Chapter 1

1. Mobile Programming

1.1. Introduction

Mobile Devices need some type of Operating System to run its services. Earlier OS were fairly simple, since they supported limited capabilities. Modern devices however, added many of the features of a full-fledged desktop computer like CPU, GPU, large storage, multitasking, multipurpose communication device etc. Modern OS combines a PC with other mobile specific features like Bluetooth, GPS, speech recognition, video camera, infrared etc, so mobile OS had to grow to support all these features. This results in the growth of market with a number of different technology platforms. Different competing platforms are driven by different actors. This requires app developers to study and choose the most winning platform available.

What is Mobile OS?

- ✓ A software platform on top of which other apps can run on.
- ✓ Experienced a 3 phase evolution
  - PC oriented, Embedded System Oriented and Smart-phone oriented evolution
- ✓ The architecture goes from complex to simple or something in between
- ✓ Evolution has driven by tech advancements in HW, SW and the Internet
- ✓ Resulted in a variety of OS platforms available on the market

Some of the mobile operating systems are:

- ✓ iOS:- Developed by Apple inc. and distributed exclusively for Apple Hardware
- ✓ Windows Phone:- A proprietary software Smartphone OS developed by Microsoft
- ✓ Android:- Developed by Open Handset Alliance (OHA), Led by Google

What is Android?

Android is a mobile operating system currently developed by Google, based on the Linux kernel and designed primarily for touch screen mobile devices such as smart phones and tablets. Android offers a unified approach to application development for mobile devices which means developers need to develop only for Android, and their applications should be able to run on different devices powered by Android. Android is an open-source operating system named Android. Google has made the code for all the low-level "stuff" as well as the needed middleware to power and use an electronic device, and gave Android free of cost to anyone who wants to write code and build the operating system from it. There is even a full application framework included, so third-party apps can be built and installed, then made available for the user to run as they like.

The "proper" name for this is the Android Open Source Project, and this is what people mean when they say things like Android is open and free.

Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to develop applications on the Android platform using the Java Programming language.

1.1.1. History and Version of Android

The history and versions of android are interesting to know. The code names of android ranges from A to J currently, such as **Aestro**, **Blender**, **Cupcake**, **Donut**, **Eclair**, **Froyo**, **Gingerbread**, **Honeycomb**, **Ice Cream Sandwitch**, **Jelly Bean**, **KitKat**, **Lollipop, Marshmallow, Nougat, Oreo, Pie**. Let's understand the android history in a sequence.

1. Initially, **Andy Rubin** founded Android Incorporation in Palo Alto, California, United States in October, 2003.
2. In 17th August 2005, Google acquired android Incorporation. Since then, it is in the subsidiary of Google Incorporation.
3. The key employees of Android Incorporation are **Andy Rubin**, **Rich Miner**, **Chris White** and **Nick Sears**.
4. Originally intended for camera but shifted to smart phones later because of low market for camera only.
5. Android is the nick name of Andy Rubin given by coworkers because of his love to robots.
6. Google formed Open Handset Alliance (OHA) on 5th November 2007.

What is Open Handset Alliance?

The Open Handset Alliance is a group of 84 technology and mobile companies who have come together to accelerate innovation in mobile and offer consumers a richer, less expensive, and better mobile experience.

Which was the first commercially available Smartphone running Android?

The first commercially available Smartphone running Android was the **HTC Dream** it came to the market as the T-Mobile G1 in the USA, released on October 22 2008.

Android Versions Names

Android has its version named after sweets. The most recent is being Android 12 with API level 30.

Date	Version	Code name	API Level
September 23, 2008	1.0	No codename	1
February 9, 2009	1.1	Petit Four	2
April 27, 2009	1.5	Cupcake	3
September 15, 2009	1.6	Donut	4
October 26, 2009	2.0 - 2.1	Éclair	5 - 7
May 20, 2010	2.2 – 2.2.3	Froyo	8
December 6, 2010	2.3 – 2.3.7	Gingerbread	9 - 10
February 22, 2011	3.0 – 3.2.6	Honeycomb	11 - 13
October 18, 2011	4.0 – 4.0.4	Ice Cream Sandwich	14 - 15
July 9, 2012	4.1 - 4.3	Jelly Bean	16 - 18
October 31, 2013	4.4 - 4.4.4	KitKat	19-20
November 12, 2014	5.0 - 5.1.1	Lollipop	21-22
October 5, 2015	6.0 - 6.0.1	Marshmallow	23
August 22, 2016	7.0 -7.1.2	Nougat	24-25
August 21, 2017	8.0	Oreo	26-27
August 6, 2018	9.0	Pie	28
September 3, 2019	10.0	Android 10	29
September 8, 2020	11.0	Android 11	30

1.2. Android Architecture

Android is a software stack. **Android software stack** is categorized into five parts:

