is called.
- This is useful when we want to call the Context from only the current running activity.
- It provides access to the current context, such as the Activity, Fragment, or View in which it is called.

➢ you can **use this** in place of a Context in many cases, such as **inflating layouts, starting new activities, or accessing resources.**

Example :

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Example 1: Show a Toast using Activity context

       Toast.makeText(this, "Welcome to MainActivity!", Toast.LENGTH_SHORT).show();

// Using `this` to start another Activity

        Button button = findViewById(R.id.startButton);

        button.setOnClickListener(view -> {

            Intent intent = new Intent(this, SecondActivity.class);

            startActivity(intent);

        });
    }
}
```

➢ **Difference between Activity Context and Application Context :**

Activity Context	**Application Context**
Activity Context has a shorter life cycle than Application Context. It is created and destroyed with the Activity.	Application Context is created when the application starts and remains till the application gets terminated.
Activity Context has a shorter life cycle and can be destroyed with the activity.	Application Context has a longer life cycle and remains in the memory throughout the lifetime of an application.
Activity Context is limited to current Activity such as Fragments and Views.	Application Context is available to the entire application including all activities and their components.

Using Intent Filter, Permissions :

- The intent is a messaging object which tells what kind of action to be performed.
- The intent's most significant use is the launching of the activity.
- Intent facilitates the communication between the components.

- **Intent Filter :**

    - Implicit intent uses the intent filter to serve the user request.
    - The intent filter specifies the types of intents that an activity, service, or broadcast receiver can respond.
    - Intent filters are declared in the Android manifest file.
    - Intent filter must contain <action>

- Most of the intent filter are describe by its
    1. **<action>,**
    2. **<category>** and
    3. **<data>**

1. <action>

```
<action android:name="string" />
```

- Adds an action to an intent filter.
- An <intent-filter> element must contain one or more <action> elements.
- If there are no <action> elements in an intent filter, the filter doesn't accept any Intent objects.

- **Examples of common action:**
    - **ACTION_VIEW:** Use this action in intent with startActivity() when you have some information that activity can show to the user like showing an image in a gallery app or an address to view in a map app
    - **ACTION_SEND:** You should use this in intent with startActivity() when you have some data that the user can share through another app, such as an email app or social sharing app.

2. <category>

```
<category android:name="string" />
```

- Adds a category name to an intent filter. A string containing additional information about the kind of component that should handle the intent.
- **Example of common categories:**
    - **CATEGORY_BROWSABLE:** The target activity allows itself to be started by a web browser to display data referenced by a link.

3. <data>

```
<data android:scheme="string"
    android:host="string"
    android:port="string"
    android:path="string"
    android:pathPattern="string"
    android:pathPrefix="string"
    android:mimeType="string" />
```

- Adds a data specification to an intent filter. The specification can be just a data type, just a URI, or both a data type and a URI.

Example of Activity , Intent & Intent Filter :

activity_main.xml file

```
<?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">
        <Button
                android:id="@+id/sendButton"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="SEND"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintHorizontal_bias="0.476"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintVertical_bias="0.153" />

        <Button
                android:id="@+id/buttonView"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="View"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintHorizontal_bias="0.498"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent"
                app:layout_constraintVertical_bias="0.826" />

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

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.example.menuapplication">

        <application
                android:allowBackup="true"
                android:icon="@mipmap/ic_launcher"
                android:label="@string/app_name"
                android:roundIcon="@mipmap/ic_launcher_round"
                android:supportsRtl="true"
                android:theme="@style/Theme.MenuApplication">
                <activity android:name=".MainActivity">
                        <intent-filter>
                        <action android:name="android.intent.action.MAIN" />
                        <category  android:name="android.intent.category.LAUNCHER" />
                        </intent-filter>

                        <!--SEND INTENT FILTER-->
                        <intent-filter>
                        <action android:name="android.intent.action.SEND"/>
                        <category android:name="android.intent.category.DEFAULT"/>
                        <data android:mimeType="text/plain"/>
                        </intent-filter>
```

}

```
                              <!--VIEW INTENT FILTER-->
                              <intent-filter>
                              <action android:name="android.intent.action.VIEW"/>
                              <category android:name="android.intent.category.DEFAULT"/>
                              <category android:name="android.intent.category.BROWSABLE"/>
                              <data android:scheme="http"/>
                              </intent-filter>

                    </activity>

          </application>

</manifest>
```

MainActivity.java file

```
package com.example.myapplication;

import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity

public class MainActivity extends AppCompatActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);


                    // send button on click listener
                    sendButton.setOnClickListener {
                              var intent = Intent(Intent.ACTION_SEND) // intent
                              intent.type = "text/plain"
                              intent.putExtra(Intent.EXTRA_EMAIL, "niranjantest@gmail.com")
                              intent.putExtra(Intent.EXTRA_SUBJECT, "This is a dummy message")
                              intent.putExtra(Intent.EXTRA_TEXT, "Dummy test message")
                              startActivity(intent)
                    }

                    // View on click listener
                    buttonView.setOnClickListener {
                              var intent = Intent(Intent.ACTION_VIEW)
                              startActivity(intent)
                    }
          }
}
```