
# S.Y.B.Sc. (C. S.)

## SEMESTER - IV (CBCS)

# ANDROID DEVELOPER FUNDAMENTALS

## SUBJECT CODE: USCS407

**Prof. (Dr.) D. T. Shirke**
Offg. Vice Chancellor
University of Mumbai, Mumbai

**Prin. Dr. Ajay Bhamare**
Offg. Pro Vice-Chancellor,
University of Mumbai

**Prof. Prakash Mahanwar**
Director,
IDOL, University of Mumbai

**Programme Co-ordinator** : **Shri Mandar Bhanushe**
Head, Faculty of Science and Technology IDOL,
Univeristy of Mumbai – 400098

**Course Co-ordinator** : **Ms. Mitali Vijay Shewale**
Doctoral Researcher,
Veermata Jijabai Technological Institute
HR Mahajani road, Matunga, Mumbai

**Editor** : **Palash Ingle**
Research Assistant,
Sejong University, Korea.

**Course Writers** : **Ansari Mohd Shahid**
Assistant Professor,
Maharashtra College of Arts, Science and
Commerce Nagpada, Mumbai 400008

: **Shraddha Bhushan Sable**
Assistant Professor,
S. K. College of Science & Commerce
Nerul, Navi Mumbai 400706

: **Ifrah Kampoo**
Assistant Professor,
D.G. Ruparel College of Arts, Science and
Comerce, Matunga West, Mumbai 400016

June 2023, Print - 1

**Published by : Director,**
Institute of Distance and Open Learning,
University of Mumbai,
Vidyanagari,Mumbai - 400 098.

**DTP composed and Printed by: Mumbai University Press**

# CONTENTS

**S.Y.B.Sc. (C. S.)**

**SEMESTER - IV (CBCS)**

# ANDROID DEVELOPER FUNDAMENTALS

# SYLLABUS

| **Course**: **USCS407** | **TOPICS (Credits : 02 Lectures/Week: 03)** **Android Developer Fundamentals** | |
|---|---|---|
| **Objectives:** To provide the comprehensive insight into developing applications running on smart mobile devices and demonstrate programming skills for managing task on mobile. To provide systematic approach for studying definition, methods and its applications for Mobile-App development. | | |
| **Expected Learning Outcomes:** 1) Understand the requirements of Mobile programming environment. 2) Learn about basic methods, tools and techniques for developing Apps 3) Explore and practice App development on Android Platform 4) Develop working prototypes of working systems for various uses in daily lives. | | |
| **Unit I** | What is Android? Obtaining the required tools, creating first android app, understanding the components of screen, adapting display orientation, action bar, Activities and Intents, Activity Lifecycle and Saving State, Basic Views: TextView, Button, ImageButton, EditText, CheckBox, ToggleButton, RadioButton, and RadioGroup Views, ProgressBar View, AutoCompleteTextView, TimePicker View, DatePicker View, ListView View, Spinner View | **15L** |
| **Unit II** | User Input Controls, Menus, Screen Navigation, RecyclerView, Drawables, Themes and Styles, Material design, Providing resources for adaptive layouts, AsyncTask and AsyncTaskLoader, Connecting to the Internet, Broadcast receivers, Services, Notifications, Alarm managers, Transferring data efficiently | **15L** |
| **Unit III** | Data - saving, retrieving, and loading: Overview to storing data, Shared preferences, SQLite primer, store data using SQLite database, ContentProviders, loaders to load and display data, Permissions, performance and security, Firebase and AdMob, Publish your app | **15L** |

**Textbook**(s):

1) "Beginning Android 4 Application Development", Wei-Meng Lee, March 2012, WROX.

**Additional Reference(s):**

1) https://developers.google.com/training/courses/android-fundamentals
2) https://www.gitbook.com/book/google-developer-training/android-developer-fundamentals-course-practicals/details

# 1

# INTRODUCTION TO ANDROID STRUCTURE

**Unit Structure :**

# 1.0 LEARNING OBJECTIVES

- By studying this unit students are be able to install the android studio and able to create the android application in Android Studio.
- Also Students are able to run the applications using Android Virtual Device(AVD) .
- At the end of this unit students can make simple mobile applications.

# 1.1 INTRODUCTION

- Android is a mobile operating system developed by Google. It is used by several smart phones and tablets.
- The Android operating system (OS) is based on the Linux kernel.
- Unlike Apple's IOS, Android is open source, meaning developers can modify and customize the OS for each phone.

**An install of Android Studio includes:**

- Android SDK the latest version of the Android SDK
- Android SDK tools and platform tools for debugging and testing your apps A system image for the Android emulator lets you create and test your apps on different virtual devices
- Downloading and Installing Android Studio
- Android Studio is available from Android's developer site at

**developer.android.com/ sdk/**

- If you do not already have it installed, you will need to install the Java Development Kit (JDK8), which you can download from

**www.oracle.com**

# 1.2 WHY ANDROID

**Zero/negligible development cost**

- The development tools like Android SDK, JDK, and Eclipse IDE etc. are free to download for the android mobile application development. Also Google charge a small fee $25, to distribute your mobile app on the Android Market.

**Open Source**

- The Android OS is an open-source platform based on the Linux kernel and multiple open-source libraries. In this way developers are free to contribute or extend the platform as necessary for building mobile apps which run on Android devices.

**Multi-Platform Support**

- In market, there are a wide range of hardware devices powered by the Android OS, including many different phones and tablet. Even development of android mobile apps can occur on Windows, Mac OS or Linux.

**Multi-Carrier Support**

- Worldwide a large number of telecom carriers like Airtel, Vodafone, Idea Cellular, AT&T Mobility, BSNL etc. are supporting Android-powered phones.

**Open Distribution Model**

- Android Market place (Google Play store) has very few restrictions on the content or functionality of an android app. So the developer can distribute theirs app through the Google Play store and as well other distribution channels like Amazon's app store.

## 1.2.1 FEATURES OF ANDROID

There are numerous features of android. Some of them are listed below:

| Feature | Description |
|---|---|
| **Connectivity** | Android supports multiple connectivity technologies including GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX |
| **Storage** | SQLite, a lightweight relational database, is used for data storage purposes |
| **Media support** | Android supports various type of audio/video/still media formats like: H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, BMP and WebP |
| **Web browser** | The web browser available in Android is based on the open-source Blink (previously WebKit) layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3 |
| **Messaging** | SMS and MMS are available forms of messaging, it also include threaded text messaging and Android Cloud To Device Messaging (C2DM) and now support the enhanced version of C2DM, Android Google Cloud Messaging (GCM) is also a part of Android Push Messaging services |
| **Multi-tasking** | Multitasking of applications, with unique handling of memory allocation, is available, using this user can jump from one task to another and at the same time various application can run simultaneously |

| Feature | Description |
|---|---|
| **Resizable widgets** | Widgets are re-sizable, so users can expand them to show more content or shrink them to save space |
| **Multi-touch** | Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero |
| **Wi-Fi** | A technology that lets apps discover and pair directly, over a high-bandwidth peer-to-peer connection. |
| **Screen capture** | Android supports capturing a screenshot by pressing the power and home-screen buttons at the same time. This features supports after Android 4.0 |
| **Multi-Language** | Android supports multiple languages, also supports single direction and bi-directional text |

**The Android Studio**

- Android Studio provides a unified environment where you can build apps for Android phones, tablets, Android Wear, Android TV, and Android Auto. Structured code modules allow you to divide your project into units of functionality that you can independently build, test, and debug

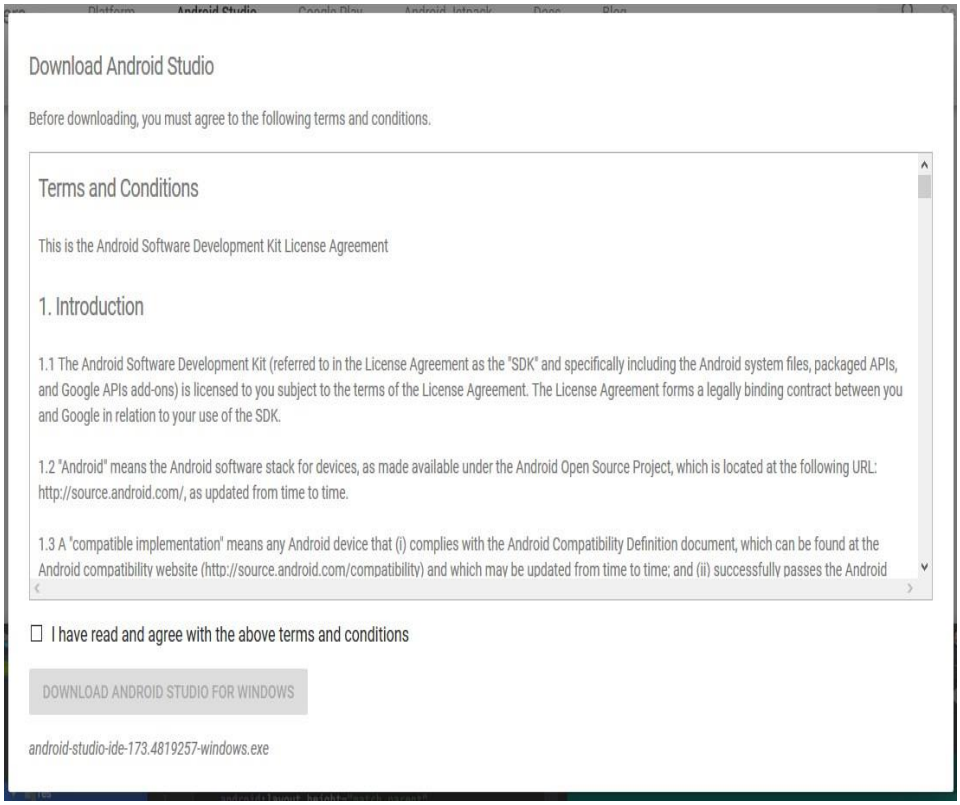

## 1.4 STEPS TO INSTALL ANDROID STUDIO

**Step – 1:**
Head over to bellow link to get the Android Studio executable or zip file .

https://developer.android.com/studio/#downloads

**Step – 2:**
Click on the download android studio button .


Click on the “I have read and agree with the above terms and conditions” checkbox followed by the download button.




Click on save file button in the appeared prompt box and the file will start downloading.

**Step – 3:**

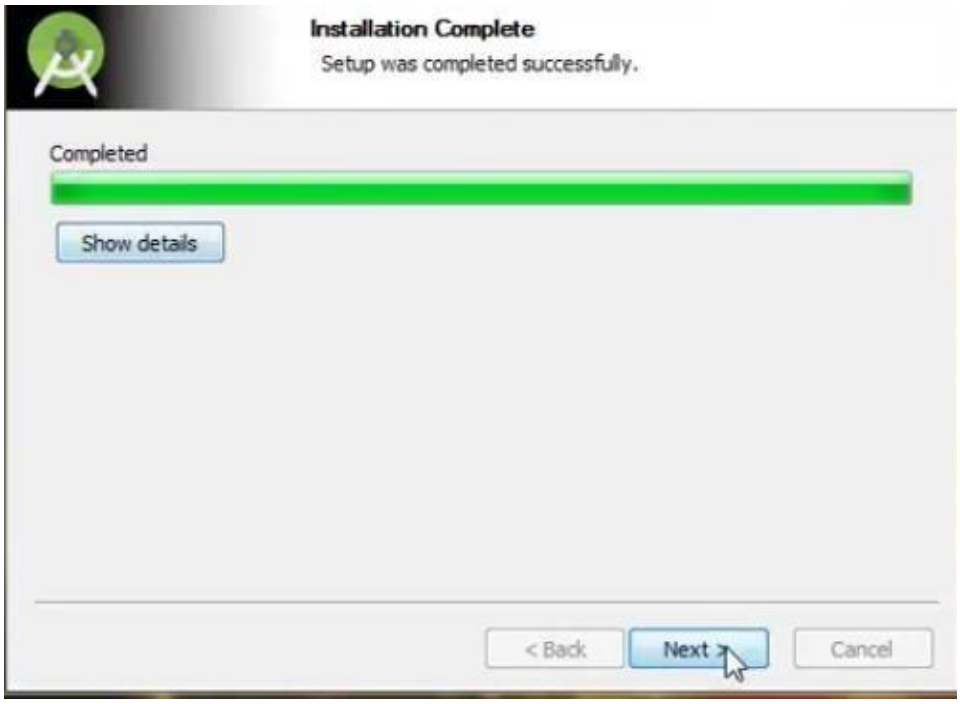
After the downloading has finished, open the file from downloads and run it . It will prompt the following dialogue box

Click on next.

In the next prompt it'll ask for a path for installation. Choose a path and hit next.

**Step – 4:**

It will start the installation, and once it is completed, it will be like the image shown below




Click on next

**Step – 5 :**

Once “Finish” is clicked, it will ask whether the previous settings needs to be imported [if android studio had been installed earlier], or not.

It is better to choose the ‘Don’t import Settings option’ .


**Step – 6 :**

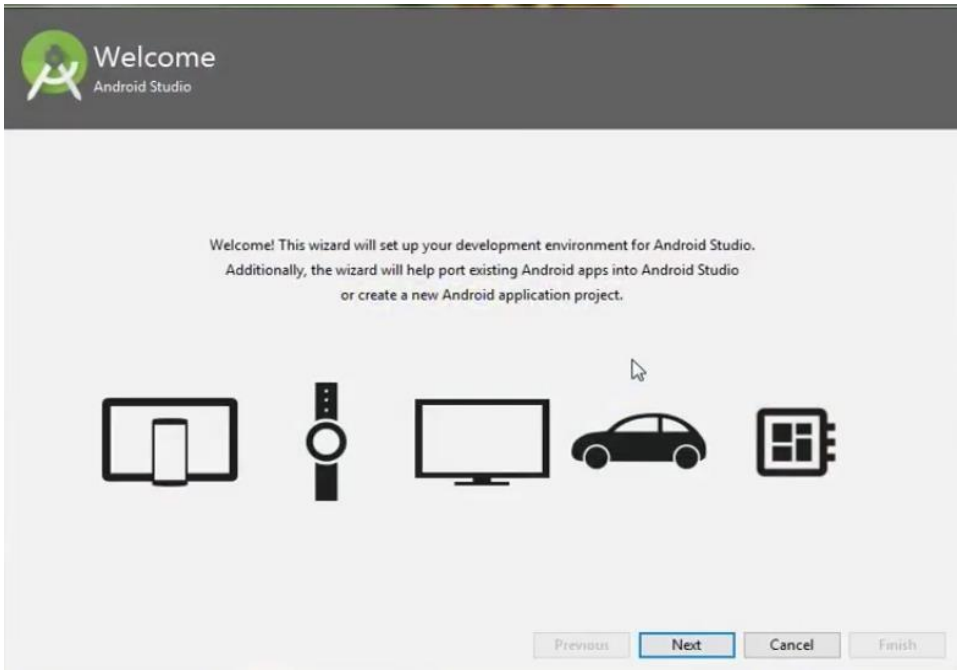
This will start the Android Studio.



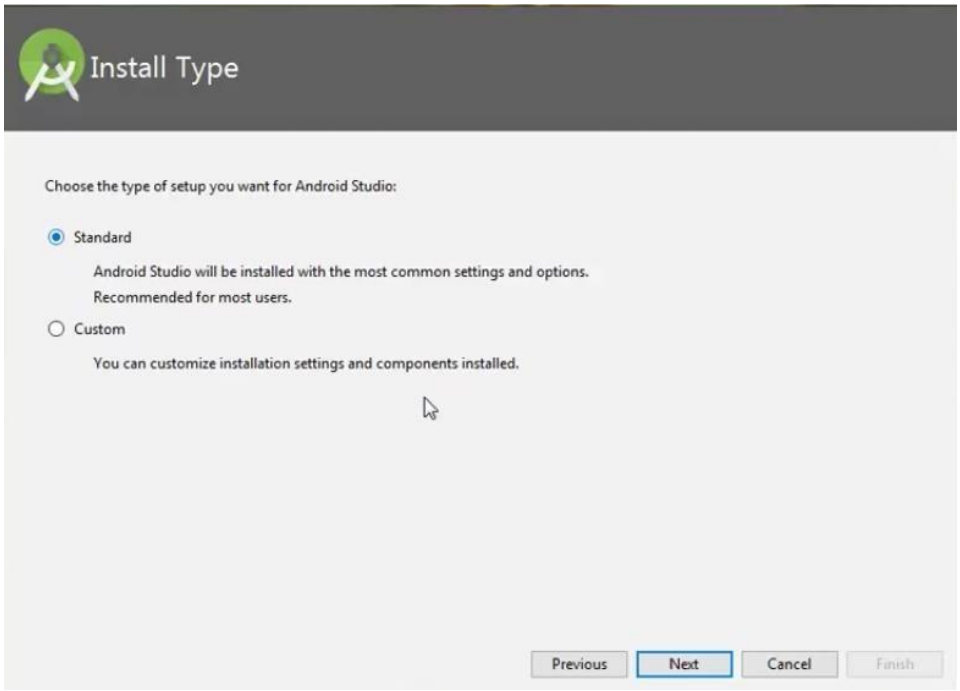
Meanwhile it will be finding the available SDK components .

**Step – 7:**

After it has found the SDK components, it will redirect to the Welcome dialog box .



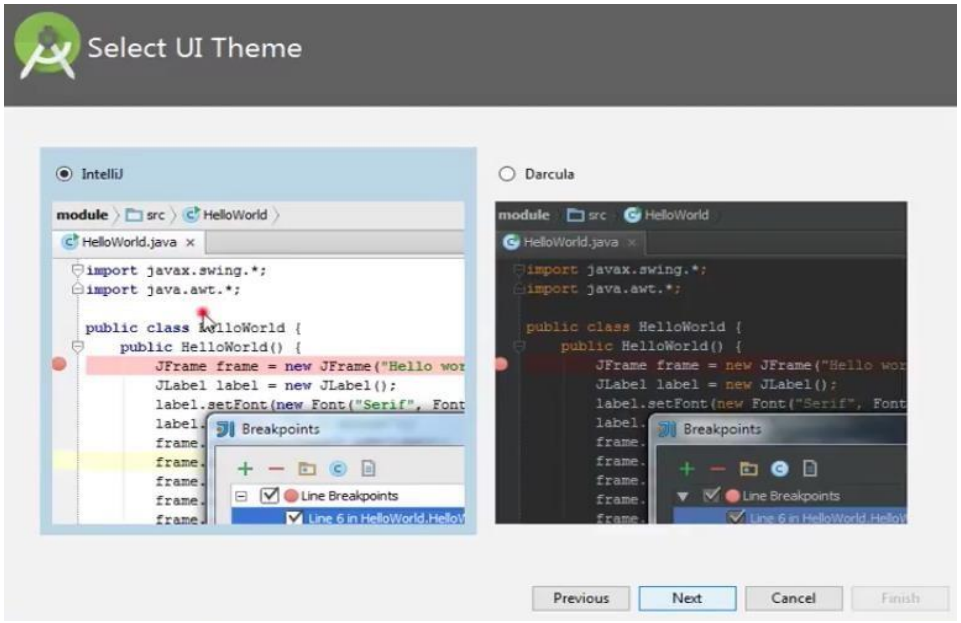

Click on next .




Choose Standard and click on Next.

Now choose the theme, whether Light theme or the Dark one .

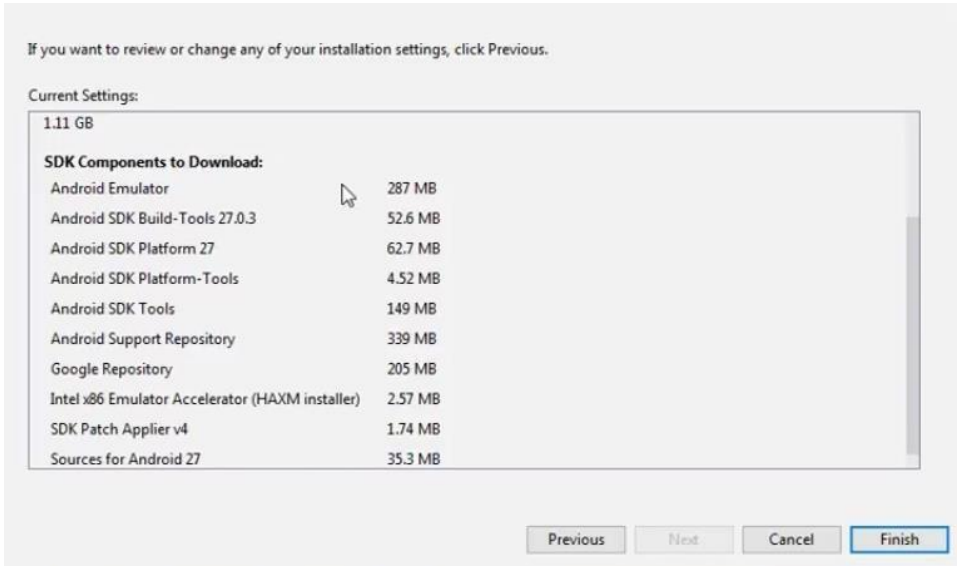
The light one is called the IntelliJ theme whereas the dark theme is called Darcula . Choose as required.