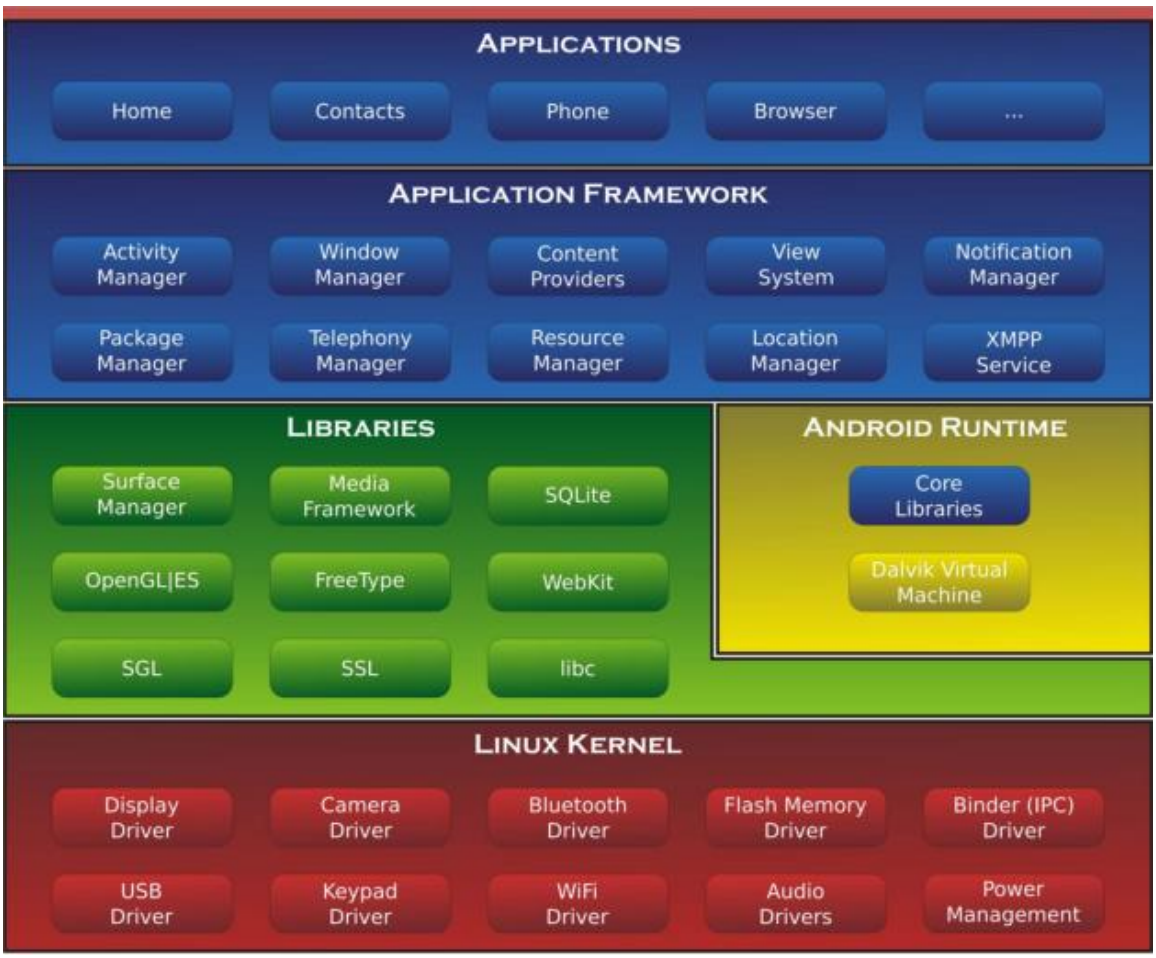


Figure 1.2.1 Android Architecture

Linux Kernel

In android the operating system is Linux Kernel. It is the heart of android architecture that exists at the root of android architecture. Linux kernel is responsible for device drivers, power management, memory management, device management and resource access. You can find the Linux Kernel version running on your device under "About phone" or "About tablet" in Android's settings.

Functions of Linux Kernel/ Operating System

- ✓ Memory Management: Allocate a memory to a new file, Free the memory when a specific file is deleted etc
- ✓ Power Management: providing power to various devices like Bluetooth, camera etc
- ✓ Resource Management: It provides resources to each process, thus providing the ability to do multiple operations at the same time. i.e. Surfing Internet, Listening Songs etc
- ✓ Driver Management: It handles installation of various drivers.

Middle Ware:

1. **Native Applications/Libraries:**

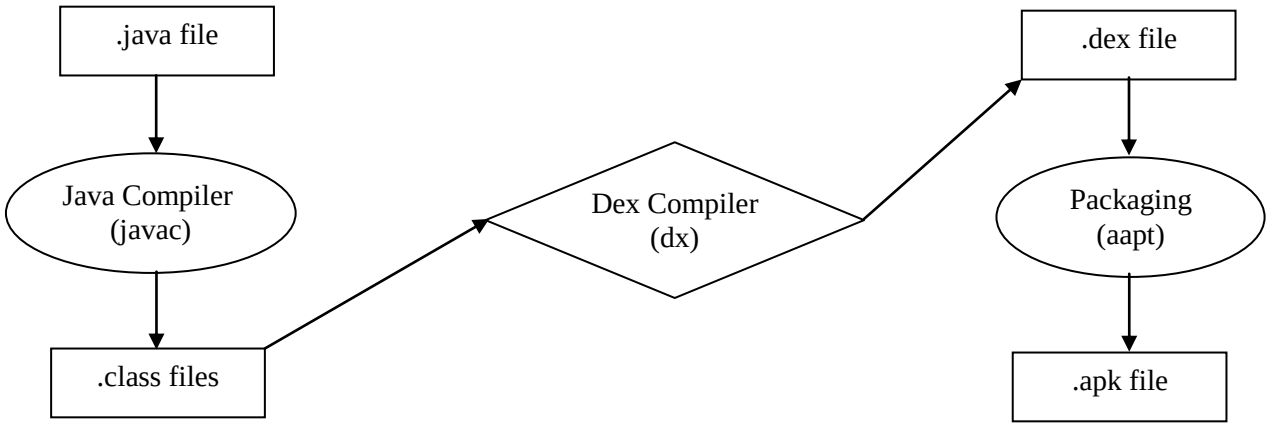
Just by using Java we cannot interact with native applications. (Here native applications means that programs written in some other languages like C, C++, Assembly language which are specific to hardware and operating system). Thus we need the support of native libraries for interacting with such low level media components. On the top of linux kernel, there are **Native libraries** such as WebKit, OpenGL, FreeType, SQLite, Media, C runtime library (libc) etc. The WebKit library is responsible for browser support, SQLite is for database, FreeType for font support, Media for playing and recording audio and video formats.

2. **Application Framework:**
   The android framework is the set of API's that allow developers to quickly and easily write apps for android phones. Android framework includes **Android API's** such as UI (User Interface), telephony, resources, locations, Content Providers (data) and package managers. It provides a lot of classes and interfaces for android application development. Application developers are allowed to make use of these services in their applications.
3. **Android Runtime:**
   It comprises of Dalvik Virtual Machine and Core Libraries which is responsible to run android application. DVM is like JVM but it is optimized for mobile devices. It consumes less memory and provides fast performance.