Click on the Next button

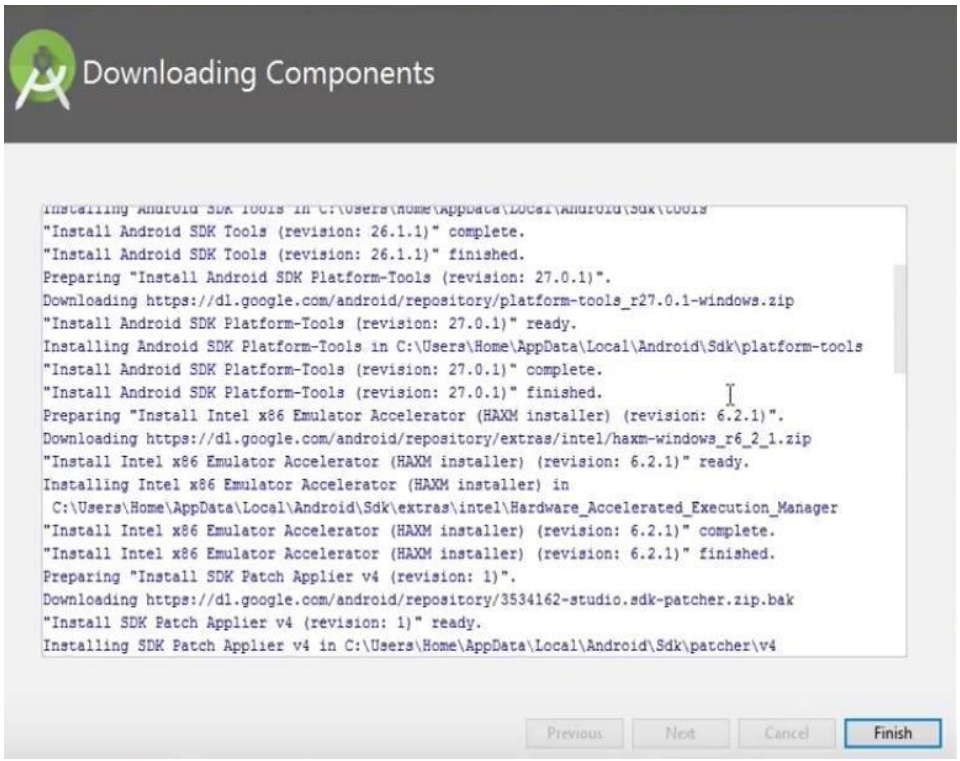
Step – 8 :

Now it is time to download the SDK components.




Click on Finish .

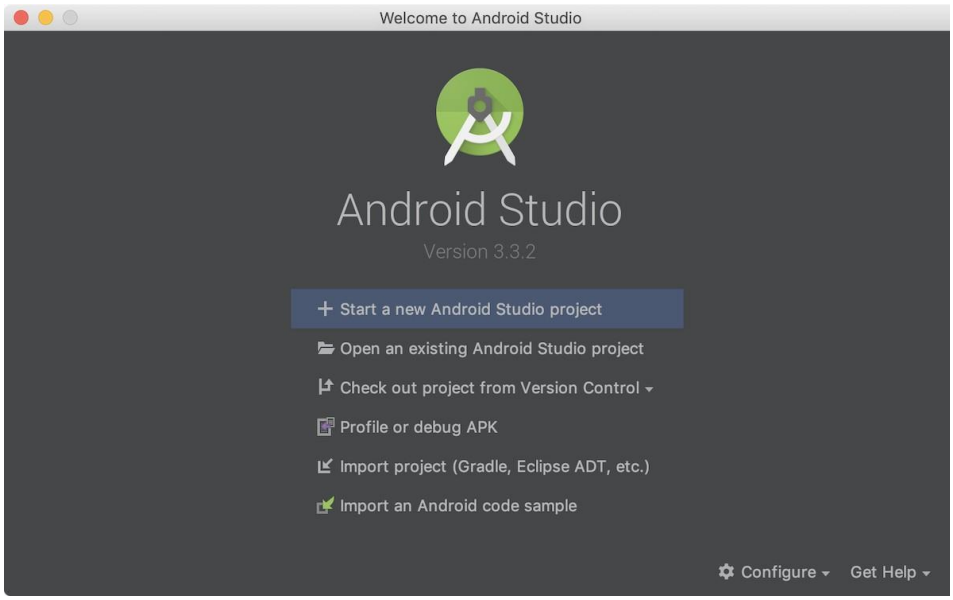
It has started downloading the components




The Android Studio has been successfully configured. Now it's time to launch and build app.

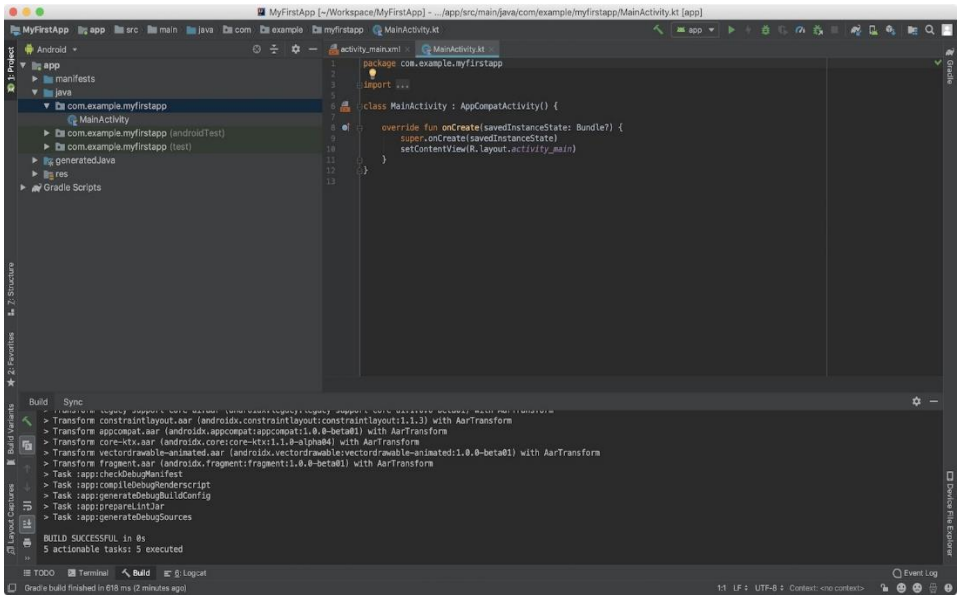
To create your new Android project, follow these steps:

- Install the latest version of Android Studio.
- In the Welcome to Android Studio window, click Start a new Android Studio project.



### 1.2.3 Creating A Project in Android

- If you have a project already opened, select File > New > New Project.
- In the Choose your project window, select Empty Activity and click Next.
- In the Configure your project window, complete the following:
    1. Enter "My First App" in the Name field.
    2. Enter "com.example.myfirstapp" in the Package name field.
    3. If you'd like to place the project in a different folder, change its Save location.
    4. Select either Java or Kotlin from the Language drop-down menu.
- Select the checkbox next to Use androidx.* artifacts.
- Leave the other options as they are.
- Click Finish.




- After some processing time, the Android Studio main window appears.

| To Open Project Window<br>select View > Tool Windows > ProjectTo Open MainActivity.java file<br>app > java > PackageName > MainActivity.javaTo Open Layout activity_main.xml file<br>app > res > layout > activity_main.xml |
|---|

### 1.2.4 Creating AVD in Android Studio

1. After the project is created, there are 2 files, MainActivity.java and activity_main.xml

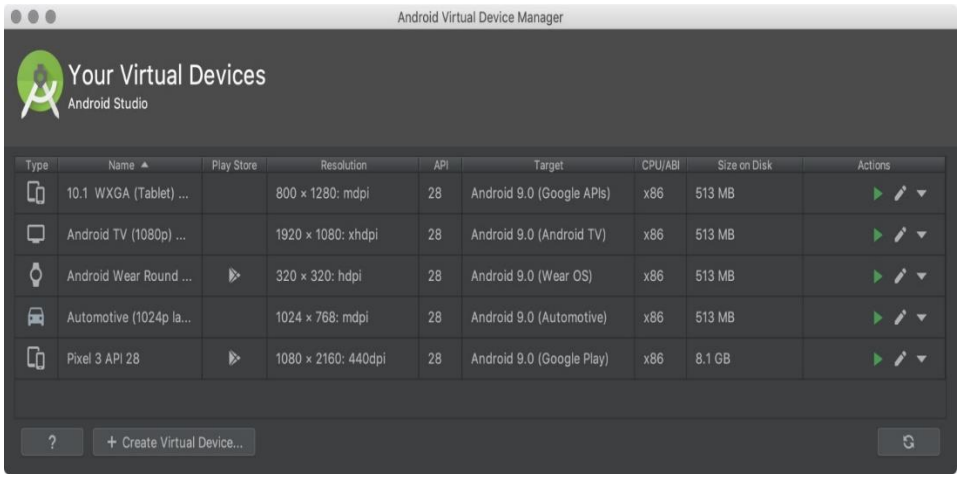
2. Go to activity_main.xml and select Design View
3. In Design View, change the layout to LinearLayout(Vertical) select

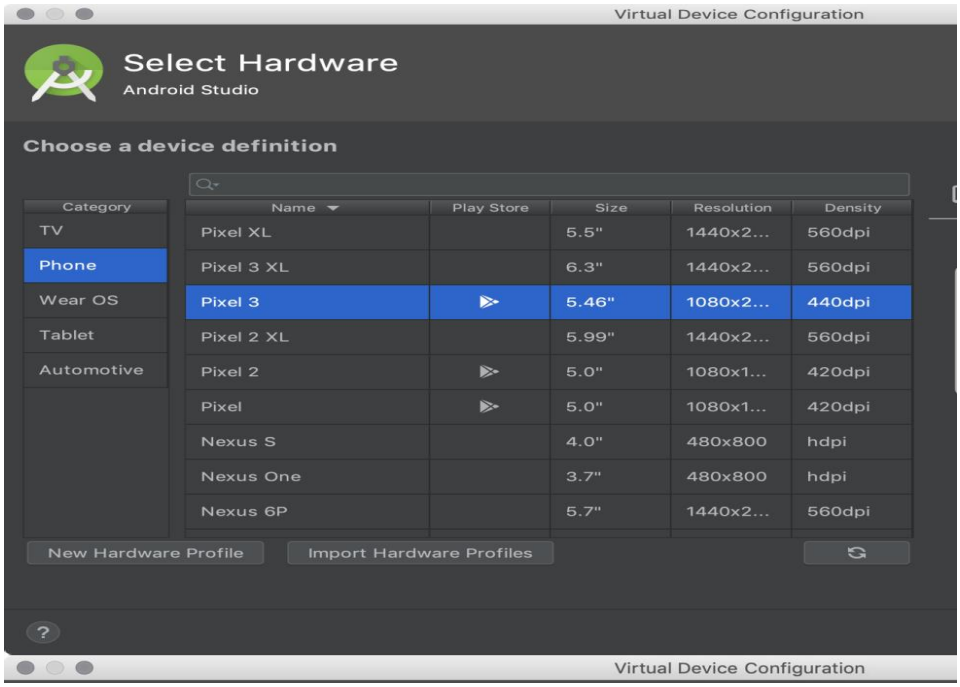
Add TextView, and change the text to “Hello World!”

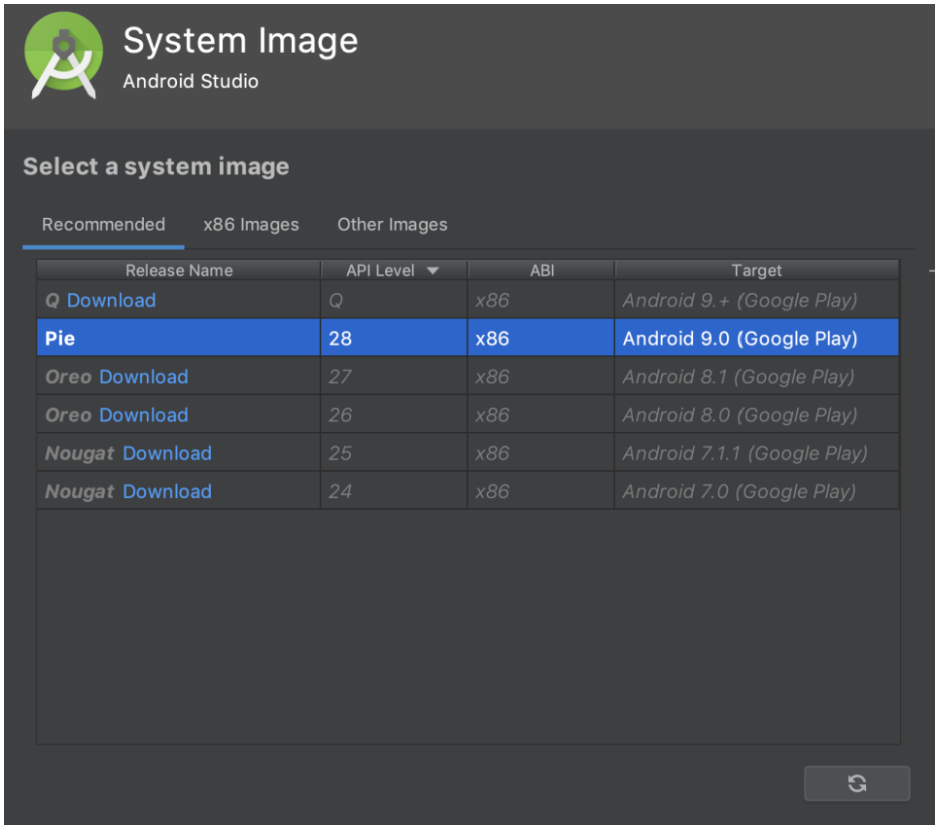
4. Click on Run and select the AVD already created(if not created, first create the AVD)
5. Output screen should show “Hello World”

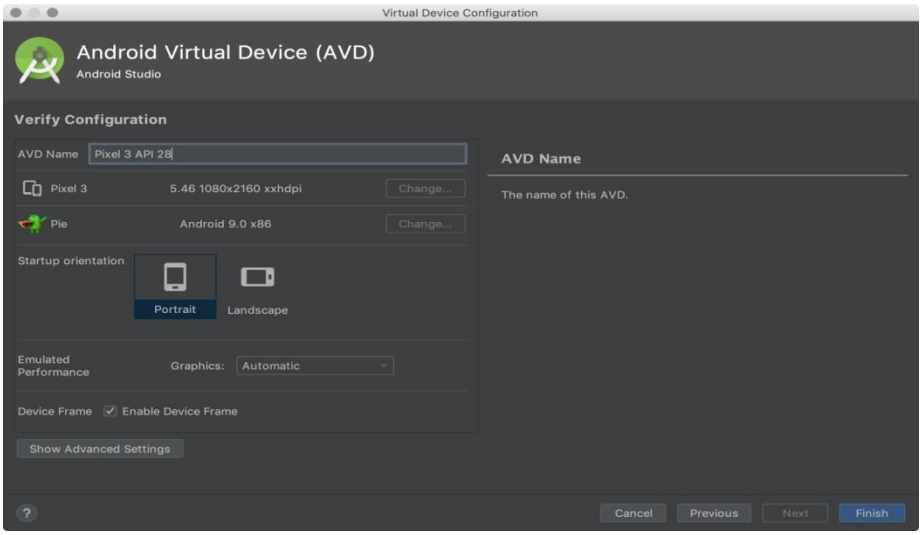
To create a new AVD:

1. Open the AVD Manager by clicking Tools > AVD Manager.
2. Click Create Virtual Device, at the bottom of the AVD Manager dialog....
3. Select a hardware profile, and then click Next.
4. Select the system image for a particular API level, and then click Next.
5. Change AVD properties as needed, and then click Finish.



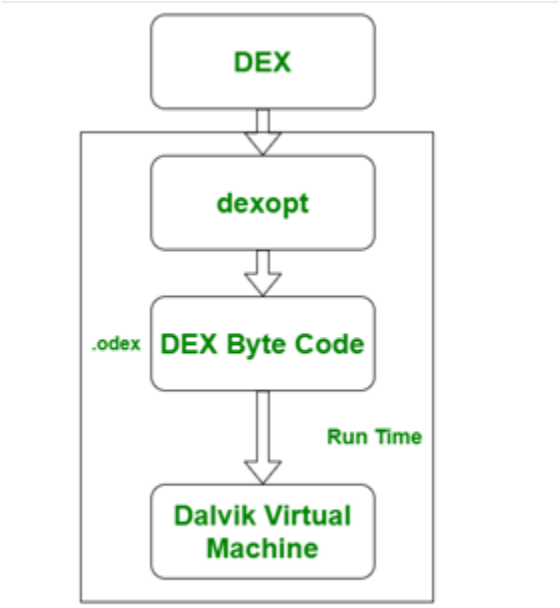

System Image
Android Studio
Select a system image
Recommended
x86 Images
Other Images
Release Name
API Level
ABI
Target
Q Download
Q
x86
Android 9.+ (Google Play)
Pie
28
x86
Android 9.0 (Google Play)
Oreo Download
27
x86
Android 8.1 (Google Play)
Oreo Download
26
x86
Android 8.0 (Google Play)
Nougat Download
25
x86
Android 7.1.1 (Google Play)
Nougat Download
24
x86
Android 7.0 (Google Play)


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.
Cancel
Previous
Next
Finish


Android Emulator - Android_Accelerated_x86_Oreo:5554
12:03
Hello World
Hello World!

## 1.3 ANDROID RUNTIME (ART)

- Dalvik Virtual Machine or DVM is a Register-Based virtual machine that was designed and written by Dan Bornstein.
- Dalvik is a discontinued process virtual machine (VM) in the Android OS that executes applications written for Android.

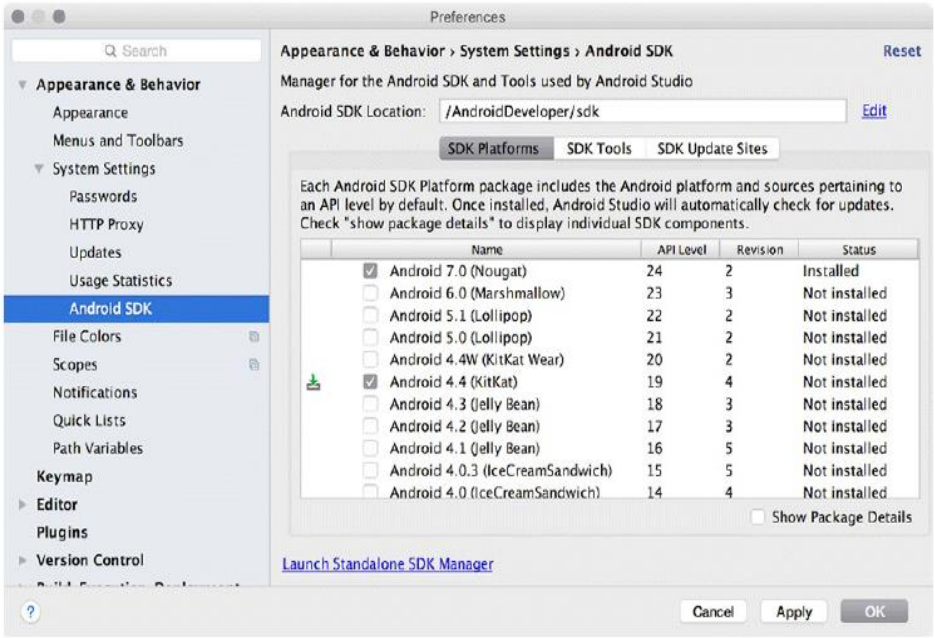



Dalvik Virtual Machine

## 1.4 THE ANDROID STUDIO

- Android Studio provides the SDK and the emulator system image from the latest platform.
- However, you may want to test your apps on earlier versions of Android.
- You can get components for each platform using the Android SDK Manager.
- In Android Studio, select Tools → Android → SDK Manager.
- Under the Quick Start section,
- select Configure → SDK Manager.)
- The SDK Manager is shown in Figure .

**Figure 1 Android SDK Manager**




## 1.5 INTRODUCTION TO GRADLE

- In Android Studio, Gradle is used for building our android application projects, hence playing the role of a build system.
- Gradle is a build system, which is responsible for code compilation, testing, deployment and conversion of the code into . dex files and hence running the app on the device.
- As Android Studio comes with Gradle system pre-installed, there is no need to install additional runtime softwares to build our project.
- Whenever you click on Run button in android studio, a gradle task automatically triggers and starts building the project and after gradle completes its task, app starts running in AVD or in the connected device.
- A build system like Gradle is not a compiler, linker etc, but it controls and supervises the operation of compilation, linking of files, running test cases, and eventually bundling the code into an apk file for your Android Application.

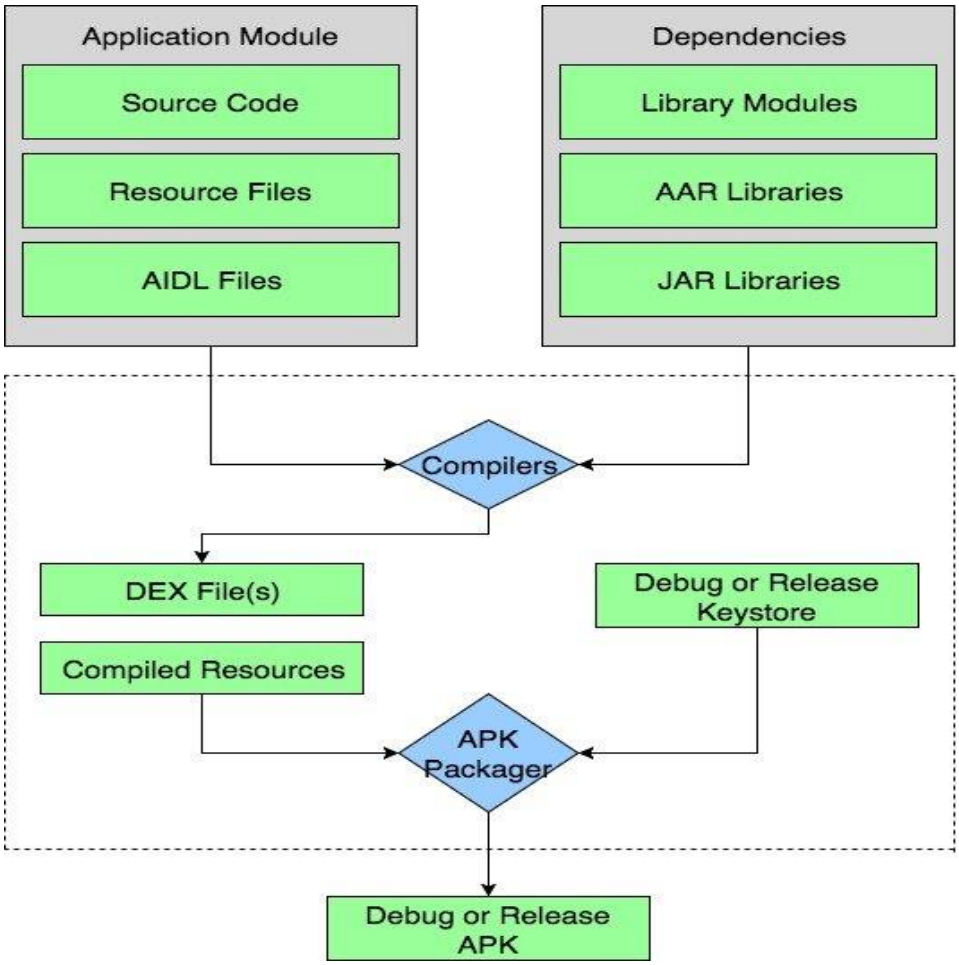
There are two build.gradle files for every android studio project of which, one is for application and other is for project level(module level) build files.

**build.gradle (project level)**

The Top level (module) build.gradle file is project level build file, which defines build configurations at project level. This file applies configurations to all the modules in android application project.

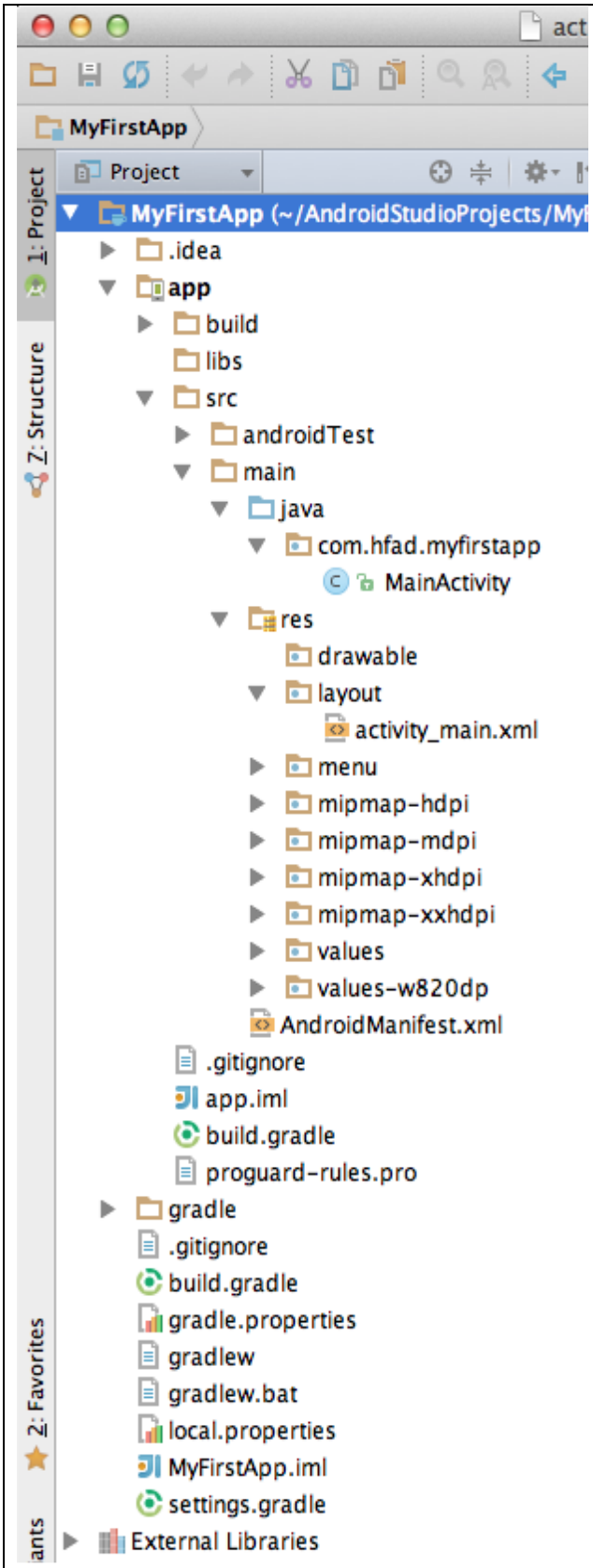
**build.gradle (application level)**

The Application level build.gradle file is located in each module of the android project. This file includes your package name as applicationID, version name(apk version), version code, minimum and target sdk for a specific application module. When you are including external libraries(not the jar files) then you need to mention it in the app level gradle file to include them in your project as dependencies of the application.




## 1.5.1 FUNDAMENTALS OF GRADLE

- An Android app is really just a bunch of valid files in a particular folder structure, and Android Studio sets all of this up for you when you create a new app.
- The easiest way of looking at this folder structure is with the explorer in the leftmost column of Android Studio.




The folder structure includes different types of files

If you browse through the folder structure, you'll see that the wizard has created various types of files and folders for you:

**Java and XML source files**

These are the activity and layout files the wizard created for you.

**Resource files**

These include default image files for icons, styles your app might use, and any common String values your app might want to look up.

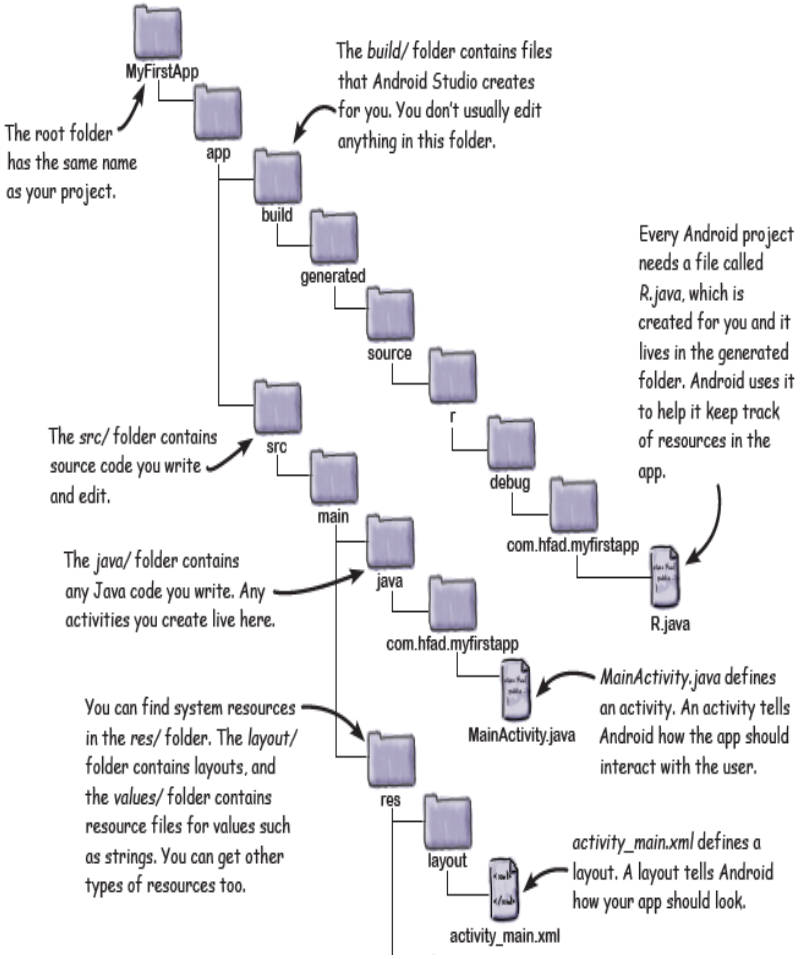
**Android libraries**

In the wizard, you specified the minimum SDK version you want your app to be compatible with. Android Studio makes sure it includes the relevant Android libraries for this version.

**Configuration files**

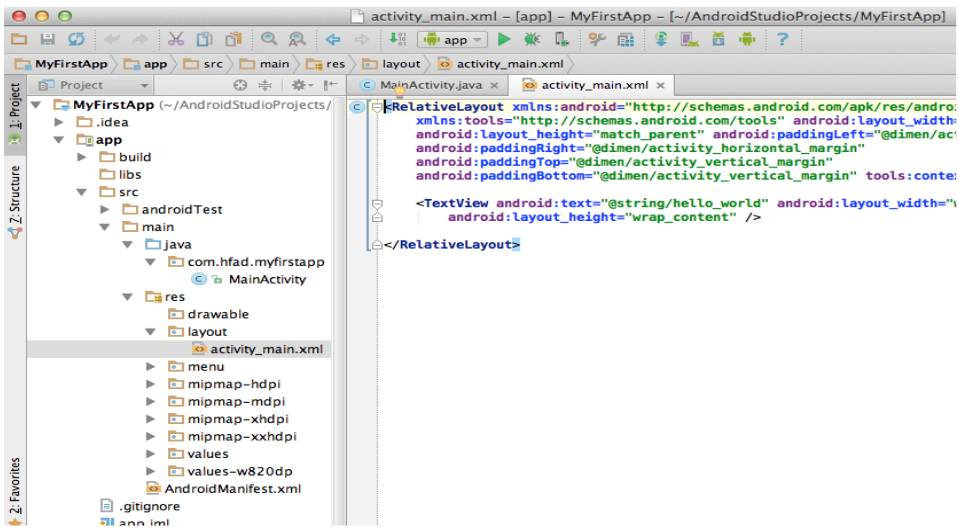
The configuration files tell Android what's actually in the app and how it should run.

- Android Studio projects use the gradle build system to compile and deploy your apps. Gradle projects have a standard layout. Here are some of the key files and folders you'll be working with:



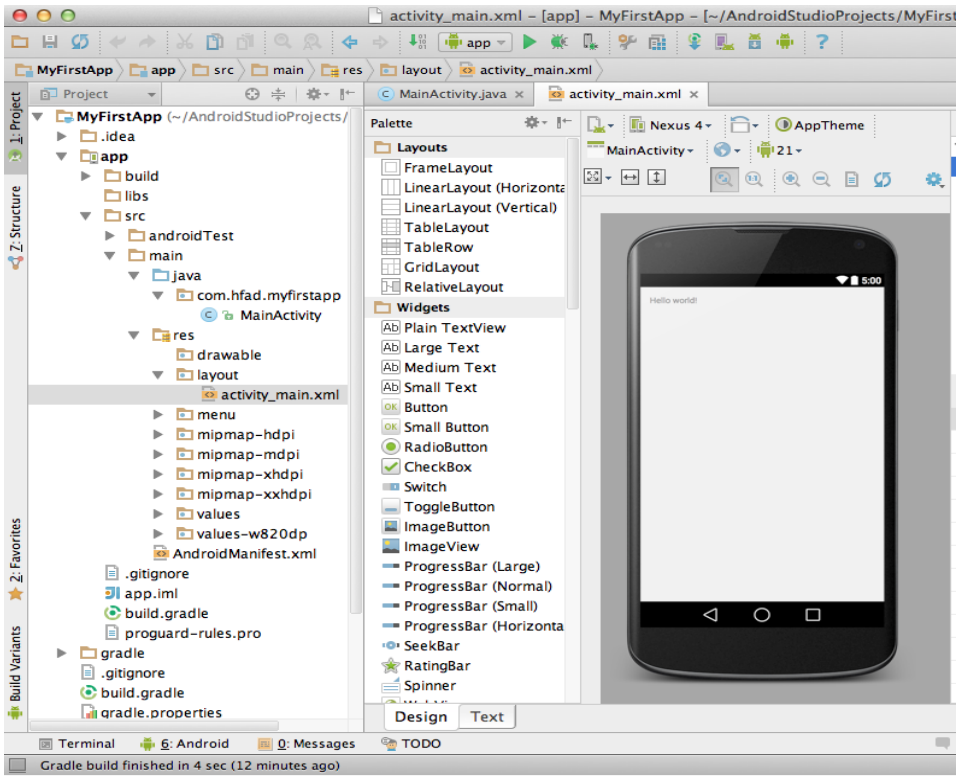

### 1.5.1.1 The code editor

Most files get displayed in the code editor. The code editor is just like a text editor, but with extra features such as color coding and code checking.



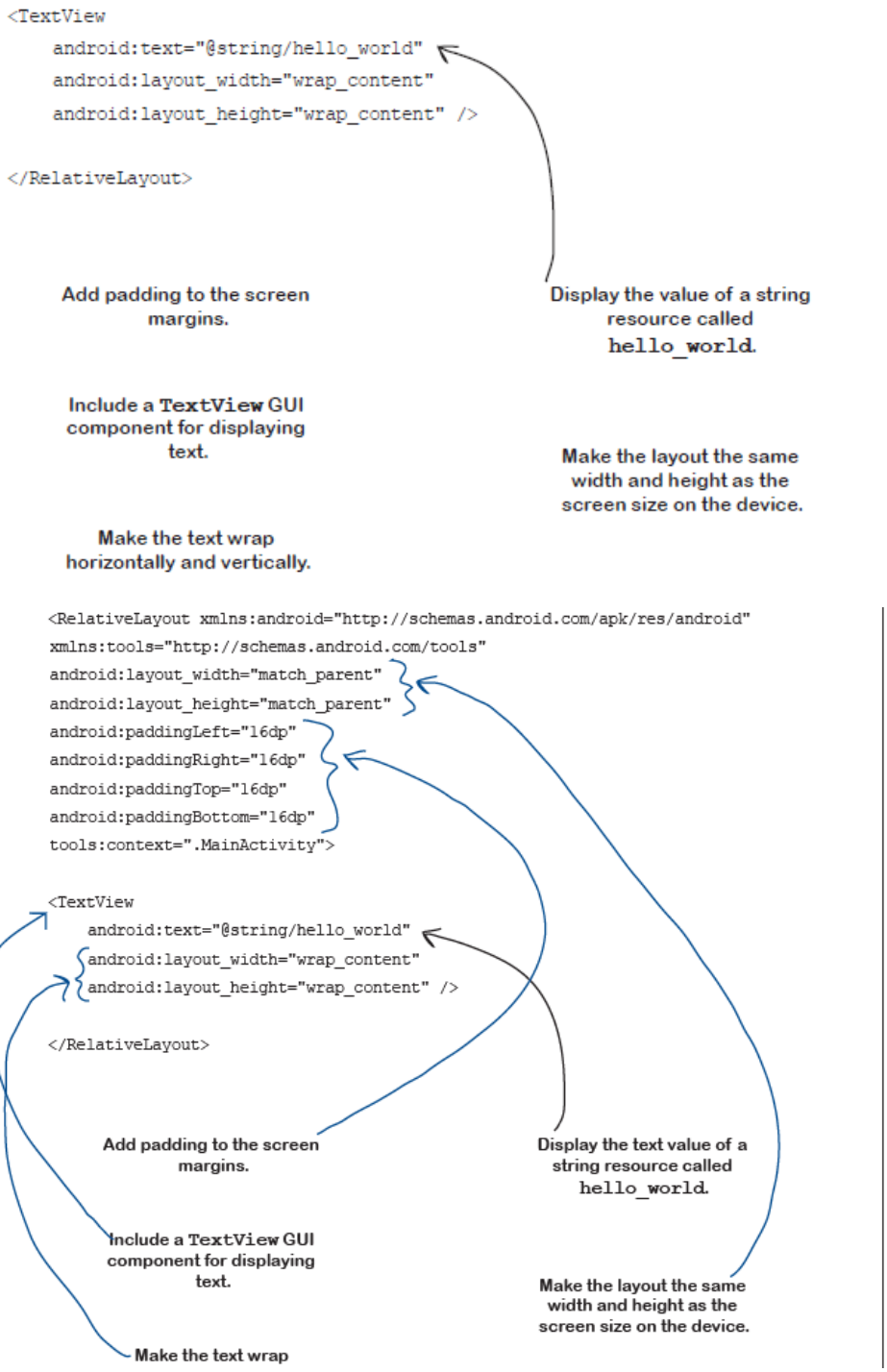
### 1.5.1.2 The Design Editor

- If you're editing a layout, you have an extra option. Rather than edit the XML, you can use the design editor.
- The design editor allows you to drag GUI components onto your layout, and arrange them how you want.
- The code editor and design editor give different views of the same file, so you can switch back and forth between the two.




**activity_main.xml**

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingLeft="16dp"
android:paddingRight="16dp"
android:paddingTop="16dp"
android:paddingBottom="16dp"
tools:context=".MainActivity">
```




## 1.6 BASIC BUILDING BLOCKS

- An android **component** is simply a piece of code that has a well defined life cycle e.g. Activity, Receiver, Service etc.
- The core building blocks or fundamental components of android are activities, views, intents, services, content providers, fragments and AndroidManifest.xml.

### 1.6.1 Activity and View

- An activity is a class that represents a single screen. It is like a Frame in AWT.
- A view is the UI element such as button, label, text field etc. Anything that you see is a view.

### 1.6.2 Intent

- Intent is used to invoke components. It is 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.**

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

## 1.4 SERVICE

- Service is a background process that can run for a long time.
- There are two types of services local and remote.
- Local service is accessed from within the application whereas remote service is accessed remotely from other applications running on the same device.

**Content Provider**

- Content Providers are used to share data between the applications.

**Fragment**

- Fragments are like parts of activity. An activity can display one or more fragments on the screen at the same time.

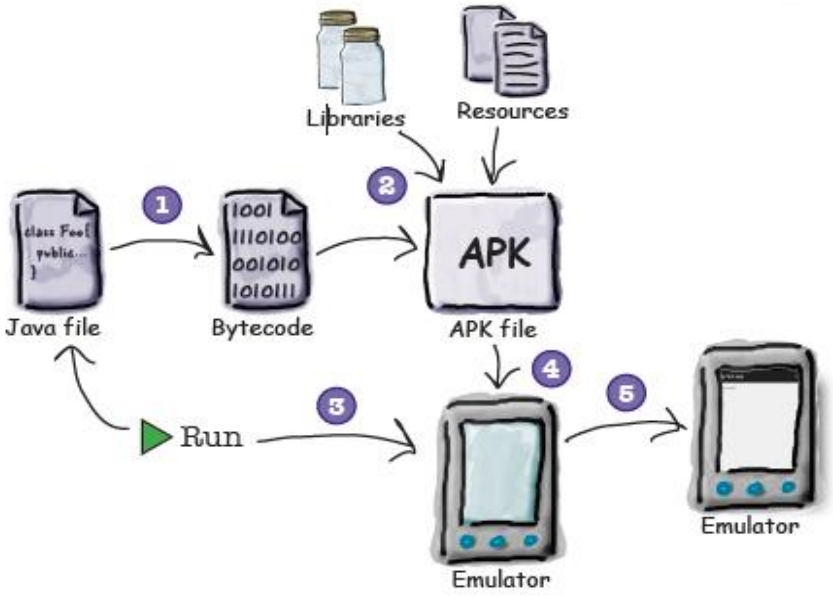
**AndroidManifest.xml**

- It contains informations about activities, content providers, permissions etc. It is like the web.xml file in Java EE.

### 1.6.4 Android Virtual Device (AVD)

- It is used to test the android application without the need for mobile or tablet etc. It can be created in different configurations to emulate different types of real devices.
- The Android emulator allows you to run your app on an Android virtual device (AVD). The AVD behaves just like a physical Android device.
- You can set up numerous AVDs, each emulating a different type of device.

**Compile, package, deploy and run**




- The Java source files get compiled to bytecode.
- An Android application package, or APK file, gets created.
- The APK file includes the compiled Java files, along with any libraries and resources needed by your app.
- Assuming there's not one already running, the emulator gets launched with the AVD.
- Once the emulator has been launched and the AVD is active, the APK file is uploaded to the AVD and installed.
- The AVD starts the main activity associated with the app.
- Your app gets displayed on the AVD screen, and it's all ready for you to test out.

**activity_main.xml has two elements**

Here's the code from activity_main.xml that Android Studio generated for us.

**The code contains two elements.**

- The first element is the <RelativeLayout> element. This element tells Android to display items on the layout in relative positions. You can use <RelativeLayout>, for instance, to center items in the middle of the layout, align them to the bottom of the screen on your Android device, or position them relative to other items.

- The second element is the <TextView> element. This element is used to display text to the user. It's nested within the <RelativeLayout>, and in our case it's being used to display the sample text "Hello world!".

**Strings.xml**

- strings.xml is the default resource file used to hold name/value pairs of strings so that they can be referenced throughout your app.

- Android Studio created a string resource file for us called strings.xml, so let's see if it contains a hello_world resource. Use the explorer to find it in the app/src/main/res/values folder, and open it by double-clicking on it.

- Here's what our code in the strings.xml file looks like:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">My First App</string>
<string name="hello_world">Hello world!</string>
<string name="action_settings">Settings</string>
</resources>
```

- strings.xml includes a string with a name of hello_world, and a value of "Hello world!".