Applications:

Applications layer is the top most layer of Android Architecture. All applications using android framework uses android runtime and libraries, while android runtime and native libraries are using Linux Kernel.

Dalvik Virtual Machine:



Android applications can be developed using languages such as C++, C#, Java etc. But the official language is Java.

JDK (Java Development Kit) compile our Java code and it produce a file called **.class** file. It is also called as Java bytecode. Java Bytecode is a platform independent code because it can run on multiple platforms.

In android Dex compiler take all the .class file as input and it produce a single lightweight file called **.dex** file. Dex compile produces a single .dex file for multiple .class files. Dex is termed as Dalvik Executable Code. DVM takes this single .dex file as input and generates the final machine code.

With Android version 2.2(Froyo) Google introduced **JIT(Just in Time)** compilation in Dalvik. Till Android Version 5.0(Lollipop) Dalvik was the runtime used by Android which is being replaced by **ART(Android Run Time)** in later versions.

Why do we need a .dex file when actually even .class file is platform independent?

1. DVM is Register based which is designed to run on low memory unlike JVM which is stack based. Thus though DVM we are able to achieve small memory footprint with higher battery life.
2. Java tools are free but the JVM is not free, so that android developers from Google have made their own virtual machine and made it as free.

ART vs Dalvik

ART is a new Android runtime which was introduced experimentally in the 4.4 release KitKat. If you have KitKat then you can see the preview in "setting >Developer Option>Select Runtime".

Why did Google switched from Dalvik to ART?

Android apps are deployed in Dalvik bytecode, which is portable, unlike native code. In order to be able to run the app on a device, the code has to be compiled to machine code.

Dalvik is based on JIT compilation. It means that each time you run an app, the part of the code required for its execution is going to be translated (compiled) to machine code at that moment. As you progress through the app, additional code is going to be compiled and cached, So that the system can reuse the code while the app is running. Since JIT compiles only a part of the code, it has a smaller memory footprint and uses less physical space on the device.

ART, on the other hand compiles the intermediate language, Dalvik bytecode, into a system independent binary. The whole code of the app will be pre-compiled during installation (once), thus removing the lag that we see when we open an app on our device. With no need for JIT compilation, the code should execute much faster. Except for the potential speed increase, the use of ART can provide an important secondary benefit. As ART runs app machine code directly (native execution), it does not hit the CPU as hard as **Just-in-Time** code compiling on Dalvik. Less CPU usage results in less battery drain, which is a big plus for portable devices in general.

So why was not ART implemented earlier?

- ✓ First of all, the generated machine code requires more space than the existing bytecode.
- ✓ Moreover, the code is pre-compiled at install time, so the installation process takes a bit longer.
- ✓ Furthermore, it also corresponds to a larger memory footprint at execution time.
- ✓ This means that fewer apps run concurrently

When first Android devices hit the market, memory and storage capacity were significantly smaller and presented a bottleneck for performance. This is the reason why a JIT approach was the preferred option at that time. Today, memory is much cheaper and thus more abundant, even on low-end devices, so ART is a logical step forward.

Android Application Components

- ✓ **Activities:** An activity in Android represents a single screen with which a user can interact with. An application can have more than one Activity and each Activity operates independently, but can be linked to one another and each Activity you create must be defined in your application's manifest file.

Each Activity in android will be subclass of Activity class defined in Android SDK. Thus public class MainActivity extends Activity (if MainActivity is the name of the Activity).

- ✓ **Services:** Long running background process without any need of user interaction is known as Service. For example, as service might play music in the background while the user is in different application, or it might fetch data over the network without blocking user interaction with an activity. Each service in android will be subclass of service class defined in Android SDK. Thus public class MyService extends Service (if MyService is the name of the service).
- ✓ **Broadcast Receiver:** They handle communication between Android OS and applications. For example the notification that the device battery is low, the sign of earphone as soon as you plug the headset. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs.
- ✓ **Content Provider:** content providers are used to share data between the applications. In android, the data cannot be shared directly between the two applications.

Additional Components

There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them. These components are:

- ✓ ***Fragments***: Represent a behavior or a portion of user interface in an Activity.
- ✓ ***Views***: UI elements that are drawn onscreen including buttons, lists forms etc.
- ✓ ***Layouts***: View hierarchies that control screen format and appearance of the views.
- ✓ ***Intents***: Messages wiring components together.
- ✓ ***Resources***: External elements, such as strings, constants and drawable pictures.
- ✓ ***Manifest***: Configuration file for the application.

1.3. Getting Started

In order to write an Android application, we need a Java Development Kit (JDK), Java Runtime Environment (JRE) and development environment. The common Android application development environments are Android Studio, Eclipse and NetBeans. But officially Android Studio is the recommended IDE for developing Android Projects. Android Studio is a very useful tool made by Google for all Android Developers.

Android Studio includes everything you need to begin developing Android apps. Included in the download kit, are the Software Development Kit (SDK), with all the Android libraries we may need, and the infrastructure to download the many Android emulator instances, so that we can initially run our application, without needing a real device.

Android Emulator is used to run, debug and test the android application. If you don't have the real device, it can be the best way to run, debug and test the application. It uses an open source processor emulator technology called **QEMU**. The emulator tool enables you to start the emulator from the command line. You need to write: ***emulator -avd <AVD NAME>***

In case of Eclipse IDE, you can create AVD by **Window menu > AVD Manager > New**.

If you want to test the android applications using a real device, unlock the developer's option in Android as follows: Go to ***Settings>General>About Phone***. Then scroll and select Software ***Information>Build Number.*** Now quickly tap on "Build Number" seven times and you will see the message "You are now a developer".