- As you can see, there's a line of code that looks just like what we are looking for. It describes a string resource with a name of hello_ world, and a value of "Hello world!":

**<string name="hello_world">Hello world!</string>**

- There are two things that allow Android to recognize strings.xml as being a string resource file:

- The file is held in the folder app/src/main/res/values.

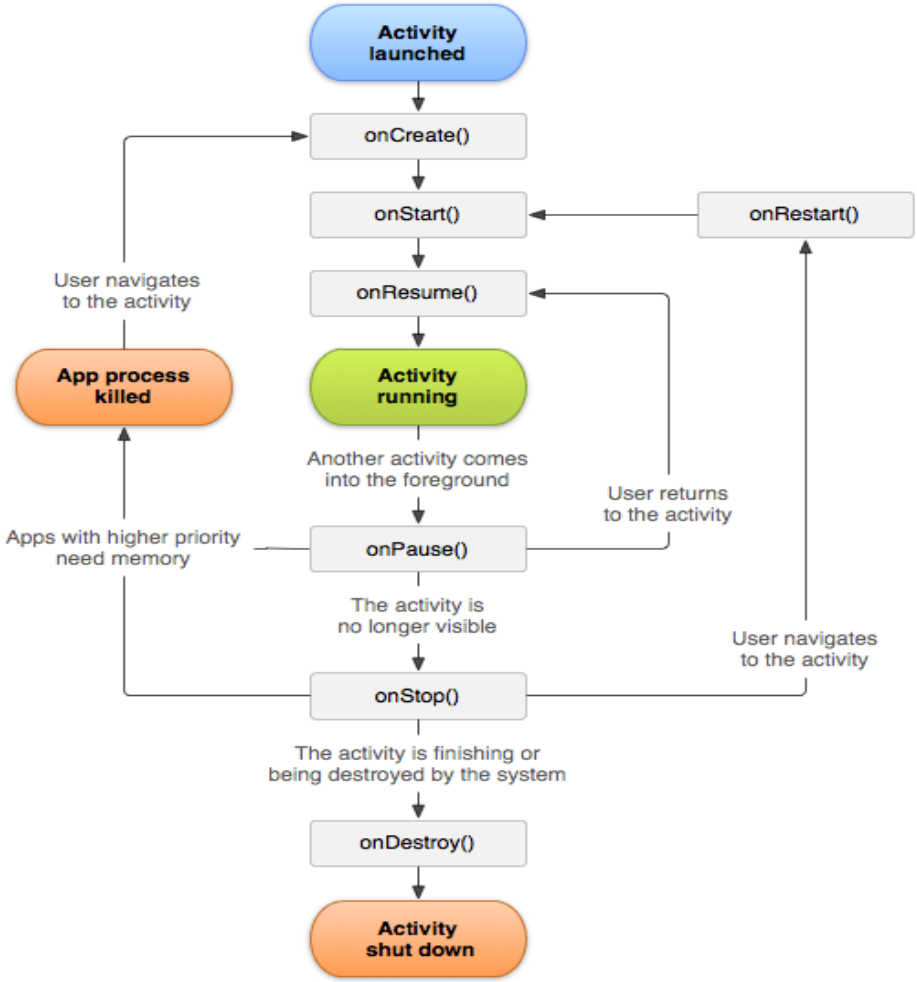
- XML files held in this folder contain simple values, such as strings and colors.

- The file has a <resources> element, which contains one or more <string> elements.

## Activities

- An activity provides the window in which the app draws its UI. This window typically fills the screen, but may be smaller than the screen and float on top of other windows. ... Typically, one activity in an app is specified as the main activity, which is the first screen to appear when the user launches the app.
- Android Activity Lifecycle is controlled by 7 methods of android.app.Activity class. The android Activity is the subclass of ContextThemeWrapper class.
- An activity is the single screen in android. It is like window or frame of Java.
- By the help of activity, you can place all your UI components or widgets in a single screen.
- The 7 lifecycle method of Activity describes how activity will behave at different states.

| 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. |




```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.androi.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="example.javatpoint.com.activitylifecycle.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" />
</android.support.constraint.ConstraintLayout>
```

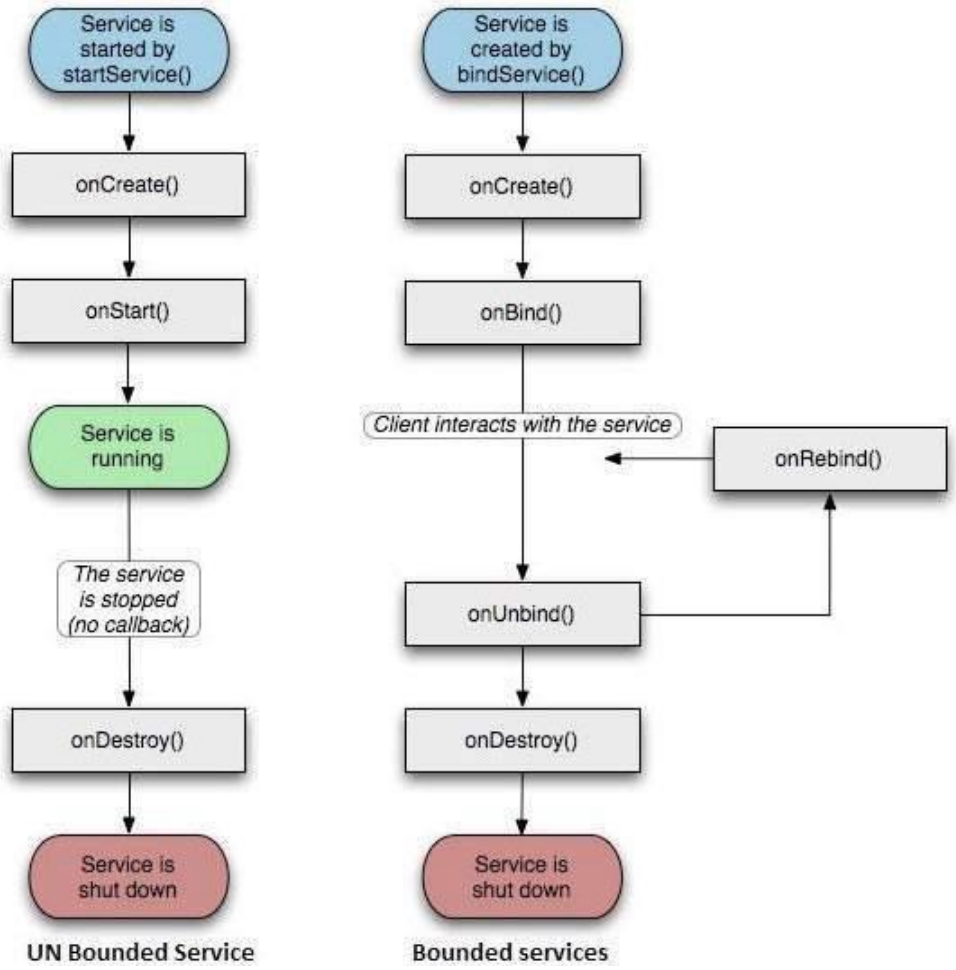
```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

   @Override
  protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Log.d("lifecycle","onCreate invoked");
   }
   @Override
   protected void onStart() {
      super.onStart();
      Log.d("lifecycle","onStart invoked");
   }
   @Override
   protected void onResume() {
      super.onResume();
      Log.d("lifecycle","onResume invoked");
   }
   @Override
   protected void onPause() {
      super.onPause();
      Log.d("lifecycle","onPause invoked");
   }
   @Override
   protected void onStop() {
      super.onStop();
     Log.d("lifecycle","onStop invoked");
   }
   @Override
    protected void onRestart() {
      super.onRestart();
      Log.d("lifecycle","onRestart invoked");
   }
   @Override
   protected void onDestroy() {
      super.onDestroy();
      Log.d("lifecycle","onDestroy invoked");
   }
}
```

### 1.6.6 Android Services

- A service is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed.

- A service can essentially take two states −

| Sr.No. | State & Description |
|---|---|
| 1 | Started<br>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. |
| 2 | Bound<br>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). |

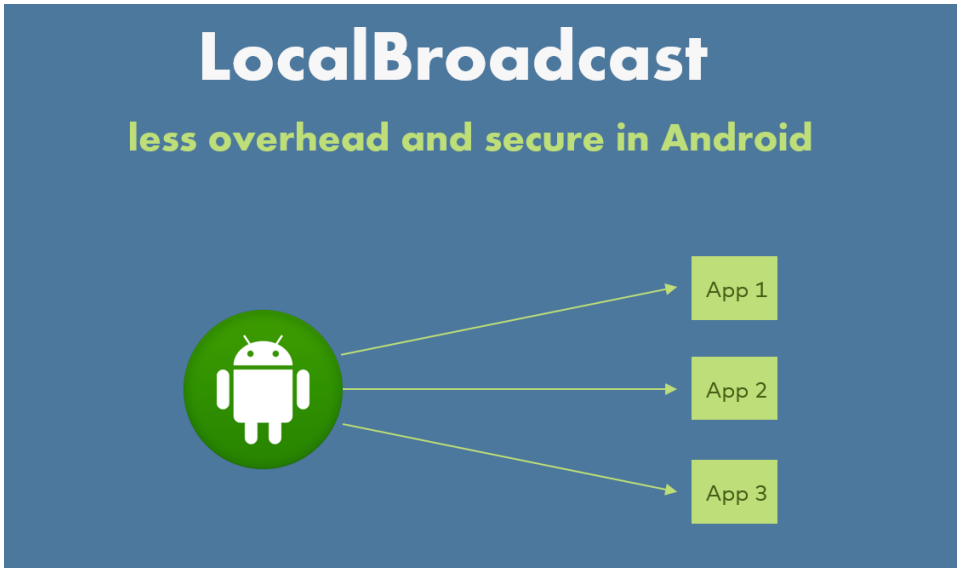


### 1.6.7 Broadcast Receiver and Content Provider

- Broadcast receiver is an Android component which allows you to send or receive Android system or application events. All the registered application are notified by the Android runtime once event happens.
- It works similar to the publish subscribe design pattern and used for asynchronous inter-process communication.
- For example, applications can register for various system events like boot complete or battery low, and Android system sends broadcast when specific event occur. Any application can also create its own custom broadcasts.

**Register Broadcast**

- There are two ways to register broadcast receiver-
- Manifest-declared (Statically) : By this receiver can be registered via the AndroidManifest.xml file.
- Context-registered (Dynamically) : By this register a receiver dynamically via the Context.registerReceiver() method.

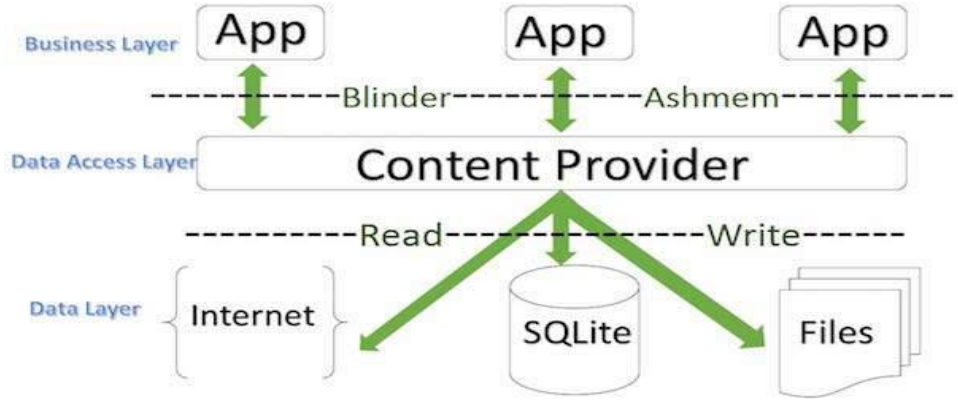



```
Create a new instance of the LocalBroadcastManager

LocalBroadcastManager localBroadcastManager =
LocalBroadcastManager.getInstance(context);
```

**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.




Sometimes it is required to share data across applications. This is where content providers become very useful.

**public class My Application extends ContentProvider {**

**}**

## 1.7 UI COMPONENTS

- Android UI Controls are those components of Android that are used to design the UI in a more interactive way.
- It helps us to develop an application that makes user interaction better with the view components.
- Android provides us a huge range of UI controls of many types such as buttons,
- text views, etc.
- A View is an object that draws something on the screen that the user can interact with and a ViewGroup is an object that holds other View (and ViewGroup) objects in order to define the layout of the user interface.

### 1.7.1 TextView

- TextView is a UI Component that displays the text to the user on their Display Screen.
- We can create it in two ways:
- XML file:
- For this, we declare it in the layout tag as follows:

```
<Linear Layout xmls:android= "http://schemas.android.com/apk/res/android"
    <TextView
    //attributes to describe it
    />
</LinearLayout>
```

**Activity file:**

In this, we declare it using the setText() method as follows:

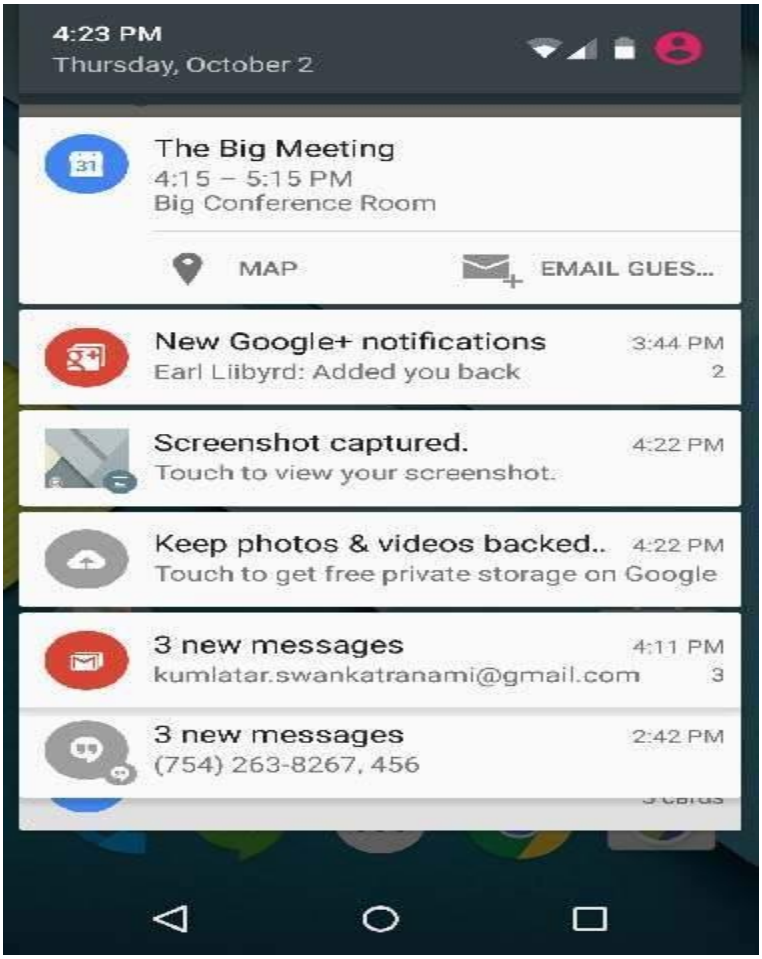
```
setContentView(R.layout.activity_main);
LinearLayout linearlayout_name = (LinearLayout)findViewById(R.id.LinearLayout);
TextView textview_name = new TextView(this);
textveiw_name.setText("Hello I am Text View");
linearLayout.addView(textView);
```

| Sr.No. | UI Control & Description |
|---|---|
| 1 | TextView<br>This control is used to display text to the user. |
| 2 | EditText<br>EditText is a predefined subclass of TextView that includes rich editing capabilities. |
| 3 | AutoCompleteTextView<br>The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing. |
| 4 | Button<br>A push-button that can be pressed, or clicked, by the user to perform an action. |
| 5 | ImageButton<br>An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user. |

| Sr.No. | UI Control & Description |
|---|---|
| 6 | CheckBox<br>An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive. |
| 7 | ToggleButton<br>An on/off button with a light indicator. |
| 8 | RadioButton<br>The RadioButton has two states: either checked or unchecked. |
| 9 | RadioGroup<br>A RadioGroup is used to group together one or more RadioButtons. |
| 10 | ProgressBar<br>The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background. |
| 11 | Spinner<br>A drop-down list that allows users to select one value from a set. |
| 12 | TimePicker<br>The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode. |
| 13 | DatePicker<br>The DatePicker view enables users to select a date of the day. |

### 1.7.2 Notification

- A notification is a message you can display to the user outside of your application's normal UI.
- When you tell the system to issue a notification, it first appears as an icon in the notification area.
- To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time
- Android Toast class provides a handy way to show users alerts but problem is that these alerts are not persistent which means alert flashes on the screen for a few seconds and then disappears




## 1.8 COMPONENTS FOR COMMUNICATION –INTENTS

- An intent is to perform an action on the screen. It is mostly used to start activity, send broadcast receiver,start services and send message between two activities.
- There are two intents available in android as Implicit Intents and Explicit Intents.
- An Intent is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways, there are three fundamental use cases:

**Starting an activity**

- An Activity represents a single screen in an app. You can start a new instance of an Activity by passing an Intent to startActivity(). The Intent describes the activity to start and carries any necessary data.
- If you want to receive a result from the activity when it finishes, call startActivityForResult(). Your activity receives the result as a separate Intent object in your activity's onActivityResult() callback. For more information, see the Activities guide.

**Starting a service**

- A Service is a component that performs operations in the background without a user interface. With Android 5.0 (API level 21) and later, you can start a service with JobScheduler. For more information about JobScheduler, see its API-reference documentation.
- For versions earlier than Android 5.0 (API level 21), you can start a service by using methods of the Service class. You can start a service to perform a one-time operation (such as downloading a file) by passing an Intent to startService(). The Intent describes the service to start and carries any necessary data.
- If the service is designed with a client-server interface, you can bind to the service from another component by passing an Intent to bindService(). For more information, see the Services guide.

**Delivering a broadcast**

- A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an Intent to sendBroadcast() or sendOrderedBroadcast().

**Intent types**

- There are two types of intents:
- Explicit intents specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name.
- You will 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.
- For example, you might start a new activity within your app in response to a user action, or start a service to download a file in the background.
- Implicit intents do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.
- For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

**Implicit Intent Syntax**

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

**Explicit Intent Syntax:**

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

### 1.8.1 Intent Filters

- You have seen how an Intent has been used to call an another activity.
- Android OS uses filters to pinpoint the set of Activities, Services, and Broadcast receivers that can handle the Intent with help of specified set of action, categories, data scheme associated with an Intent. You will use <intent-filter> element in the manifest file to list down actions, categories and data types associated with any activity, service, or broadcast receiver.
- Following is an example of a part of AndroidManifest.xml file to specify an activity com.example.My Application.CustomActivity which can be invoked by either of the two mentioned actions, one category, and one data −

```
<activity android:name=".CustomActivity"
  android:label="@string/app_name">

  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <action android:name="com.example.My Application.LAUNCH" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="http" />
  </intent-filter>

</activity>
```

## 1.9 ANDROID VERSIONS, NAME, AND API LEVEL

- The development of the Android operating system was started in 2003 by Android, Inc. Later on, it was purchased by Google in 2005. The beta version of Android OS was released on November 5, 2007, while the software development kit (SDK) was released on November 12, 2007.
- The first Android mobile was publicly released with Android 1.0 of the T-Mobile G1 (aka HTC Dream) in October 2008.

| Code name | Version numbers | API level | Release date |
|---|---|---|---|
| No codename | 1.0 | 1 | September 23, 2008 |
| No codename | 1.1 | 2 | February 9, 2009 |
| Cupcake | 1.5 | 3 | April 27, 2009 |
| Donut | 1.6 | 4 | September 15, 2009 |
| Eclair | 2.0 - 2.1 | 5 - 7 | October 26, 2009 |
| Froyo | 2.2 - 2.2.3 | 8 | May 20, 2010 |
| Gingerbread | 2.3 - 2.3.7 | 9 - 10 | December 6, 2010 |
| Honeycomb | 3.0 - 3.2.6 | 11 - 13 | February 22, 2011 |
| Ice Cream Sandwich | 4.0 - 4.0.4 | 14 - 15 | October 18, 2011 |
| Jelly Bean | 4.1 - 4.3.1 | 16 - 18 | July 9, 2012 |
| KitKat | 4.4 - 4.4.4 | 19 - 20 | October 31, 2013 |
| Lollipop | 5.0 - 5.1.1 | 21- 22 | November 12, 2014 |
| Marshmallow | 6.0 - 6.0.1 | 23 | October 5, 2015 |
| Nougat | 7.0 | 24 | August 22, 2016 |
| Nougat | 7.1.0 - 7.1.2 | 25 | October 4, 2016 |
| Oreo | 8.0 | 26 | August 21, 2017 |
| Oreo | 8.1 | 27 | December 5, 2017 |
| Pie | 9.0 | 28 | August 6, 2018 |
| Android 10 | 10.0 | 29 | September 3, 2019 |
| Android 11 | 11 | 30 | September 8, 2020 |

## 1.10 SUMMARY

- This unit has provided a brief overview of Android, and highlighted some of its capabilities.
- If you have followed the sections on downloading the tools and the Android SDK, you should now have a working system — one that is capable of developing more interesting Android applications other than the Hello World application.
- In the next unit, you will learn about the concepts of activities and intents, and the very important roles they play in Android

## 1.11 KEYWORD

- SDK : Software development kit use to deploy android application.
- AVD : Android Virtual Device creates virtual environment for android application.
- Android Studio : is used to develop android application.
- Kernel : Kernel is linux based in android.
- Intent : It is used to connect activities.

## 1.12 LEARNING ACTIVITY

1. Define TextView?

_____

_____

_____

_____

2. Define Layout?

_____

_____

_____

_____

## 1.13 UNIT END QUESTIONS

**A. Descriptive Questions**

Short Answer

1. What is an AVD?
2. What is the difference between the android:versionCode and android:versionName attributes in the AndroidManifest.xml file?
3. What is the use of the strings.xml file?
4. What is Activity ? Draw and Explain Activity Life Cycle?

Long Answer

1. Write down the steps to download and install android studio.
2. Explain any TWO UI components?
3. What is intent? Why it is used?
4. Explain the concept of notifications in Android.

**B. Multiple Choice Questions**

1) How many sizes are supported by Android?
   a) Android supported all sizes
   b) Android does not support all sizes
   c) Android supports small,normal, large and extra-large sizes
   d) Size is undefined in android

2) How many broadcast receivers are available in android?

   a) sendIntent()

   b) onRecieve()

   c) implicitBroadcast()

   d) sendBroadcast(),sendOrderBroadcast(),and sendStickyBroadcast().

3) What is LastKnownLocation in android?

   a) To find the last location of a phone

   b) To find known location of a phone

   c) To find the last known location of a phone.

   d) To find the last known location of user

4) How to find the JSON element length in android JSON?

   a) count()

   b) sum()

   c) add()

   d) length()

5) What is off-line synchronization in android?

   a) Synchronization with internet

   b) Background synchronization

   c) Synchronization without internet

   d) Foreground synchronization

**Answers**
1-c, 2-d, 3-c. 4-d, 5-c

## 1.14 REFERENCES

**References book**

1) "Professional Android 4 Application Development" by Reto Meier

2) "Programming Android Java Programming for the New Generation of Mobile Devices" by Zigurd Mennieks

3) "Android Cookbook" by Ian F Darwin

4) "Android Programming: The Big Nerd Ranch Guide" by Bill Phillips and Chris Stewart

**Textbook references**

1) "Professional Android 4 Application Development" by Reto Meier

2) "Programming Android Java Programming for the New Generation of Mobile Devices" by Zigurd Mennieks

3) "Android Cookbook" by Ian F Darwin

4) "Android Programming: The Big Nerd Ranch Guide" by Bill Phillips and Chris Stewart

**Website**

Introduction to Android: http://developer.android.com/guide/index.html.

Android API: http://developer.android.com/reference/packages.html

Java 6 API: http://docs.oracle.com/javase/6/docs/api/

Android Fundamentals: http://developer.android.com/guide/components/fundamentals.html

❄❄❄❄❄❄❄

# 2

# USER INPUT CONTROLS

**Unit Structure :**



## 2.0 OBJECTIVES

After going through this chapter you will be able to

1. Know the various user controls that can be used while app development
2. Know various types of menus and drawables in Android
3. Know the difference between themes and styles
4. Know how adaptive layout work while using screen

## 2.1 USER INPUT CONTROLS

**Designing Android UI**

An Android Operating System (OS) interacts with users through different devices. It also communicates via an intermediary called the User Interface (UI). Through the user interface, the end user is able to see and interact with it. UI is used to navigate and utilize various components of Android OS on an Android application.

They are represented to the user in the following different forms:

- **Graphical User Interface (GUI)** allows users to interact with visual representations on digital devices or smartphones.
- **Voice-controlled interfaces** help users to interact via voice commands such as Alexa.

- **Gesture-based interface** lets users interact with the interface through body motions.

Sometimes, the implementation of a weakly designed UI results in facing difficulties while interacting with the Android OS. So, the UI layout needs to be professional with effective navigation designed for users.

**Android UI design**

Android UI design includes the use of prebuilt Android UI components, such as UI controls, to create an efficient GUI for our applications. A UI screen of an Android App consists of four parts, which are as follows:

- Status bar
- App bar
- Content area
- Bottom navigation bar

The Android UI components comprise different types of layouts and special interfaces such as menus, notifications, and dialogs.

**Android UI Controls**

Various types of UI controls are available in android to implement the user interface for any android application.

**TextView:** It displays the text to the user on the display screen. It entered text can be edited. Although text editing is allowed, the basic class does not allow editing.

**EditText**: It allows users to enter some text. It also contains certain features through which confidential data can be kept hidden.

**Button:** It allows users to perform some action as soon as some events such as user click, double click takes place.

**ImageButton:** ImageButton is the same as a button but it carries an image on it to perform an action. In this, we need to give the source of the image so that the system can load it.

**ToggleButton:** It displays ON/OFF state of a button with a light indicator.

**RadioButton:** It has two states, checked or unchecked. Users can select only one radiobutton at a time from a group.

**RadioGroup:** A group of Radio buttons that are of similar type where only one of all the radiobuttons can be chosen.

**CheckBox:** It has two states, checked or unchecked. Users can select multiple checkboxes at a time.

**ProgressBar:** It shows the progress of certain action that is proceeding like copying a file from one location to other or downloading a file.

Two modes of progressbar:

- **Determinate Mode**: The progress is shown in terms of percentage of action completed and estimated time to complete the action is also shown.
- **Indeterminate Mode**: It does not show process completion time on screen and therefore it goes continuously.

**Spinner:** It is a dropdown menu and used to select a particular choice from a list of given options. When the down arrow is clicked, a list of values are displayed on screen and it allows us a faster selection of any choice from those options.

**TimePicker:** It helps us to select a time of the day in 12 hrs or 24 hrs format. It gives a virtual Clock or a watch to select it.

**DatePicker:** In gives a virtual calendar or dropdown to select the day or date and a time.

**SeekBar:** It is considered as an extension of a Progress bar. SeekBar has a draggable pointer to drag on the left or right which helps to set the progress or to choose a particular range of values.

**RatingBar:** It is considered as an extended version of a seekbar. It allows us to give a rating by touching it. A user can rate on a scale of 5 with a difference of 0.5 and rating is done in stars.

**AlertDialog**: It is a dialogbox that gives alert or warning to the users. Once it appears on the screen, the user needs to choose an option shown on the screen. For example, when you enter the wrong password for email id or memory full message or wrong name identified for opening a file or folder.

**Switch:** A switch holds either an ON or OFF state. ON means Yes and OFF means No. A user can alter its state multiple times.

**AutoCompleteTextView:** It is an extension of EditText where the user is given with a few suggestions of some values/texts and any value can be selected by the user while filling AutoCompleteTextView.

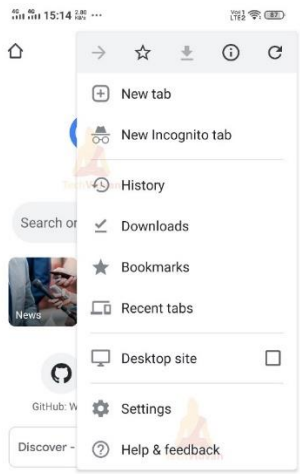
## 2.2 MENU

In android, Menu is an essential component of the User Interface that helps us in providing common functionalities around the application. Menu enhances rich user interaction experience throughout the application. To create and use a menu it needs to be define in a separate XML file and use the same in the android application. Also menu APIs can be used to represent user actions and several other options during android application activities.

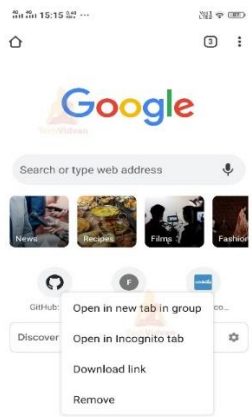
**Types of Menus**

Android provides three types of Menus to define a set of options and actions in the android applications.

**Android Options Menu :** It is a primary collection of menu items and is useful for actions used for searching in application. The options menu is usually present on the action bar where you can see several options listed in the options menu from which you can make a selection.




**Android Context Menu:** It is a floating menu that appears only when the user long press over an element. Context menu is useful for components that affect the selected content or context frame. It is floating menu that means its position is not fixed and usually appears just beside the element you click for a long time on an element.



**Android Popup Menu** : A Popup menu is a list of items displayed vertically and is used to provide actions on the screen specific to related content. It presents the view that invokes the menu. Popup Menu always appears over the view and covers your view.


**Menu designing in XML**

In android studio, Menu needs to be defined in standard XML format. This XML menu resource can be loaded as a menu object in any activity or fragment used in android application.

- Right click on the res folder to create menu directory.
- Navigate to New and select Android Resource Directory.
- Give the name to resource directory as menu
- Select Resource type as menu
- One directory will be created under the res folder.
- Right click on menu folder
- Navigate to New and select menu resource file
- Give the name to resource file as menu_example.
- One menu_example.xml file will be created under menu folder.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http:// schemas.android.com/apk/res/android">
   <item android:id="@+id/coffee"
      android:icon="@drawable/ic_coffee"
      android:title="@string/coffee" />
   <item android:id="@+id/tea"
      android:icon="@drawable/ic_tea"
```

```
      android:title="@string/tea"
   <item android:id="@+id/soda"
      android:icon="@drawable/soda"
      android:title="@string/soda" />

</menu>
```

**Description**

**<menu>** - This root element helps to define Menu in an XML file that contains multiple elements.

**<item>** - It contains nested <menu> elements to create a submenu and helps in creating a single item in the menu.

**<group>** - It is optional and invisible. It is used for <item> elements to categorize the menu items so that they can share properties like active state, and visibility.

## 2.3 SCREEN NAVIGATION

In Android, any activity represents a single screen. But the most of the applications have multiple activities to represent various screens, for example, one activity presents a list of the application settings, and other might displays the application status.

The up navigation allows any application to move to one level up i.e. to the previous activity. To implement it, we need to first find out which activity is the appropriate parent for every activity. It can done by specifying parentActivityName attribute in an activity.

android:parentActivityName = "com.example.test.MainActivity"

Then, we need to call setDisplayHomeAsUpEnabled method of getActionBar() in the onCreate method of the activity which enables the back button in the top action bar.

getActionBar().setDisplayHomeAsUpEnabled(true);

And finally, we need to override onOptionsItemSelected method. When the user press it, activity receives a call to onOptionsItemSelected(). The ID for the action is android.R.id.home.

```
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case android.R.id.home:
    NavUtils.navigateUpFromSameTask(this);
    return true;
  }
}
```

## 2.4 RECYCLERVIEW

The RecyclerView is a more flexible widget and an enhanced version of GridView and ListView. It is a container for big datasets that can be easily scrolled by restricting on a limited number of views. Also, this widget can be used while using dynamic data collections means collections where elements change runtime depending on network events or user actions.

**Implementation of RecyclerView**

1. Plan how the list or grid is going to appear
2. We can use any of the RecyclerView library's standard layout managers.
3. Design the appearance of each element of the list according to how it is going to look and behave.
4. Based on this design, extend the ViewHolder class.
5. This ViewHolder class will provide the functionalities for list items.
6. ViewHolder is a wrapper around a View that is managed by RecyclerView.
7. Adapter is defined to associate data with the ViewHolder views.
8. Three sub parts need to be constructed to implement a basic RecyclerView. Following are these sub parts that offers users the degree of control they require while making various designs of their choice.

**The Card Layout:** It is an XML layout treated as an item for the list created by RecyclerView.

**The ViewHolder:** This class holds the reference to the card layout view that have to be modified dynamically during the program execution by a list of data obtained through either online databases or any other way.

**The Data Class:** This class acts as a structure for storing the information for every item of the RecyclerView.

## 2.5 DRAWABLES

A Drawable resource is a concept in android used for a graphic that can be drawn on the screen. It can retrieved with APIs like getDrawable() or apply to another XML resource with attributes such as android:drawable and android:icon. Example :- Graphical file can be represented via a BitmapDrawable class. Each Drawable is kept as individual files in res/drawable folders. Bitmaps can be stored in the form of following resolutions :- -mdpi, -hdpi, -xhdpi, -xxhdpi. These are subfolders of res/drawable and created by default while creating a project in android studio.

Android system selects the correct bitmap automatically even if these bitmaps are provided in various folders based on the device configuration. In case bitmaps are not provided for all supported resolutions, an Android system balances the closest fit up or down that is typically undesired as the bitmap might get blurred.

There are several types of drawable resource files as follows:

1. **Shape Drawables**

   Shape Drawables are XML files that allow us to define a geometric object with borders, colors, and gradients which can be assigned to Views. The advantage of using it is that they automatically adjust to the correct size.

```
<?xml version="1.0" encoding="UTF-8"?>
<shape
 xmlns:android="http://schemas.android.com/apk/res/android"
android:shape="rectangle">
<stroke
android:width="4dp"
android:color="#FFFFFFFF" />
<gradient
android:endColor="#DDBBBCCC"
android:startColor="#DD777888"
android:angle="90" />
<corners
android:bottomRightRadius="7dp"
android:bottomLeftRadius="5dp"
android:topLeftRadius="7dp"
android:topRightRadius="5dp" />
</shape>
```

2. **State Drawables**

   State drawables allow us to define various states. For each state, a drawable can be assigned to the View. The following example defines an assigned drawable for a button depending on its state.

```
<?xml version="1.0" encoding="utf-8"?>
<selector
xmlns:android="http://schemas.android.com/apk/res/android">
<item android:drawable="@drawable/button_pressed"
android:state_pressed="true" />
<item android:drawable="@drawable/button_checked"
android:state_checked="true" />
</selector>
```

### 3. Transition Drawables

Transition Drawables define transitions that can be triggered in the coding.

```
<?xml version="1.0" encoding="utf-8"?>
<transition
xmlns:android="http://schemas.android.com/apk/res/android">
<item android:drawable="@drawable/one_image" />
 <item android:drawable="@drawable/two_image" />
</transition>
```

### 5. Vector drawables

Vector drawables are similar to svg files but it has a limited scope. Vector drawables automatically fit to the density of any device. Android supports animated vector drawables by using the AnimatedVectorDrawable class. It allows users to combine vector drawables and animations. Vector drawables create and morph various images. We can start and stop this morphing via code but controlling or halting the animation at a particular frame is not possible.

```
<vector
xmlns:android="http://schemas.android.com/apk/res/android"
android:height="32dp"
android:width="32dp"
android:viewportHeight="500"
android:viewportWidth="500" >
<group
android:name="rotationGroup"
android:pivotX="300.0"
android:pivotY="300.0"
android:rotation="60.0" >
<path
   android:name="v"
   android:fillColor="#0000FF"
   android:pathData="M300,70 l 0,-70 70,70 0,0 -70,70z" />
</group>
</vector>
```

### 6. Animation Drawables

Users can also define animation drawables and assign it to a View via the setBackgroundResource() method.

```
<animation-list android:id="@+id/selected"
android:oneshot="false">
```

```
<item android:drawable="@drawable/phase1" android:duration="400" />
<item android:drawable="@drawable/phase2" android:duration="400" />
</animation-list>
```

7. **9 Patch Drawables**

   9 Patch drawables have a one-pixel additional border. From the top and left we can define the area which can be scaled if the Drawable is very small for the view. It is the stretch area. On the right and bottom sides also we can define the area where a text could be placed.

8. **Custom Drawables**

   Users can also create custom Drawable, which can use the Canvas API for their display to design them as per the user's need.

## 2.6 THEMES AND STYLES

Style is a set of attributes in android that can be applied to view elements. Styles has various parameters like font, color, background, margin, text size, text style, etc. Using these properties, we can define our own style and apply it to a UI component or to the entire layout.

Themes are a standardized type of style followed throughout the application. View, non-view elements or layout follows the same theme throughout. When the theme is applied, every view applies each of the theme's attributes that it supports.

### Defining Styles

A style is defined in an XML resource under **res/values/**directory of your project and will have **<resources>** as the root node which is mandatory for the style file. We can define as many styles as we want using **<style>** tag but each style should have unique name that identifies the style.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="CustomFontStyle">
      <item name="android:layout_width">fill_parent</item>
      <item name="android:layout_height">wrap_content</item>
      <item name="android:capitalize">characters</item>
      <item name="android:typeface">monospace</item>
      <item name="android:textSize">12pt</item>
      <item name="android:textColor">#00FF00</item>/>
   </style>
</resources>
```

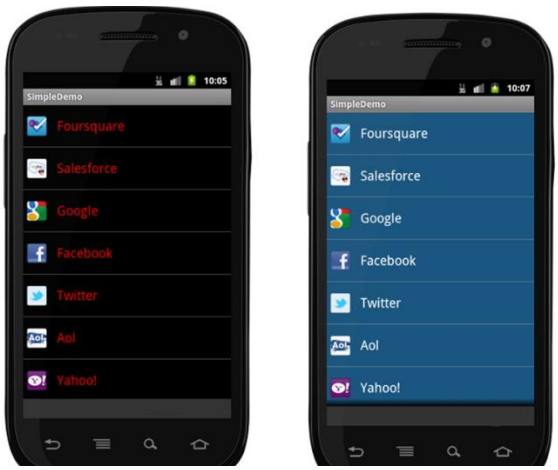
### Using Styles

```
<?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_id"
      style="@style/CustomFontStyle"
      android:text="@string/hello_world" />

</LinearLayout>
```

### Style inheritance

Styles can be inherited in android as similar as cascading style sheets in html. You can inherit properties from an existing style and then define only the properties that we want to change or add.

```
<style name="TextviewStyle" parent="@android:style/Widget.TextView">
    <item name="android:textColor">#86AD33</item>
    <item name="android:textStyle">bold</item>
    <item name="android:textSize">20dp</item>
</style>
```




### Themes vs Styles

In Android, styles and themes allow us to separate the details of the app design from UI behaviour and structure. Styles and themes are defined in a style resource file in res/values/ named styles.xml.

Though themes and styles have many similarities, they are used for different purposes. Themes and styles have the similar basic structure—a key-value pair that maps *attributes* to *resources*.

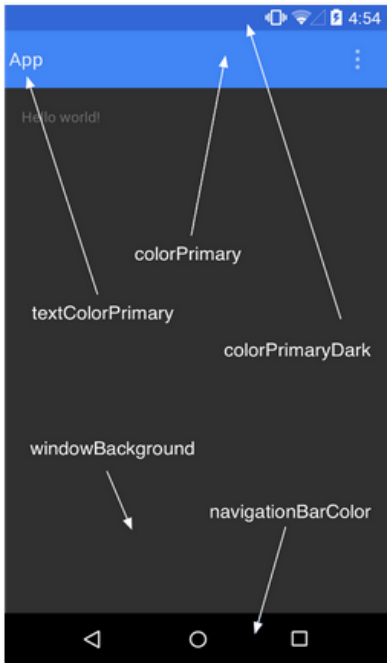
A *style* states attributes for a particular type of view. For an example, style might be stated as any button's attributes. Every specified attribute of button is an attribute that need to be set in the layout file. Extracting all the attributes of a style makes it easier to use and maintain them across multiple widgets.

A *theme* defines a collection of named resources referenced by styles, layouts, widgets, and so on. Themes can assign semantic names, like colorPrimary, to Android resources.

Styles and themes can work together. For example, a style that specifies one part of a button - colorPrimary, and another part - colorSecondary and the actual definitions of those colors can be defined in the theme. When the device goes into night mode, app can switch from "light" theme to "dark" theme. There is no need to change the styles as styles uses semantic names and not specific color definitions. For example, a style can be defined to specify a certain text size and color, then apply it to instances of a certain type of View element. A theme is a set of one or more formatting attributes that can be applied as a unit to all activities in an application. For example, a theme can be defined that sets specific colors for the window frame, foreground and background of the panel, text sizes and colors for menus and then apply it to the activities of the application.

**Defining themes**

```
<color name="custom_theme_color">#b0b0ff</color>
<style name="CustomTheme" parent="Theme.AppCompat.Light">
    <item name="android:windowBackground">@color/custom_theme_color</item>
    <item name="android:colorBackground">@color/custom_theme_color</item>
</style>
```



## 2.7 MATERIAL DESIGN

In android, Material Design Components (MDC) helps developers to implement Material Design in their Android application. MDC is a special type of design that are guided by UX designers and a team of engineers at Google. These components allow reliable development workflow to develop functional Android applications. Material design is the key feature that attracts and connects the customer to the application.

The following are the basic things that need to be considered before material designing:

### 1. Colors and Theming

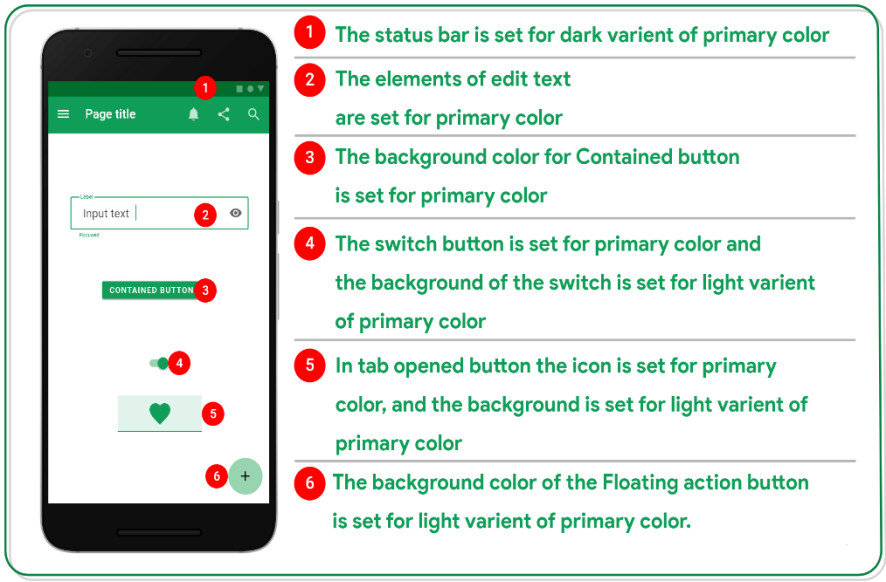
Choosing the right type of color combination signifies the application's brand and style. For example, if main or primary color of an application is the Red, then throughout the application, the red color will be frequently shown.