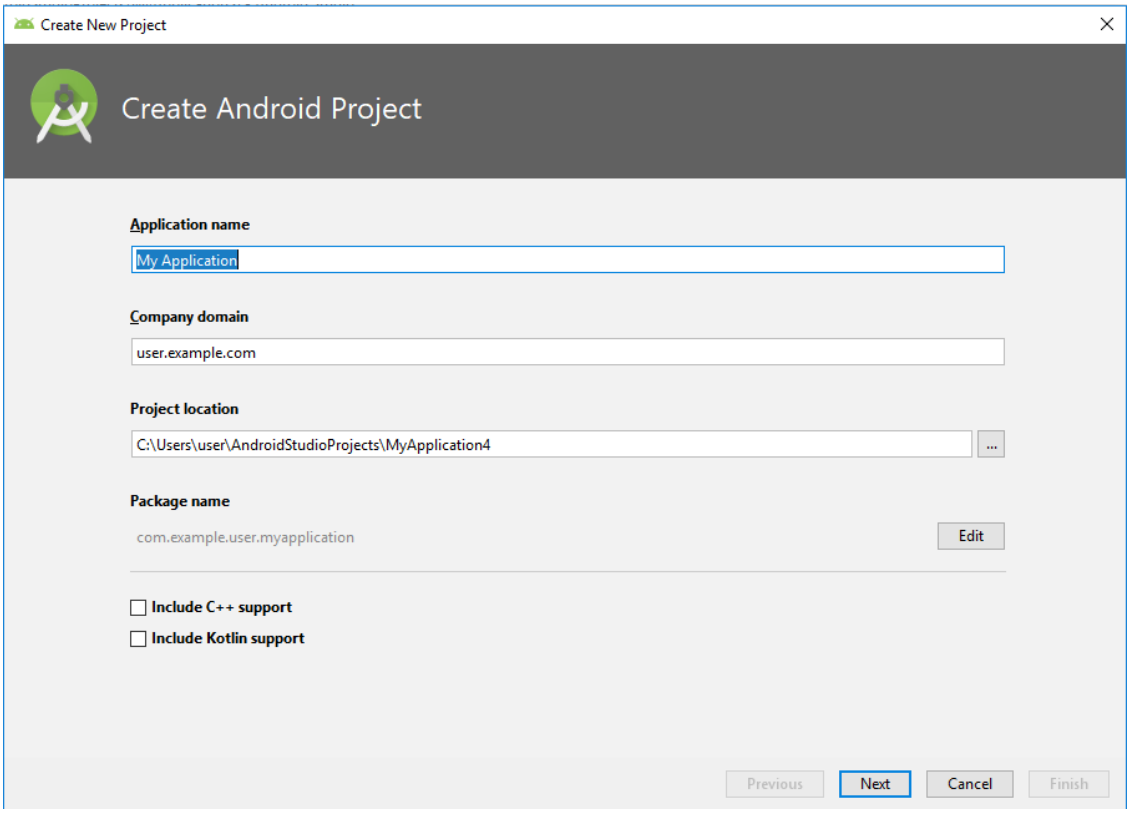
Android Studio Installation

- ✓ Install JDK and JRE in your system
- ✓ Launch Android Studio .exe. If the JDK is not detected automatically you need to mention its path explicitly.
- ✓ Proceed further to include all the components
- ✓ Click Next. Until you complete the installation.

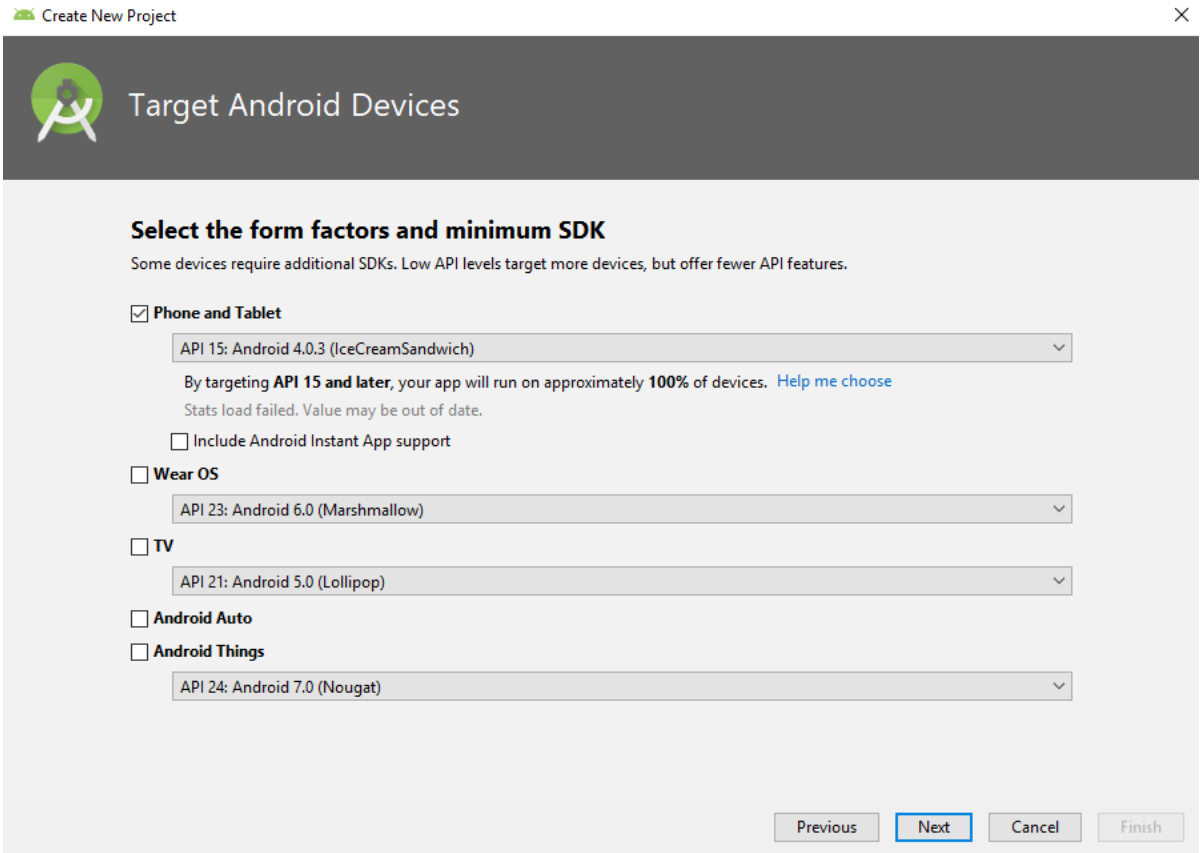
Building your first app (Hello World)

Step1: Start a new Android Studio Project

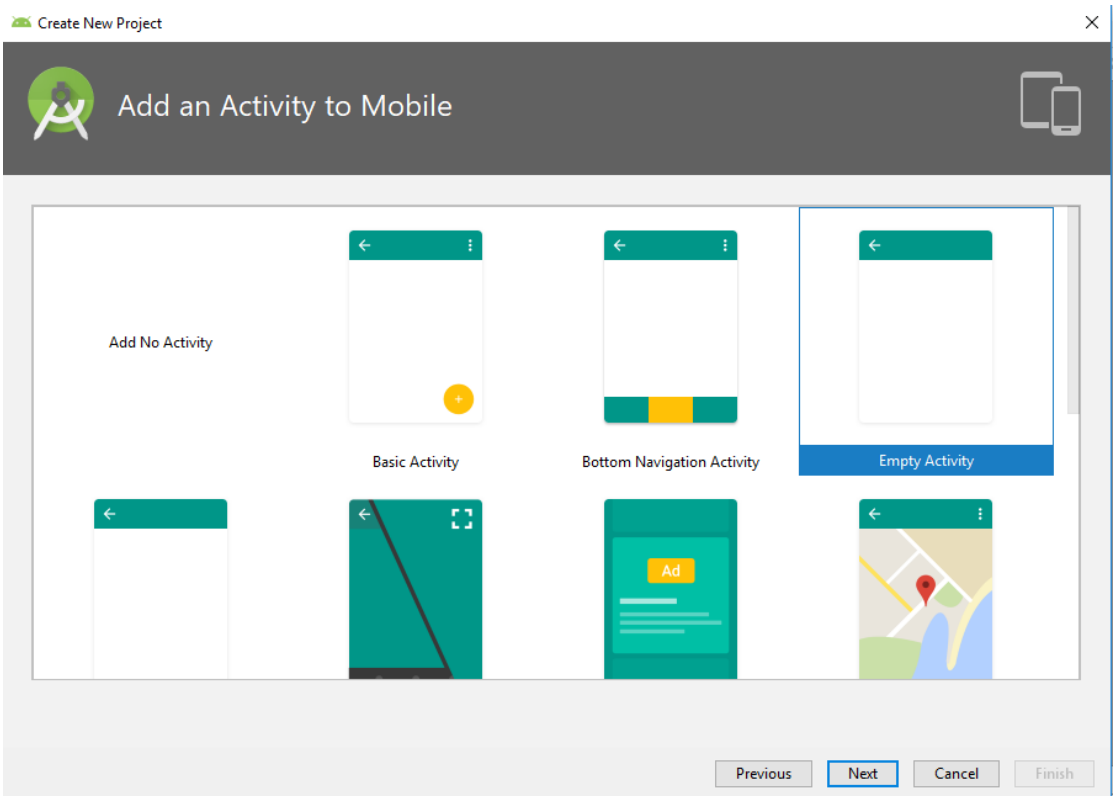
Step2: Fill Application Name and Package Name and click next



Step3: Select Phone and Tablet in Target Android Device and Proceed



Step4: Select Empty Activity and click Next.

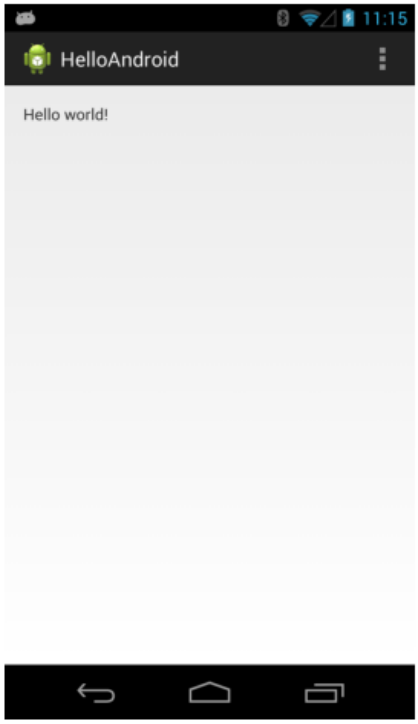


Step5 Click next and Finish.

Running Application

In Android Studio, click the app module in the Project window and then select Run>Run 'app'(or Run Icon in the toolbar). In the Select Deployment Target window, select your device and click ok.

Output



Android Project Structure

When you start a new project, Android Studio creates the necessary structure for all your files and makes them visible in the project window on the left side of the IDE *click View>Tool Windows>Project.* By default, Android Studio displays our project files in the **Android view**.

Android App Module:

When you create a new project, the default module name is "**app".** A module is a collection of source files and build settings that allow you to divide your project into discrete units of functionality. Your Project can have one or more modules and one module may use another module as a dependency. Each module can be independently built, tested and debugged.

Module provides a container for your app's source code, resource files, and app level settings such as the module-level build file and Android Manifest file. To add a new module to your project *click File>New>New Module*. In the create new module window, Android Studio offers different app modules such as Phone & Tablet, Android Wear, Android TV and Glass Modules. Each module provides essential files and some code templates that are appropriate for the corresponding app or device type.

Android Manifest.xml:

Android Manifest.xml describes the fundamental characteristics of an app and each of its components. It works as an interface between your android OS and your application, so if you forget to declare your components in this file then it will not be considered by the OS.