There are three types of colors to be chosen for developing the android application:-

**Primary Color**: Primary color needs to be chosen very carefully because this color is frequently visible in the application components like high emphasis buttons and also on the top and bottom navigation bar.

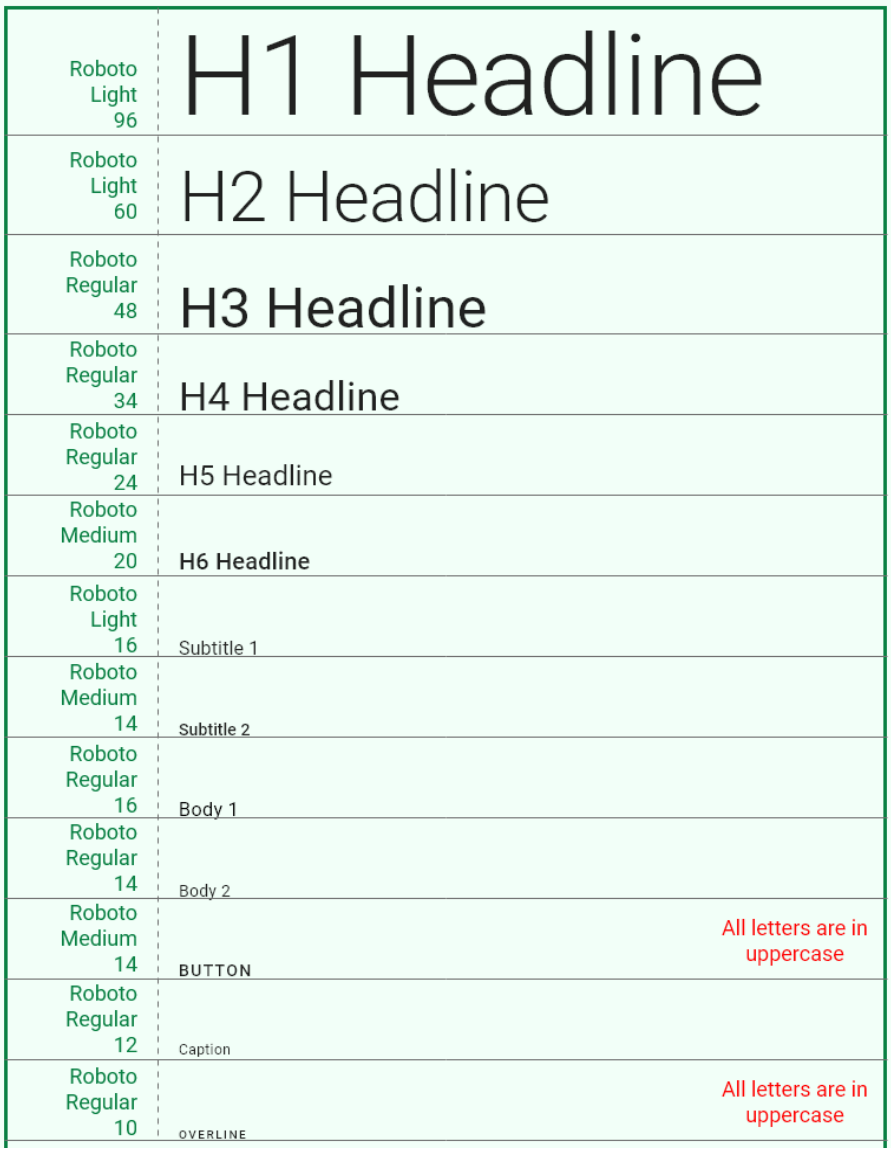
**Secondary Color**: Secondary color needs to be chosen only when there is a low complexity level of the application. This color will be applied to the elements where it needs a little color accent like the background color for the sliders, Floating Action Buttons, toggle buttons, progress bars, etc.

**Light and Dark variants**: These are the variations of the primary color. The dark variant of the primary color is set for the status bar and the light variant of the primary color is set for the Floating action button, outline for the edit texts.



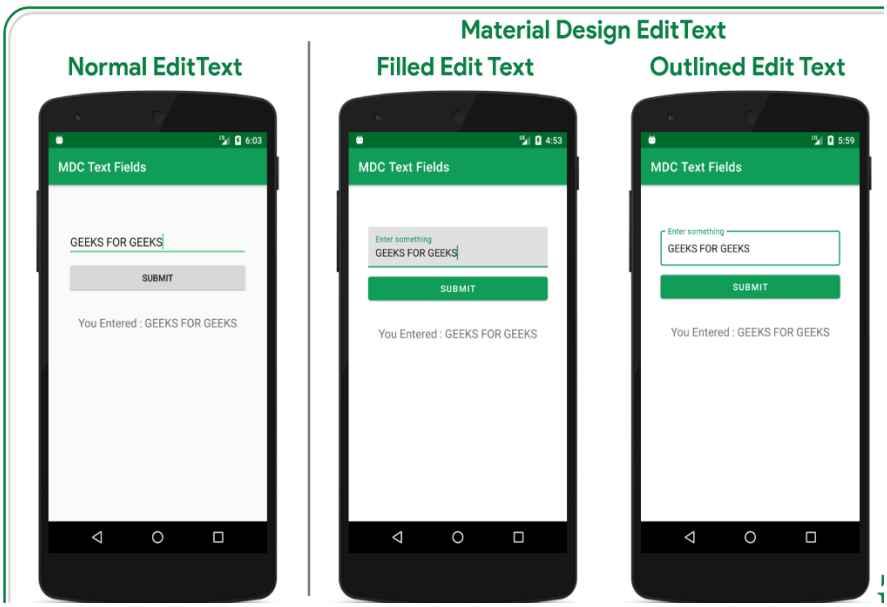
## 2. Typography (Choosing the Right Font)

In android, Roboto font meets all the requirements. But the developer has choice to customize the application with the different font, the font needs to be chosen where it has all its variants like light face, regular face, medium face, and sometimes the dark face. Google offers a variety of font families with all the variants so choosing the font from Google font is suggested. Some guidelines need to be followed for choosing the font. The various contexts of the font are captions, Body, Subtitles, Button, captions, etc.




## 3. Material Design Components

Material design components are the components that allow a lot of features for the users and easy implementation for the developers. These components are noticeable in as compare to normal UI components with respect to style, customization and look. Even these components can adapt dark theme and change styles when it is toggled by the user.




### 4. Shaping the Components

In material design, there are three types of shaping methods :- Cut corner, Rounded corner and Triangle edge.


These methods can be applied for the material design buttons, text fields, floating action buttons, navigation bars, etc. We need to add the dependency and start implementing styling for material design components.

## 2.8 PROVIDING RESOURCES FOR ADAPTIVE LAYOUTS

A layout that keeps changing automatically well with different screen sizes and orientations, different devices, different languages, and different versions of Android is called as an adaptive layout.

**Externalizing resources**

When we externalize resources, we keep them separate from the application code. For example, we can name the string and add it to the res/values/strings.xml file.

**Grouping resources**

Grouping resources means storing and organizing resources in the res/ folder by type by using standardized names for these folders. For example

Android studio Project contains various folders named as - drawable, layout, menu, mipmap, values.

**Alternative resources**

Many applications provide *alternative resources* for supporting specific device configurations. For example, app should include alternative drawable resources for various screen densities, and different languages. At runtime, Android identifies the current device configuration and the appropriate resources are loaded.

**Providing default resources**

*Default resources* identify default designs and content for an application. For example, when the app executes in a locale, Android loads default strings from res/values/strings.xml. If this file is missing some string, then app shows an error while executing.

## 2.9 SUMMARY

Chapter I focuses on various types and uses of user input controls, menus, and drawable. Also, it briefs about the need for resources for adapting numerous layouts on Android screens. It also focuses on the importance of screen navigation, RecyclerView, and Material Design for enhancing app performance. It also explains how the use of style and theme helps to improve the app interface.

## 2.10 EXERCISE

1. Explain various types of Android UI Controls.
2. Discuss types of menus used in Android.
3. Write a note on RecyclerView.
4. What is Drawable and explain various types of it?
5. Differentiate between styles and themes.
6. What are the basic things to be considered before material design?

## 2.11 REFERENCES

1. "Beginning Android 4 Application Development", Wei-Meng Lee, March 2012, WROX.
2. https://developers.google.com/training/courses/android-fundamentals
3. https://www.gitbook.com/book/google-developer-training/android-developer-fundamentals-course-practicals/details

❄❄❄❄❄❄❄

# 3

# DATA TRANSFER AND MANAGEMENT

**Unit Structure :**



## 3.0 OBJECTIVES

After going through this chapter you will be able to

1. Know the functioning of AsyncTaskLoader
2. Know the functions of broadcast receiver, services, notifications and alarm managers
3. Know efficient data transfer
4. Know getting connected with internet

## 3.1 ASYNCTASK AND ASYNCTASKLOADER

In android, there are various ways to perform background processing. Two of those ways are:

Using AsyncTask, background processing can be done directly. Using AsyncTaskLoader, background processing can be done indirectly.

AsyncTaskLoader performs an asynchronous task in the background of any application, so that the user can interact with that application while processing. Once the process is completed, the result will be updated to the interface.

When the configuration of the device changes, AsyncTask and AsyncTaskLoader behaves differently. For example, when the user rotates the screen, activity is destroyed and recreated. AsyncTask reexecutes and a new Thread gets created, however the old Thread becomes separated and uncontrolled. AsyncTaskLoader is reused depending on the Loader ID that

is already registered with the LoaderManager class and that avoids duplication of background tasks, and prevents creation of useless tasks.

Android AsyncTask performs background operation on background thread and updates in main thread. AsyncTask allows communication between background thread and main thread.

While executing AsyncTask follows these four steps:

**onPreExecute()** − Before executing background operations screen should show something on screen like progressbar or animation. It is invoked on UI thread before execution of any task.

**doInBackground(Params)** – It is executed immediately once onPreExecute() finishes. Background operations are performed on background thread but these operations should not include any main thread activities.

**onProgressUpdate(Progress…)** – It is used when updating some information on UI while doing some background operations. It is used to inform the progress of UI thread while background process is executing.

**onPostExecute(Result)** – This method executes on UI thread once the background process is finished.

**Limitations of AsyncTask**

**Changes to device configuration cause problems –** While an AsyncTask is running if device configuration changes, the activity that created the AsyncTask is destroyed and re-created. AsyncTask can not access newly generated activity, and the results cannot be published too. Old AsyncTask objects stay around, and your app may run out of memory or crash. AsyncTask can not be destroyed even if the activity that created it is destroyed.

**When can we use AsyncTask?**

- Short or interruptible tasks.
- Tasks that don't need to report back to UI or user.
- Low-priority tasks that can be left unfinished.

## 3.2 CONNECTING TO THE INTERNET

Android allows any application to connect to the internet or any other local network to perform network operations. Any device can have numerous types of network connections.

To check for network connection or internet android provides ConnectivityManager class. An object of this class need to be instantiated by calling getSystemService() method.

ConnectivityManager check = (ConnectivityManager)

this.context.getSystemService(Context.CONNECTIVITY_SERVICE);

After instantiating the object, to gather the information of all the networks getAllNetworkInfo() method can be used. This method provides an array of NetworkInfo.

```
NetworkInfo[] info = check.getAllNetworkInfo();
```

Now we need to check Connected State of the network.

```
for (int i = 0; i<info.length; i++){
  if (info[i].getState() == NetworkInfo.State.CONNECTED){
    Toast.makeText(context, "Internet is connected
    Toast.LENGTH_SHORT).show();
  }
}
```

**States of a network**

1 Connecting

2 Disconnected

3 Disconnecting

4 Suspended

5 Unknown

## 3.3 BROADCAST RECEIVERS

In android, Broadcast means occurrence of system generated events such as starting of device, recieving of a message or incoming calls, or when airplane mode is on for a device, etc. Broadcast Receivers are used to respond for such system generated events. We can register system and application generated events. When these events take place register receivers get notified.

**Types of Broadcast Receivers:**

**Static Broadcast Receivers:** They are declared in the manifest file and works even if the app is closed.

**Dynamic Broadcast Receivers:** They work only if the app is active or minimized.

**System-wide generated intents**

**android.intent.action.BATTERY_LOW** - Indicates low battery condition on the device.

**android.intent.action.BOOT_COMPLETED** - This is broadcast once after the system has finished booting

**android.intent.action.CALL** - To perform a call to someone specified by the data

**android.intent.action.DATE_CHANGED** - Indicates that the date has changed

**android.intent.action.REBOOT** - Indicates that the device has been a reboot

**android.net.conn.CONNECTIVITY_CHANGE** - The mobile network or wifi connection is changed

**android.intent.ACTION_AIRPLANE_MODE_CHANGED** - This indicates that airplane mode has been switched on or off.

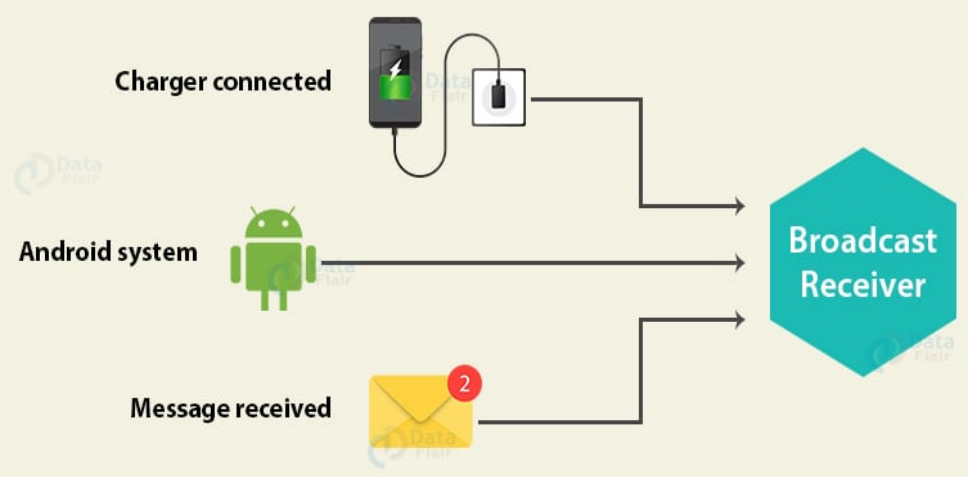
**android.intent.action.POWER_CONNECTED** – It indicates that the power is connected to the device.

**Creating the Broadcast Receiver:**

```
class AirplaneModeChangeReceiver:BroadcastReceiver() {

     override fun onReceive(context: Context?, intent: Intent?) {

    }

}
```

**Registering a BroadcastReceiver:**

```
IntentFilter(Intent.ACTION_AIRPLANE_MODE_CHANGED).also {

                registerReceiver(receiver,it)

    }
```




Broadcast Receiver in Android is a component used to broadcast the messages to the system or any applications. Broadcast receivers have no user interface. It's used for Asynchronous Inter-Process communication. Examples of Broadcast receiver – low battery notification, notification when something downloads.

**Types of broadcast receivers**

**1. Ordered Broadcasts**

Ordered Broadcasts are also known as synchronous broadcasts, and are done in proper order and the priority is decided by android:priority attribute. The broadcast with the highest priority would execute first and broadcasts with the same priority would not follow any order. In ordered broadcast, one broadcast is delivered only to one receiver at a time. When receiver receives a broadcast it is up to the receiver to pass or abort the broadcast. If receiver wants, it passes the broadcast to the next receiver or else the broadcast does not reach the next receiver.

**2. Normal Broadcasts**

Normal broadcasts are known as asynchronous or unordered broadcasts. It executes unorderly or all at a time. They are efficient, but lack full utilization of the results. Normal broadcasts are sent using Context:sendBroadcast. In normal broadcast, it is possible for the system to send only one broadcast at a time to avoid overhead.

## 3.4 SERVICES

In android, services are a special component that enables an application to execute in the background to perform lengthy operational tasks. The main aim of a service is to ensure that the application remains active in the background to operate multiple applications at the same time. A user interaction is not needed as it is designed to operate lengthy processes without user intervention. A service can be executed continuously in background even if the application is closed or user switches to another application. Application components can be connected to itself or other components to perform inter-process communication(IPC).

**Types of Android Services**

**1. Foreground Services**

Foreground Services are services that notify the user about ongoing operations. These services are visible to the users. Users interact with the service easily by providing notifications about ongoing task and try what is happening. These services continue to run even when users are using other applications. While downloading a file, the user can keep track of the progress and can also pause and resume the downloading. Another example is a music player.

**2. Background Services**

Background services execute in the background that doesn't require any user intervention. Users can't see or access these services. These services don't notify the user about ongoing background tasks and even users cannot access them. Processes such as schedule syncing of data or storing of data are examples of background services. An example is syncing and Storing data.

### 3. Bound Services:

The services that allow components of an application like activity to bound themselves with it are called as bound services. These services perform its task as long as any application component is bound to it. Multiple components are allowed to bind themselves with a service at a time. bindService() method is used to bind an application component with a service

## The Life Cycle of Android Services

### 1. Started Service (Unbounded Service):

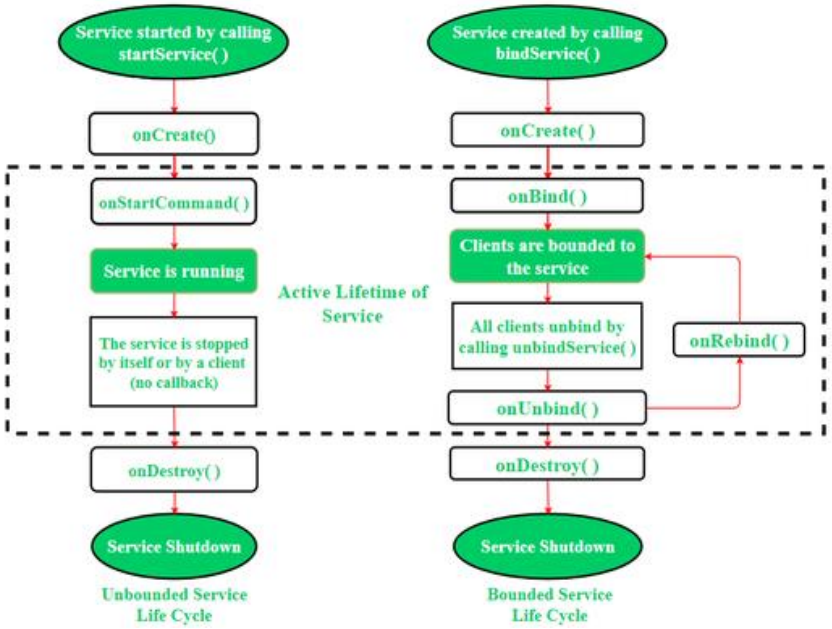
A service is said to be started when startservice() method is called by an application component. A single operation is performed and none of the results are returned. Once service starts, it executes in the background even if a component is destroyed created the service.

Service can be stopped only in one of the following cases:

- By using the stopService() method.
- By stopping itself using the stopSelf() method.

### 2. Bounded Service:

Android application components send requests to the service and fetches results. Bound service executes till some application component is bound to it. Many components can bind to one service at a time, but when they all unbind, the service is destroyed automatically. A Service is said to be bound only when an application component binds itself to a service by using bindService() method. It helps in creating a client-server relation that allows components to interact with any service. These services executes in the background as long as other applications bound to it. To stop execution of these services, we can use unbindService() method to unbind all components themselvs from the service.



**Methods of Android Services**

Following methods are used to carry out service operations on an application :-

**onStartCommand() -** This method is called when component requests for starting a service using startService() method and once it is started, it can be stopped explicitly using stopService() method.

**onBind() -** This is mandatory and it implements services. It is invoked when an application component calls bindService() method. We need to provide an interface for clients to communicate with the service.

**onUnbind() -** This method is called when all clients get disconnected from a particular service interface provided by the service.

**onRebind() -** This method is called once the clients are disconnected from the particular service interface and it is needed to connect the service with new clients. This method is called after onBind() method.

**onCreate() -** This method is called whenr a service is created using onStartCommand() or onBind() method. This method is needed for a one-time set up.

**onDestroy() -** This method is called when a service is no longer in use. This method is invoked before the service destroys as a final clean up call like clean up of resources like threads, receivers, registered listeners, etc.

**Example of Android Services**

A common example of services is playing of music in the background. Music plays continuously in the background as soon as the user starts the service and it continues even though the user switches to other application. To pause or stop the music the user needs to stop the service explicitly.

## 3.5 NOTIFICATIONS

In android, notification gives short and timely information about the action that has happened in the application even if that particular app is not exeuting. The notification displays the title, icon, and some content text. Notification is a kind of alert for an application that is visible in any of the Android's UI elements. This application might be running in the background but may not be used by the user. It notifies to the user about a process that was initiated by any application or by the user or by the system.