The default AndroidManifest.xml files looks like this:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.helloworld"
   android:versionCode="1"
   android:versionName="1.0" >
<uses-sdk
      android:minSdkVersion="8"
      android:targetSdkVersion="15" />
   <application
       android:icon="@drawable/ic_launcher"
       android:label="@string/app_name"
       android:theme="@style/AppTheme" >
       <activity
           android:name=".MainActivity"
           android:label="@string/title_activity_main" >
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category
  android:name="android.intent.category.LAUNCHER"/>
           </intent-filter>
       </activity>
   </application>
</manifest>
```

Android Java

It contains the Java source code files separated by package names and JUnit test code. All the Activity classes are stored inside java folder. For e.g. The Activity class called MainActivity.java is stored in this directory under the package name you mentioned in the wizard.

Android Resource (res)

It contains all non-code resources, such as xml layouts, UI strings and bitmap images, divided into corresponding sub-directories. Some of the examples of resource types are:

- ***Animation Resources***- defines predetermined animations. Animations are saved in *res/anim/* and accessed from the *R.anim* class. Frame animations are saved in *res/drawable/* and accessed from the *R.drawable* class.
- ***Color State List Resource***-it defines color resources that changes based on the view state. It is saved in *res/color/* and accessed from the *R.color* class
- ***Drawable Resource***- defines various graphics with citmaps or XML. It is saved in *res/drawable/* and accessed from the *R.drawable* class.
- ***Layout Resource***- defines the layout for your application UI. It is saved in *res/layout/* and accessed from the *R.layout* class.
- ***Menu Resource***- defines the contents of your application menus. It is saved in *res/menu/* and accessed from the *R.menu* class.
- ***String Resource***- defines string, string arrays and plurals (and include string formatting and styling). It is saved in *res/values/* and accessed from the *R.string*, *R.array* and *R.plurals* classes.
- ***Style Resource***- defines the look and format for UI elements. It is saved in *res/values/* and accessed from the *R.style* class.

Android Strings

A string resource provides text strings for your application with optional text styling and formatting.

Types of resources that can provide your application with strings:

- **String:** XML resource that provides a single string.
- **String Array:** XML resource that provides an array of strings.

Android Styles

A style resource defines the format and look for a UI. It can specify properties such as height, padding, font color, font size, background color etc. Styles in Android share a similar philosophy to cascading style sheets in web design i.e they allow you to separate the design from the content.

Chapter 2

2. Android Layout and UI Widgets

2.1. Views and View Groups

What are Views?

Anything which occupies a rectangular area on the screen and is responsible for drawing and event handling is known as a view. Thus every UI element is subclass of view, to be more precise of **android.view.View.**

The android SDK provides a set of pre-built views that can be used to construct a user interface, for example buttons, checkbox, EditText etc. which are also known as widgets.

Widgets: - can be thought of as a pre-built control through which the user interacts with your application. There is nothing such as widget class in android. It simply means that the class is in **android.widgets** package in android.

Android contains the following commonly used View subclasses: TextView, EditText, ImageView, ProgressBar, Button, ImageButton, ChechBox etc. We can also create our widget by extending view or any of its subclass inside android.widgets.

What are ViewGroup?

The ViewGroup is a subclass of View (*android.view.View)* and provides invisible container that hold other views or other ViewGroups and define their layout properties. Thus all the Layouts and Containers inherit from ViewGroup. The view group is the base class for layouts and views containers.

Android contains the following commonly used ViewGroup subclasses: LinearLayout, RelativeLayout, ListView, GridView etc.

2.2. Layout and its types

Layouts are subclasses of ViewGroup which specifies how a view should be arranged inside the viewgroups. It defines the visual structure for a user interface.

You can create layouts in two ways:-

- Declare UI elements in XML
- Instantiate layout elements at runtime

The standard Layouts are:

- ✓ **LinearLayout**: A layout that arranges its children in a single column or a single row. The direction of the row can be set by calling setOrientation( ).
- ✓ **RelativeLayout**: A layout where the positions of the children can be described in relation to each other or to the parent.
- ✓ **FrameLayout**: It is used to block out an area on the screen to display a single item.
- ✓ **TableLayout**: A layout that arranges its children into rows and columns.

2.2.1. Linear Layout

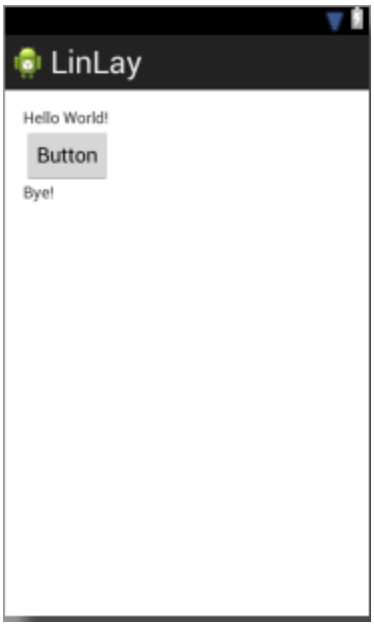
Linear layout is a view group that aligns all children in a single direction, vertically or horizontally. Layout direction is specified by ***android:orientation="horizontal" or android:orientation="vertically"*** attribute.

LinearLayout horizontal arranges the elements horizontally in the LinearLayout. **Example**: In the example below we have arranged a TextView and a Button horizontally.

```
<LinearLayout 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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity"
    android:orientation="horizontal">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bye" />

</LinearLayout>
```

LinearLayout vertical arranges the elements vertically in the LinearLayout. **Example**: In the example below we have arranged a TextView and a Button vertically.