**Create and Send Notifications**

**Step 1 - Create Notification Builder**

We can create notifications using NotificationCompat.Builder.build().

NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)

**Step 2 - Setting Notification Properties**

By using Notification Builder we can set various notification properties like small and large icons, title, priority etc.

**Notification Properties**

The properties of Android notification are set using NotificationCompat.Builder object.

**setSmallIcon():** It sets the icon of notification.

**setContentTitle():** It is used to set the title of notification.

**setContentText():** It is used to set the text message.

**setAutoCancel():** It sets the cancelable property of notification.

**setPriority():** It sets the priority of notification.

```
mBuilder.setSmallIcon(R.drawable.notification_icon);

mBuilder.setContentTitle("Alert!");

mBuilder.setContentText("Memory Full!!!");
```

**Step 3 - Attach Actions**

It is optional. An action allows users to move directly from the notification to any Activity in the application, where they could check for the events or do any further work. If we want to attach any action with any notification, we can define PendingIntent that contains an Intent initiating an Activity for that application and to associate this PendingIntent with gesture, any of the method of NotificationCompat.Builder can be used.

**Step 4 - Issue the notification**

Finally, we can call NotificationManager.notify() and pass Notification object to the system. We have to make sure that any method of NotificationCompat.Builder.build() to be called before notifying it.

```
NotificationManager mNotificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

mNotificationManager.notify(notificationID, mBuilder.build());
```

## 3.6 ALARM MANAGERS

AlarmManager class provides access to the alarm services of your system. AlarmManager helps us to schedule an alarm for a particular time. Also, we can schedule execution of an application at a particular time. The moment the alarm goes off, the system broadcasts the intent registered for it. AlarmManager holds wake lock of the CPU till onReceive() method is in execution so that the device does not sleep until the broadcast is handled. Alarms provides a way to perform time-based operations.

**Characteristics of an alarm:**

- Alarms sets off intents at a particular time or at certain time intervals.
- Alarms can be executed outside the application also, it allows the alarm to trigger even when the application is not executing.
- Alarms can be used with broadcast receivers to perform a particular action or to start a service
- Alarms minimize resource requirements.
- Alarms execute till they are force stopped.

**How to set an alarm?**

We first need to create object of AlarmManager class and then pass it to pending intents

AlarmManager am = (AlarmManager) getSystemService( Context.ALARM_SERVICE);

Intent ai = new Intent(context, alarmreceiver.class);

PendingIntent pi = PendingIntent.getBroadcast(context, 0 , ai, 0);

Int interval = 8000;

manager. setInexactRepeating( AlarmManager/RTC_WAKEUP, System.currentTimeMillis(), interval, pendingIntent);

**How to invoke AlarmManager**

**setInExactAndRepeating**: It does not trigger the alarm at the exact time.

**setExact:** It ensures that system triggers alarm on exact time.

**setExactAndAllowWhileIdle**: It is allowed to be executed, even in low power modes of devices.

## 3.7 TRANSFERRING DATA EFFICIENTLY

Data transferring is an essential part of android applications and sometimes it affects battery life and rises data usage costs. One of the app's most significant sources of battery drain is using the wireless radio to transfer data. A fully active wireless radio consumes more power. For a 3G network the radio has the following three energy states:

**Full power:** It is used when a connection is active and allows devices to transfer data at the highest possible rate.

**Low power:** An intermediate state that uses about 50% less battery.

**Standby:** The minimal energy state where no active network connection is required.

Android uses a state machine to determine how to transition between various states. To reduce the latency, a state machine waits for a shorter time before it transits to the lower energy states. The radio state machine on each device is associated with transition delay and startup latency, based on the wireless radio technology (2G, 3G, LTE, etc.). It is defined and configured by the carrier network where the device is operating.

**Best practices to keep in mind while developing an app.**

**Bundling network transfers**

Every time a new network connection is created, the radio transitions to the full power state. In case of 3G radio state machine, it takes full power for the transfer duration. For a typical 3G device, every data transfer session produces the radio to fetch power for almost 20 seconds. It is important to bundle and queue up data transfers. We can bundle transfers that can occur within a certain time and make it all execute simultaneously. It ensures radio draws power for minimum time.

**Prefetching**

Prefetching means applications guess the content or data the user wants next, and fetch it before time. For example, when the user is watching a video he can fetch the next part of the video. Data prefetching is an effective way to minimize the number of independent data transfers. Prefetching allows us to download data we might need for a given time period in a single burst within a single connection with full capacity. It minimizes the number of radio activations that are required for data downloading. So, we can conserve battery life as well as enhance latency for the user, minimize the required bandwidth, and decrease download time.

**Devices can network using different types of hardware:**

Wireless radios use certain amounts of battery depending on technology use and higher bandwidth consumes high energy. Here, higher bandwidth means we can prefetch more data during the same amount of time. WiFi radio uses less battery than wireless and provides greater bandwidth.

**Monitor battery state**

To reduce battery consumption, we need to monitor the state of a battery and wait for certain conditions before initiating battery-specific operations. BatteryManager broadcasts details of battery and charging to the Intent that includes the charging status.

**JobScheduler**

Keeping track of the connectivity and battery status regularly can be a challenge, and it requires use of components like broadcast receivers, which consumes system resources even when application is stopped. JobScheduler is a class provided by android SDK which allows us to transfer data efficiently.

## 3.8 SUMMARY

Chapter II focuses on various types of broadcast receivers and services. Also, it briefs about the benefits of AsyncTaskLoader in android. It also focuses on how to create notifications and how to use alarm manager for enhancing app performance. It also explains how to transfer data efficiently and connecting to internet.

## 3.9 EXERCISE

1. Explain the role of broadcast receiver in android.
2. Discuss the types of broadcast receivers.
3. Write a note on AsyncTaskLoader.
4. What is services in android and explain the various types of it?
5. State and explain the life cycle of android services with a neat diagram.
6. What do you mean by notification in android and describe steps to create and send notifications?
7. Write a note on alarm manager.
8. Explain the best practices to keep in mind while developing an app for efficient data transfer.

## 3.10 REFERENCES

1. "Beginning Android 4 Application Development", Wei-Meng Lee, March 2012, WROX.
2. https://developers.google.com/training/courses/android-fundamentals
3. https://www.gitbook.com/book/google-developer-training/android-developer-fundamentals-course-practicals/details

❄❄❄❄❄❄❄

# 4

# DATA - SAVING, RETRIEVING AND LOADING

**Unit Structure :**



## 4.0 OBJECTIVES

This chapter will help us to understand how data can be saved, retrieved and loaded using various databases in Android.

## 4.1 INTRODUCTION

Android provides variety of SharedPreferences object to help you save simple application data. For example, your application may have an option to allow users to specify the font size of the text displayed in your application. In this case, your application needs to remember the size set by the user so that the next time he or she uses the application again, your application can set the size appropriately. In order to do so, you have several options. You can save the data to a file, but you must perform some file management routines, such as writing the data to the file, indicating how many characters to read from it, and so on.

An alternative to writing to a text file is to use a database, but saving simple data to a database is over- kill, both from a developer's point of view and in terms of the application's run-time performance.

## 4.2 SAVING AND LOADING USER PREFERENCES

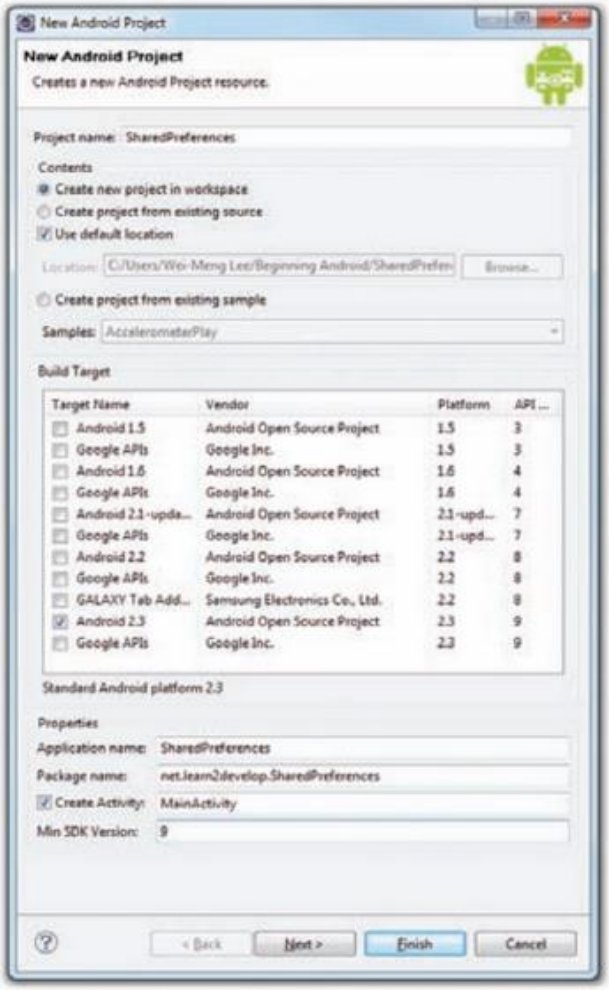
Using the SharedPreferences object, however, you save the data you want through the use of key/value pairs — specify a key for the data you want to save, and then both it and its value will be saved automatically to an XML file for you.

**Method:**

getSharedPreferences()

For creating shared preferences follow the following steps

**1.Using Eclipse, create an Android project**




**2.Add the following in main.xml file:**

<LinearLayout
xmlns:android=”http://schemas.android.com/apk/res/android”

android:orientation=”vertical”

android:layout_width=”fill_parent”

android:layout_height=”fill_parent” >

```
<SeekBar
android:id=”@+id/SeekBar01”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content” />

<TextView
android:id=”@+id/TextView01”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”@string/hello” />

<EditText
android:id=”@+id/EditText01”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content” />

<Button
android:id=”@+id/btnSave”
android:text=”Save”
android:layout_width=”wrap_content”
android:layout_height=”wrap_content” />
</LinearLayout>
```

**3. In the MainActivity.java file, add the following statements**

```
import android.app.Activity;
import android.view.View;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends Activity {
private EditText textBox;
```

```
private static final int READ_BLOCK_SIZE = 100;
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
textBox = (EditText) findViewById(R.id.txtText1);
Button saveBtn = (Button) findViewById(R.id.btnSave);
Button loadBtn = (Button) findViewById(R.id.btnLoad);
saveBtn.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
String str = textBox.getText().toString();
try
{
FileOutputStream fOut =
openFileOutput("textfile.txt",
MODE_WORLD_READABLE);
OutputStreamWriter osw = new
OutputStreamWriter(fOut);
//---write the string to the file---

osw.write(str);
osw.flush();
osw.close();

//---display file saved message---
Toast.makeText(getBaseContext(),
"File saved successfully!",
Toast.LENGTH_SHORT).show();
//---clears the EditText---
textBox.setText("");
}
catch (IOException ioe)
{
ioe.printStackTrace();
}
}
```

```
});
loadBtn.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
try
{
FileInputStream fIn =
openFileInput("textfile.txt");
InputStreamReader isr = new
InputStreamReader(fIn);
char[] inputBuffer = new char[READ_BLOCK_SIZE];
String s = "";
int charRead;
while ((charRead = isr.read(inputBuffer))>0)
{
//---convert the chars to a String---
String readString =
String.copyValueOf(inputBuffer, 0,
charRead);
s += readString;
inputBuffer = new char[READ_BLOCK_SIZE];
}

//---set the EditText to the text that has been
// read---
textBox.setText(s);
Toast.makeText(getBaseContext(),
"File loaded successfully!",
Toast.LENGTH_SHORT).show();
}
catch (IOException ioe) {
ioe.printStackTrace();
}
}
});
}
}
```

**4.Press F11 to debug the application on the Android Emulator.**

**5. Type some text into the EditText view and then click the Save button.**


6. If the file is saved successfully, you will see the Toast class displaying the "File saved successfully!" message. The text in the EditText view should disappear.

7. Click the Load button and you should see the string appearing in the EditText view again. This confirms that the text is saved correctly.

## 4.3 SAVING TO EXTERNAL STORAGE (SD CARD)

The previous section showed how you can save files to the internal storage of your Android device. Sometimes, it would be useful to save them to external storage because of its larger capacity, as well as the capability to share the files easily with other users

Using the project created in the previous section as the example, to save the text entered by the user in the SD card, modify the onClick() method of the Save button as shown in bold here:

```
saveBtn.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
String str = textBox.getText().toString();
try
{
//---SD Card Storage---

File sdCard = Environment.getExternalStorageDirectory();
```

```
File directory = new File (sdCard.getAbsolutePath() +
"/MyFiles");
directory.mkdirs();
File file = new File(directory, "textfile.txt");
FileOutputStream fOut = new FileOutputStream(file);
OutputStreamWriter osw = new
OutputStreamWriter(fOut);
//---write the string to the file---
osw.write(str);
osw.flush();
osw.close();

//---display file saved message---
Toast.makeText(getBaseContext(),

"File saved successfully!",

Toast.LENGTH_SHORT).show();
//---clears the EditText---
textBox.setText("");
}

catch (IOException ioe)
{
ioe.printStackTrace();
}
}
});
```

## 4.4 FILE AND DATA STORAGE

Android uses file system that is much similar to disk-based file systems on other platforms. There are different options available.

Android is an open-source operating system for mobile devices developed by Google. It is designed to run on smartphones, tablets, and many other mobile devices. Android provides several options for file and data storage, including internal storage, external storage, and cloud-based storage solutions.

**Internal Storage:**

Android devices typically have a built-in storage space, called internal storage, where the system stores apps, data, and other files. The amount of internal storage varies depending on the device size, but it is usually in the range of 16 to 256 GB. Internal storage is non-removable.

**External Storage:**

Many Android devices also have a slot for a microSD card, that helps to provide additional storage space for data and files. External storage is removable, which means that users can insert or remove the microSD card. However, not all devices support external storage, and the amount of storage capacity depends on the type of microSD card used.

**Cloud-based Storage:**

Cloud-based storage solutions, such as Google Drive and Dropbox, are also available on Android devices. These services allow users to store a number of files and data in the cloud, which means that they can access their files from any device with an internet connection. Cloud-based storage also provides an additional layer for backup of files, which can help prevent data loss in the event of a device failure or loss.

## 4.5 HOW MUCH SPACE DOES YOUR DATA REQUIRE?

Internal storage has limited space for app-specific data. Use other types of storage if you need to save a minimum amount of data.

**How reliable does data access need to be?**

App's basic functionality requires certain data, such as when your app is starting up, place the data within internal storage directory or a database. App-specific files that are stored in external storage aren't always accessible because some devices allow users to remove a physical device that corresponds to external storage.

**What kind of data do you need to store?**

If you have data that's only meaningful for your app, use app-specific storage. For shareable media content, use shared storage so that other apps can access the content. For structured data, use either preferences (for key-value data) or a database (for data that contains more than 2 columns).

**Should the data be private to your app?**

When storing sensitive data—data that should not be accessible from any other app—use internal storage, preferences, or a database. Internal storage has the added benefit of the data being hidden from users.

## 4.6 CATEGORIES OF STORAGE LOCATIONS

In Android, there are several categories of storage locations where apps can store data. Each category has its own unique characteristics and is intended for different types of data. The main categories of storage locations in Android are:

**Internal storage:** This is the primary storage location for an app's private data. All apps have access to their own internal storage, which is not accessible to other apps. Internal storage is used for storing app-specific data, such as databases, preferences, and cache files. Apps can also store media files, such as images and videos, in their internal storage.

**External storage:** This refers to any storage location that can be accessed by the user and other apps, such as a microSD card or USB drive. Prior to Android 10, apps could access external storage without any restrictions. However, with the introduction of scoped storage in Android 10, apps must now request permission to access external storage. Scoped storage allows apps to only access their own "scoped" storage area, which is isolated from other apps and the system.

**Network storage:** This refers to data that is stored on remote servers and accessed over the internet. Apps can use network storage to store and retrieve data, such as user profiles, media files, and other content.

**Shared preferences:** This is a lightweight storage mechanism used for storing small amounts of app-specific data, such as user preferences and settings. Shared preferences are stored in XML files and are accessible to all components of an app.

**SQLite databases:** This is a local database storage mechanism used for storing structured data. Apps can use SQLite databases to store and retrieve data, such as user profiles, messages, and other app-specific data.

**Content providers:** This is a mechanism used for sharing data between apps. Content providers are used to manage access to a shared set of app-specific data, such as contacts, calendar events, and other data that can be accessed by multiple apps.

Overall, the different categories of storage locations in Android provide developers with a variety of options for storing and accessing data, depending on the type of data and the app's specific requirements.

On most devices, internal storage is smaller than external storage. However, internal storage is always available on all devices, making it a more reliable place to put data on which your app depends.

## 4.7 PERMISSIONS AND ACCESS TO EXTERNAL STORAGE

Android defines the following storage-related permissions:

1.Read

2.Write

3.Manage

In Android, external storage refers to any storage that can be accessed by the user and other apps, such as a microSD card or USB drive. Prior to Android 10 (API level 29), apps could access external storage without any restrictions. However, with the introduction of scoped storage in Android 10, apps must now request permission to access external storage.

There are two types of permissions that an app can request to access external storage: READ_EXTERNAL_STORAGE and WRITE_EXTERNAL_STORAGE. READ_EXTERNAL_STORAGE allows an app to read files from external storage, while WRITE_EXTERNAL_STORAGE allows an app to write files to external storage. Both of these permissions are considered "dangerous" permissions, which means that apps must request them at runtime and users must grant them explicitly.

To request permission to access external storage, an app must include the necessary permission(s) in its manifest file and request permission(s) at runtime using the Android permission system. If the user grants permission, the app can access external storage using the standard Java I/O APIs.

However, with scoped storage, apps can no longer access all files on external storage without restrictions. Instead, apps can only access files in their own "scoped" storage area, which is isolated from other apps and the system. This means that apps cannot access files in other app's storage areas or in the root of external storage. Scoped storage also introduces a new permission called ACCESS_MEDIA_LOCATION, which grants an app access to the location of media files.

Scoped storage provides better security and privacy for users, but it can also introduce some challenges for app developers. Developers must update their apps to comply with the new mechanism and use new APIs to access files in external storage. They must also consider how scoped storage affects their app's file management and data sharing capabilities.

In summary, permission and access to external storage in Android has evolved with the introduction of scoped storage. While it improves security and privacy, it also requires developers to update their apps to comply with the new mechanism.

## 4.8 SCOPED STORAGE

Scoped storage is a new mechanism introduced in Android 10 (API level 29) to improve user privacy and app security by limiting an app's access to external storage. In the past, Android allowed apps to access external storage without any restrictions, which could potentially lead to security issues.

With scoped storage, an app can only access files in its own "scoped" storage area, which is isolated from other apps and the system. This means that apps cannot access files in other app's storage areas or in the root of external storage. Scoped storage also introduces a new permission called READ_EXTERNAL_STORAGE, which grants an app access to a specific file or directory in external storage.

In addition to enhancing security and privacy, scoped storage also provides better organization and management of files. Each app's scoped storage area is unique, which makes it easier for users to find and manage files related to a specific app. Scoped storage also enforces file metadata updates, which ensures that the system is aware of any changes made to files by apps.

However, scoped storage can also introduce some challenges for app developers, as they need to update their apps to comply with the new mechanism. For example, apps may need to use new APIs to access files in external storage, or request new permissions from users. Developers also need to consider how scoped storage affects their app's file management and data sharing capabilities.

## 4.9 SUMMARY

In this chapter, you learned the different ways to save persistent data to your Android device. For simple unstructured data, using the SharedPreferences object is the ideal solution. If you need to store bulk data, then consider using the traditional file system. Finally, for structured data, it is more efficient to store it in a relational database management system. For this, Android provides the SQLite database, which you can access easily using the APIs exposed.

Note that for the SharedPreferences object and the SQLite database, the data is accessible only by the application that creates it. In other words, it is not shareable. If you need to share data among different applications, you need to create a content provider.

| TOPIC | KEY CONCEPTS |
|---|---|
| **Save simple user data** | Use the `SharedPreferences` object. |
| **Sharing data among activities in the same application** | Use the `getSharedPreferences()` method. |
| **Saving data visible only to the activity that created it** | Use the `getPreferences()` method. |
| **Saving to file** | Use the `FileOutputStream` and `OutputStreamReader` classes. |
| **Reading from file** | Use the `FileInputStream` and `InputStreamReader` classes. |
| **Saving to external storage** | Use the `getExternalStorageDirectory()` method to return the path to the external storage. |
| **Accessing files in the `res/raw` folder** | Use the `openRawResource()` method in the `Resources` object (obtained via the `getResources()` method). |
| **Creating a database helper class** | Extend the `SQLiteOpenHelper` class. |

## 4.10 REFERENCES

1.Wrox. beginning android application development reference book

2.android developers

3.github for codes

## 4.11 UNIT END QUESTION

1. What is the difference between the getSharedPreferences() and getPreferences() methods?
2. Name the method that enables you to obtain the path of the external storage of an Android device.
3. What is the permission you need to declare when writing files to external storage?

❄❄❄❄❄❄❄

# 5

# DATABASE

**Unit Structure :**



## 5.0 OBJECTIVES

This chapter will help you to connect your android application to database i.e SQLite or Firebases, where create, update, insert delete operations can be performed

## 5.1 INTRODUCTION

SQL stores data in tables of rows and columns (spreadsheet…).In database field is intersection of a row and column. Fields contain data, references to other fields, or references to other tables. Rows are identified by unique IDs. Column names are unique per table

## 5.2.CONTENTPROVIDER

In Android, using a content provider is the recommended way to share data across packages. Think of a content provider as a data store. How it stores its data is not relevant to the application using it; what is important is how packages can access the data stored in it using a consistent programming interface. A content provider behaves very much like a database — you can query it, edit its content, as well as add or delete its content. However, unlike a database, a content provider can use different ways to store its data. The data can be stored in a database, in files, or even over a network.