```
<LinearLayout 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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bye" />

</LinearLayout>
```



2.2.2. Relative Layout

RelativeLayout lays out elements based on their relationships with one another, and with the parent container. This is arguably the most complicated layout and we need several properties to actually get the layout we want.

Relative to container:

These properties will layout elements relative to the parent container.

- ***android:layout_alignParentBottom***-places the element on the bottom of the container.
- ***android:layout_alignParentLeft***-places the element on the left side of the container.

- ***android:layout_alignParentRight***-places the element on the right side of the container.
- ***android:layout_alignParentTop***-places the element on the top of the container.
- ***android:layout_centerHorizontal***-centers the elements horizontally within its parent container.
- ***android:layout_centerInParent***- centers the elements both horizontally and vertically within its parent container.
- ***android:layout_centerVertical***- centers the elements vertically within its parent container..

Relative to other Elements:

These properties allow you to layout elements relative to other elements on screen. The value for each of these elements is the id of the element you are using to layout the new element.

Each element that is used in this way must have an ID defined using ***android:id="@_id/xxxxx"*** where xxxxx is replaced with the desired id. You use "***@id/xxxxx"*** to reference an element by its id. One thing to remember is that referencing an element before it has been declared will produce an error.

- ***android:layout_above***- places an element above the specified element.
- ***android:layout_below***- places an element below the specified element.
- ***android:layout_toLeftOf***- places an element to the left of the specified element.
- ***android:layout_toRightOf***- places an element to the right of the specified element.

Relative with other Elements:

These properties allow you to specify how elements are aligned in relation to other elements.

- ***android:layout_alignBaseline***- aligns baseline of the new element with baseline of the specified element.
- ***android:layout_alignBottom***- aligns bottom of the new element in with bottom of the specified element.
- ***android:layout_alignLeft***- aligns left edge of the new element with left edge of the specified element.
- ***android:layout_alignRight***- aligns right edge of the new element with right edge of the specified element.
- ***android:layout_alignTop***- places top of the new element in alignment with the top of the specified element.

```
<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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world"
```

```
            android:id="@+id/textview"
            android:layout_centerVertical="true"
            android:layout_centerInParent="true"/>
          <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/button"
            android:id="@+id/button1"
           android:layout_above="@+id/textview"
            android:layout_alignParentStart="true"/>

    </RelativeLayout>
```

2.2.3. Table Layout

TableLayout organizes content into rows and columns. Table layouts in android works, in the same way HTML table layout work.

Example:

```
<TableLayout 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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

   <TableRow>
       <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
   </TableRow>
   <TableRow>
       <TextView
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="@string/your_name" />
       <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="text"
        android:text="@string/nametxt" />
   </TableRow>
   <TableRow>
       <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/submit" />
       <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cancel" />
   </TableRow>

</TableLayout>
```

2.2.4. Frame Layout

FrameLayout is designed to display a single item at a time. You can have multiple elements within a FrameLayout but each element will be positioned based on the top left of the screen. Elements that overlap will be displayed overlapping. See the following example:

```
<FrameLayout 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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <ImageView
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/image"
        android:src="@drawable/ic_launcher" />"
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textview2"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="@string/textview3"
        android:layout_gravity="center" />

</FrameLayout>
```

As you can see on the above output the ImageView and TextView fill the parent in both horizontal and vertical layout. Gravity specifies where the text appears within its container, so I set that to center. If the gravity was not set the text would have appeared at the top left of the screen.

FrameLayout can become more useful when elements are hidden and displayed programmatically. You can use the attribute ***android:visibility*** in the XML to hide specific elements. You can even call ***setVisibility*** method from the code to accomplish the same thing. The three available visibility values are **visible, invisible** (does not display but still takes up space in the layout) **and gone** (does not display and does not take space in the layout)**.**

2.2.5. Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and there are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

- ✓ ***android:id*** - This is the ID which uniquely identifies the view.
- ✓ ***android:layout_width*** - This is the width of the layout.
- ✓ ***android:layout_height*** -This is the height of the layout
- ✓ ***android:layout_marginTop*** - This is the extra space on the top side of the layout.
- ✓ ***android:layout_marginBottom***- This is the extra space on the bottom side of the layout.
- ✓ ***android:layout_marginLeft***- This is the extra space on the left side of the layout.
- ✓ ***android:layout_marginRight***- This is the extra space on the right side of the layout.
- ✓ ***android:layout_gravity***- This specifies how child Views are positioned.

- ✓ ***android:layout_weight***- This specifies how much of the extra space in the layout should be allocated to the View.
- ✓ ***android:layout_x***- This specifies the x-coordinate of the layout.
- ✓ ***android:layout_y***- This specifies the y-coordinate of the layout.
- ✓ ***android:paddingLeft***- This is the left padding filled for the layout.
- ✓ ***android:paddingRight***- This is the right padding filled for the layout.
- ✓ ***android:paddingTop***- This is the top padding filled for the layout.
- ✓ ***android:paddingBottom***-This is the bottom padding filled for the layout.

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

You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height −

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

Gravity attribute plays important role in positioning the view object and it can take one or more (separated by '|') of the following constant values.

Constant	Value	Description
Top	0x30	Push object to the top of its container, not changing its size.
bottom	0x50	Push object to the bottom of its container, not changing its size.
Left	0x03	Push object to the left of its container, not changing its size.
Right	0x05	Push object to the right of its container, not changing its size.
center_vertical	0x10	Place object in the vertical center of its container, not changing its size.
fill_vertical	0x70	Grow the vertical size of the object if needed so it completely fills its container.
center_horizontal	0x01	Place object in the horizontal center of its container, not changing its size.
fill_horizontal	0x07	Grow the horizontal size of the object if needed so it completely fills its container.
Center	0x11	Place the object in the center of its container in both the vertical and horizontal axis, not changing its size.
Fill	0x77	Grow the horizontal and vertical size of the object if needed so it completely fills its container.

clip_vertical	0x80	Additional option that can be set to have the top and/or bottom edges of the child clipped to its container's bounds. The clip will be based on the vertical gravity: a top gravity will clip the bottom edge, a bottom gravity will clip the top edge, and neither will clip both edges.
clip_horizontal	0x08	Additional option that can be set to have the left and/or right edges of the child clipped to its container's bounds. The clip will be based on the horizontal gravity: a left gravity will clip the right edge, a right gravity will clip the left edge, and neither will clip both edges.
Start	0x00800003	Push object to the beginning of its container, not changing its size.
End	0x00800005	Push object to the end of its container, not changing its size.

View Identification

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