Android ships with many useful content providers, including the following:

1. Browser — Stores data such as browser bookmarks, browser history, and so on
2. CallLog — Stores data such as missed calls, call details, and so on
3. Contacts — Stores contact details
4. MediaStore — Stores media files such as audio, video, and images
5. Settings — Stores the device's settings and preferences

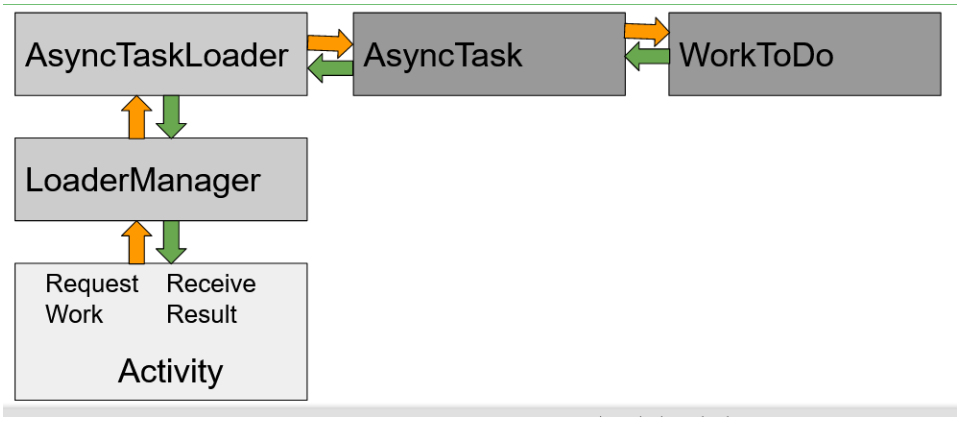
## 5.3 LOADERS

Loaders provide asynchronous loading of data**. It helps to reconnect to Activity after configuration change** .It can monitor changes in a data source and deliver new data. It is usually callbacks implemented in Activity. Different types of loaders are available

1.AsyncTaskLoader

2.CursorLoader

**Loaders can be used to execute tasks of the UI thread. LoaderManager handles configuration changes for the device. It is efficiently implemented by the framework of Android. Users don't have to wait for the loading of data.**




## 5.4 SQLITE DATABASES

activity_main.xml

- MainActivity.kt
- DatabaseHelper.kt

Step 1: activity_main.xml

<?xml version="1.0" encoding="utf-8"?>

```
<LinearLayout
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"
android:orientation="vertical"
tools:context=".MainActivity">
<EditText
android:id="@+id/editID"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:hint="Enter Unique ID"
android:textSize="25sp" />
<EditText
android:id="@+id/editName"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:hint="Enter Your Name"
android:textSize="25sp" />
<EditText
android:id="@+id/editEmail"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:hint="Enter Email Address"
android:textSize="25sp" />
<EditText
android:id="@+id/editCourse"
android:layout_width="match_parent"

android:layout_height="wrap_content"
android:hint="Enter Course Name"
android:textSize="25sp" />
<Button
android:id="@+id/btnInsert"
```

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Insert Data" />
<Button
android:id="@+id/btnUpdate"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Update Data" />
<Button
android:id="@+id/btnDelete"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Delete Data" />
<Button
android:id="@+id/btnViewAll"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="View Data" />
</LinearLayout>
```

Step 2: MainActvity.kt

```
package com.example.db
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import com.example.db.databinding.ActivityMainBinding
class MainActivity : AppCompatActivity() {

private lateinit var binding: ActivityMainBinding
private var dbHelper = DatabaseHelper(this)
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
binding = ActivityMainBinding.inflate(layoutInflater)
```

```
setContentView(binding.root)
handleInserts()
handleUpdates()
handleDeletes()
handleViewing()
}
private fun showToast(text: String) {
Toast.makeText(this, text, Toast.LENGTH_LONG).show()
}
private fun showDialog(title: String, Message: String) {
val builder = AlertDialog.Builder(this)
builder.setCancelable(true)
builder.setTitle(title)
builder.setMessage(Message)
builder.show()
}
private fun clearEditTexts() {
binding.editID.setText("")
binding.editName.setText("")
binding.editEmail.setText("")
binding.editCourse.setText("")
}
private fun handleInserts() {
binding.btnInsert.setOnClickListener {
try {
dbHelper.insertData(
binding.editName.text.toString(),
binding.editEmail.text.toString(),
binding.editCourse.text.toString()
)

clearEditTexts()
showToast("Data Inserted Successfully")
} catch (e: Exception) {
e.printStackTrace()
```

```
showToast(e.message.toString())
}
}
}
private fun handleUpdates() {
binding.btnUpdate.setOnClickListener {
try {
dbHelper.updateData(
binding.editID.text.toString(),
binding.editName.text.toString(),
binding.editEmail.text.toString(),
binding.editCourse.text.toString()
)
clearEditTexts()
showToast("Data Updated Successfully")
} catch (e: Exception) {
e.printStackTrace()
showToast(e.message.toString())
}
}
}
private fun handleDeletes() {
binding.btnDelete.setOnClickListener {
try {
dbHelper.deleteData(binding.editID.text.toString())
clearEditTexts()
showToast("Data Deleted Successfully")
} catch (e: Exception) {
e.printStackTrace()
showToast(e.message.toString())
}
}
}
private fun handleViewing() {
binding.btnViewAll.setOnClickListener(
```

```
View.OnClickListener {
val res = dbHelper.allData
if (res.count == 0) {
showDialog("Error", "No Data Found")
return@OnClickListener
}
val buffer = StringBuffer()
while (res.moveToNext()) {
buffer.append("ID: " + res.getString(0) + "\n")
buffer.append("NAME: " + res.getString(1) + "\n")
buffer.append("EMAIL: " + res.getString(2) + "\n")
buffer.append("COURSE: " + res.getString(3) + "\n\n")
}
showDialog("Data", buffer.toString())
})
}
}
```

**Step3: DatabaseHelper.kt**

```
package com.example.db
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper
class DatabaseHelper(context: Context) : SQLiteOpenHelper(context,
DATABASE_NAME, null, 1) {
override fun onCreate(db: SQLiteDatabase?) {
db?.execSQL("CREATE TABLE $TABLE_NAME (ID INTEGER
PRIMARY
KEY AUTOINCREMENT,NAME TEXT,EMAIL TEXT,COURSE
TEXT)")
```

```
}
override fun onUpgrade(
db: SQLiteDatabase, oldVersion: Int, newVersion: Int
) {
db.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
onCreate(db)
}
fun insertData(name: String, email: String, course: String) {
val db = this.writableDatabase
val contentValues = ContentValues()
contentValues.put(COL_1, name)
contentValues.put(COL_2, email)
contentValues.put(COL_3, course)
db.insert(TABLE_NAME, null, contentValues)
}
fun updateData(
id: String, name: String, email: String, course: String
): Boolean {
val db = this.writableDatabase
val contentValues = ContentValues()
contentValues.put(COL_0, id)
contentValues.put(COL_1, name)
contentValues.put(COL_2, email)
contentValues.put(COL_3, course)
db.update(TABLE_NAME, contentValues, "ID = ?", arrayOf(id))
return true
}
fun deleteData(id: String): Int {
val db = this.writableDatabase
```

```
return db.delete(TABLE_NAME, "ID = ?", arrayOf(id))
}
val allData: Cursor
get() {
val db = this.writableDatabase
return db.rawQuery("SELECT * FROM $TABLE_NAME", null)
}
companion object {
const val DATABASE_NAME = "student.db"
const val TABLE_NAME = "student_table"
const val COL_0 = "ID"
const val COL_1 = "NAME"
const val COL_2 = "EMAIL"
const val COL_3 = "COURSE"
}
}
```

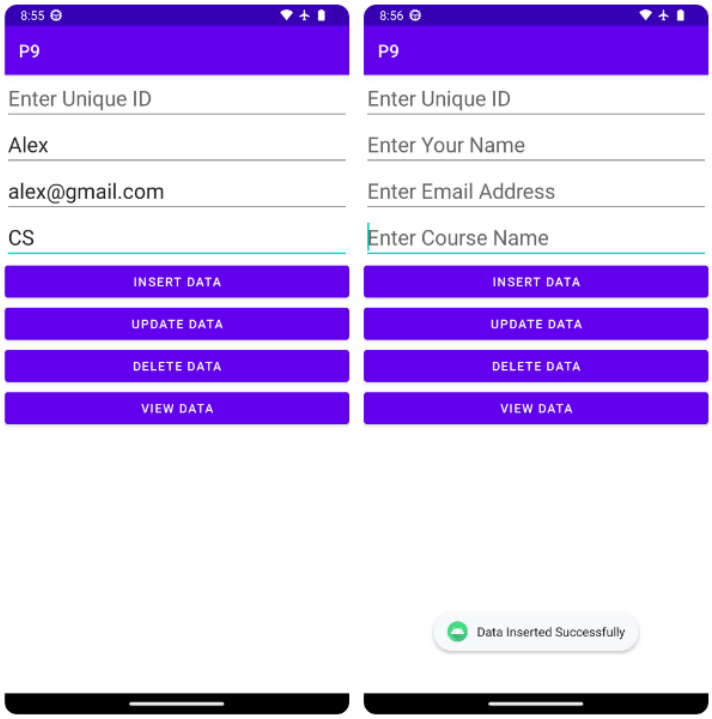
**Step 5:**

build.gradle (Module: app)

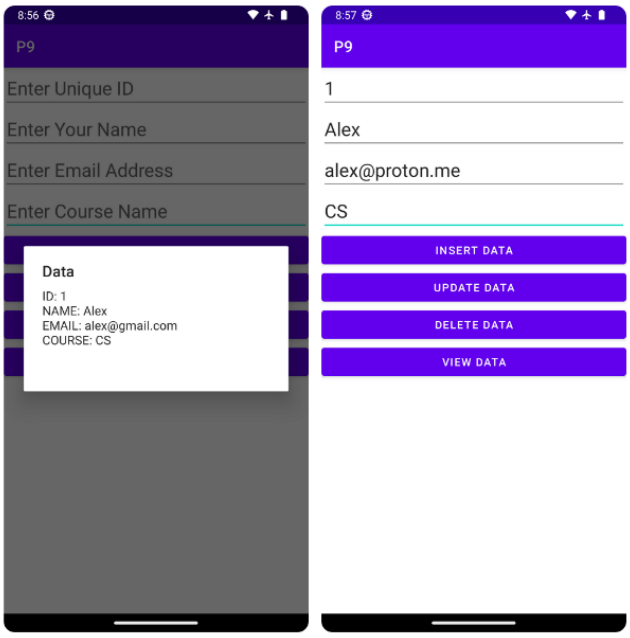
Insert the following after kotlinOptions

```
buildFeatures {
viewBinding = true
}
```

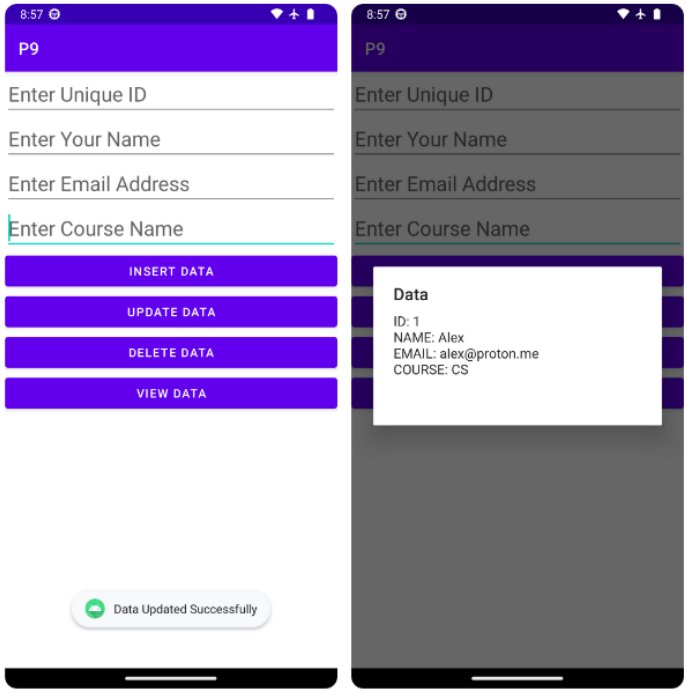
Inserting Data
Data can be inserted into the database using the INSERT DATA button, the Unique
ID does not need to be entered as it is auto-incremented




Viewing Data
The data present in the database can be viewed by clicking the VIEW DATA button,
the stored data is displayed in a dialog box.

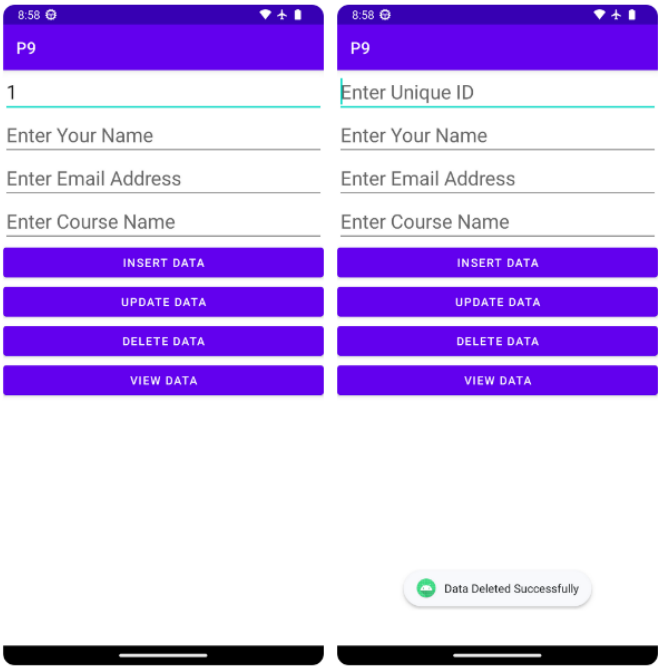



Updating Data
The data stored in the database can be updated using the UPDATE DATA button, all
the fields must be filled in order to update a database record

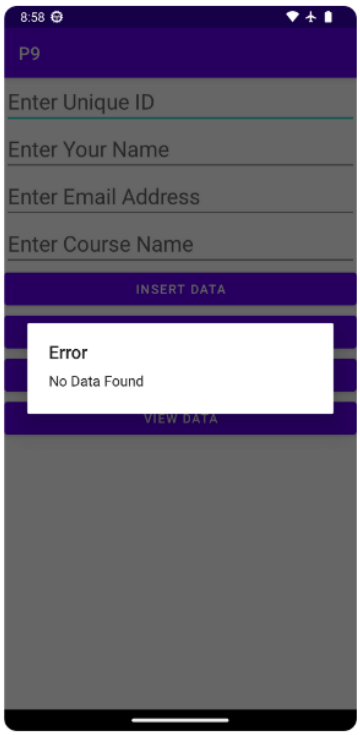



Toast Messages
A toast message is displayed when any operation is carried out on the database, a
toast message is displayed when inserting, updating or deleting records

Deleting Data
The records stored in the database can be deleted using the DELETE DATA button,
only the Unique ID is required to delete a record.



Empty Database
When no records have been stored in the database or if all the records from it were
deleted, an error message is displayed in a dialog box.




## 5.5 ADVANTAGES AND DISADVANTAGES

SQLite is a very popular database that has been successfully used with an on-disk file format for desktop applications like version control systems, financial analysis tools, media cataloging and editing suites, CAD packages, record-keeping programs, etc.

There are a lot of advantages to using SQLite as an application file format:

1) Lightweight

   SQLite is a very light weighted database so, it is easy to use it as an embedded software with devices like televisions, Mobile phones, cameras, home electronic devices, etc.

2) Better Performance

   Reading and writing operations are very fast for SQLite databases. It is almost 35% faster than a File system.

   It only loads the data which is needed, rather than reading the entire file and holding it in memory.

If you edit small parts, it only overwrites the parts of the file which was changed.

3) No Installation is Needed

SQLite is very easy to learn. You don't need to install and configure it. Just download SQLite libraries on your computer and it is ready for creating the database.

4) Reliable

It updates your content continuously so, and little or no work is lost in case of power failure or crash.

SQLite is fewer bugs prone rather than custom-written file I/O codes.

SQLite queries are smaller than equivalent procedural codes so, the chances of bugs are minimal.

5) Portable

SQLite is portable across all 32-bit and 64-bit operating systems and big- and little-endian architectures.

Multiple processes can be attached to the same application file and can read and write without interfering with each other.

It can be used with all programming languages without any compatibility issues.

6) Accessible

SQLite database is accessible through a wide variety of third-party tools.

SQLite database's content is more likely to be recoverable if it has been lost. Data lives longer than code.

7) Reduce Cost and Complexity

It reduces application costs because content can be accessed and updated using concise SQL queries instead of lengthy and error-prone procedural queries.

SQLite can be easily extended in future releases just by adding new tables and/or columns. It also preserves backward compatibility.

**SQLite Disadvantages**

SQLite is used to handle low to medium-traffic HTTP requests.

Database size is restricted to 2GB in most cases.

## 5.6 FIRE BASE DATABASES

Firebase Realtime Database is a cloud-hosted NoSQL database that allows you to store and sync data in real time between multiple clients. It is a popular backend-as-a-service (BaaS) solution provided by Google, which can be used for developing web and mobile applications.Firebase Realtime Database is a cloud-hosted NoSQL database that allows you to store and sync data in real time between multiple clients. It is a popular backend-as-a-service (BaaS) solution provided by Google, which can be used for developing web and mobile applications.

## 5.5 ADVANTAGES AND DISADVANTAGES

Some of the advantages of Firebase include:

**Syncing simultaneously from multiple clients:** Firebase's real-time database allows multiple clients to sync data simultaneously. This means that any changes made to the database are reflected on all connected devices in real time, without the need for manual synchronization.

**NoSQL cloud database:** Firebase's database is a NoSQL database hosted on the cloud. This means that developers do not have to worry about setting up and managing their servers or databases.

**Realtime:** Firebase's real-time database is designed to provide real-time data synchronization between clients. This means that data is immediately available to all connected clients as soon as it is updated.

**JSON:** Firebase's real-time database uses JSON (JavaScript Object Notation) to store and exchange data. JSON is a lightweight data format that is easy to read and write, making it ideal for use in web and mobile applications.

**Security:** Firebase Realtime Database provides security features such as authentication, authorization, and data validation to protect your data.

**Easy to use:** Firebase Realtime Database is easy to use and provides a simple API for interacting with the database. Integration with other Firebase services: Firebase Realtime Database integrates well with other Firebase services such as authentication, cloud messaging, and analytics. Firebase provides a range of advantages for developers, including real-time data synchronization, a NoSQL cloud database, real-time data exchange, and the use of JSON for storing and exchanging data. These features make Firebase an ideal choice for developers looking to build scalable, real-time applications

## 5.7 PERFORMANCE,SECURITY AND TRANSACTIONS

Android performance is excellent as it does multitasking. When it comes to security it follows confidentiality, authentication, integrity .Operation

performed on single unit of work is called transcation.A logical unit of work must have four properties

- **Atomicity**—All or no modifications are performed
- **Consistency**—When transaction has completed, all data is in a consistent state
- **Isolation**—Modifications made by concurrent transactions must be isolated from the modifications made by any other concurrent transactions
- **Durability**—After a transaction has completed, its effects are permanently in place in the system

## 5.8 PUBLISHING APP ON PLAY STORE

**Step 1: Make a Developer Account**

A developer account is must be needed to upload an app on the Google Play Store, and the process is very simple. Just go through Google Play Store and do as instruct.

**Step 2:** After you completed step 1 you will be redirected to this page where you have to click on the CREATE APPLICATION button.

Once you click on it a pop up will be shown like this where you must choose your Default language and Title of your app. Then click on the CREATE button.

**Step 3: Store listing**

After you completed step 2 you will be redirected to this page where you have to provide the Short description and Full description of your App.

**Step 4: App release**

After completing step 3 go to App releases then scroll down to Production track and click on MANAGE button.

**Step 5: Content rating**

Now after completing step 4 go to Content rating and click on CONTINUE button.

After that fill your email address as well as confirm the email address.

And then Select your app category.

And after answering them correctly don’t forget to click on SAVE QUESTIONNAIRE button.

Once you saved all those things then click on CALCULATE RATING button.

**Step 6: Pricing & distribution**

Then go to the Pricing & distribution section. Then select the country in which you want to available your app.

**Step 7: App content**

Then come to the App content section. And in the Privacy policy section click on the Start button.

**Step 8: App releases**

Again go back to the App releases section. And in the Production track click on the EDIT RELEASE button.

After usually 4 to 5 days they will review your app and let you know to either approve or reject your app.

## 5.10 SUMMARY

Database is a place where large amount of data can be stored efficiently in structured format. As when real time data needs to be captured firebase can be used as a database t o connect with android

## 5.11 REFERENCES

1.android.developers

2. Wrox. beginning android application development reference book

## 5.12 UNIT END QUESTION

1.Define SQLite Database. List its advantages and disadvantages

2. Define Firebase Database. List its advantages and disadvantages

3.Explain Content Providers.

4.Explain Loaders

5.Write different steps to publish app on google play store

❄❄❄❄❄❄❄