```
android:id="@+id/my_button"
```

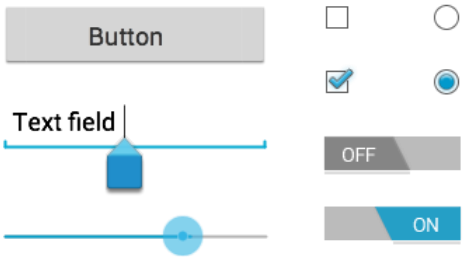
Following is a brief description of @ and + signs −

- ✓ The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
- ✓ The plus-symbol (+) means that this is a new resource name that must be created and added to our resources. To create an instance of the view object and capture it from the layout, use the following −

```
Button myButton = (Button) findViewById(R.id.my_button);
```

2.3. Android User Interface (UI) widgets

Android UI widgets are input controls which are used to build a user interface for your application. Input controls are the interactive components in your app's user interface. A **View** is an object that draws something on the screen that the user can interact with such as Text fields, Buttons etc and a **ViewGroup** is an object that holds other View (and ViewGroup) objects in order to define the layout of the user interface. The following picture shows some of the input controls.



Android provides various UI controls that allow you to build the graphical user interface for your app. These are ***TextView***, ***EditText***, ***AutoCompleteTextView***, ***Button***, ***ImageButton***, ***CheckBox***, ***ToggleButton***, ***RadioButton***, ***RadioGroup***, ***ProgressBar***, ***Spinner***, ***TimePicker***, ***DatePicker etc***

2.3.1. Create UI Controls

Buttons

In Android, **Button** represents a push button. A Push buttons can be clicked, or pressed by the user to perform an action. There are different types of buttons used in android such as CompoundButton, ToggleButton, RadioButton.

Button is a subclass of TextView class and compound button is the subclass of Button class. **On a button we can perform different actions or events like click event, pressed event, touch event etc.**

Android buttons are GUI components which are sensible to taps (clicks) by the user. *When the user taps/clicks on button in an Android app, the app can respond to the click/tap*. These buttons can be divided into two categories: the first is Buttons with text on, and second is buttons with an image on. A button with images on can contain both an image and a text. Android buttons with images on are also called *ImageButton*.

Example: the following example shows how to create a button via a layout xml file.

```
<Button
    android:id="@+id/btn1 "
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Display"
    android:onClick="btnClick()"
/>
```

- ✓ *android:id*- provides a unique identification to a UI element**.** Thus when there are two or more buttons, it is this id which helps us to distinguish between the two in java code (usually done by ***findViewByID( )*** method)
- ✓ *android:layout_width & android:layout_height*- defines the size that the UI element should occupay.
- ✓ *android:text*- defines the text which the UI element should display. If you want to create an image button use *android:drawableLeft* property and set its value "@mipmap/ic_launcher". (Where mimap is the directory in which the image is stored and ic_launcher is name of the image.

Example: the following example shows how to create a button programmatically in java file.

```
    public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
       Button button=new Button(this);
   button.setText("Display");
 RelativeLayout relativeLayout=(RelativeLayout) findViewById(R.id.rootlayout);
relativeLayout.addView(button);
  }
}
```

- ✓ Create the instance of the Button class by passing the current context as *this*.
- ✓ Set the text which the button should display.
- ✓ Create the object of the ViewGroup (Layout) in which the button needs to be placed

- ✓ Find the layout by using findViewById( ) method
- ✓ Add the button to the layout by addView function

ImageButton

As the name says, the ImageButton component is a button with an image on it. The ImageButton is represented by the Android class *android.widget.ImageButton*. It is similar with the button component except that its value is an Image than a text.

Example:

- if you want to create an image button use ***android:src*** attribute to define the location and name of the image which you want to use as button.

```
<ImageButton
  ...
  android:src="@drawable/the_image_button_icon"
/>
```

- If you want to create the button programmatically add *button.setImageSource(R.drawable.the_image_button_icon);* to define the location and name of the image which you want to use as button.

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
       Button button=new Button(this);
   button.setText("Display");
   button.setImageSource(R.drawable.the_image_button_icon);
RelativeLayout relativeLayout=(RelativeLayout) findViewById(R.id.rootlayout);
relativeLayout.addView(button);
  }
}
```

Button Click Event

When the user clicks on a button, the Button object receives an on-click event. To define the click event handler for a button, add the *android:onClick* attribute to the <Button> element in your xml layout. The value for this attribute must be the name of the method you want to call in response to a click event. For instance *android:onClick="btnOnclick()"*. The Activity hosting the layout must then implement the corresponding method. **Example**: the following method will display "Button clicked!" message when a user clicks on the button.

```
  protected void btnClick(View view) {
    Toast.makeText(getBaseContext(), "Button Clicked!",Toast.LENGTH_SHORT).show();  }
```

Android Toast

Toast is a notification message that pop up for a specific duration. It only fills the amount of space required for the message and the current activity remains visible and interactive. The ***android.widget.Toast*** class is the subclass of ***java.lang.Object*** class.

Constants of Toast Class:

- ✓ *public static final int LENGTH_LONG*- displays view for a long duration of time i.e. 3.5 seconds. (Toast.LENGTH_LONG)
- ✓ *public static final int LENGTH_SHORT*- displays view for a short duration of time i.e. 2 seconds. (Toast.LENGTH_SHORT)

Example:

```
Context context=getApplicationContext();
CharSequence text=”Button Clicked!”;
int duration=Toast.LENGTH_SHORT;
Toast toast=Toast.makeText(context,text,duration);
toast.show();
```

You can also chain methods like this:

```
Toast.makeText(getApplicationContext(), “Button Clicked!”,Toast.LENGTH_SHORT).show();
```

The method takes three parameters:

- ✓ an application context, the text message and the duration for the toast.

Positioning the Toast:

A standard toast notification appears near the bottom of the screen, centered horizontally. You can change this position with the **setGravity(int, int, int)** method.

The method accepts three parameters:

- ✓ a gravity constant, an x-position offset and y-position offset.

Example: this example displays the toast at the top-left corner of your application

```
Toast toast=Toast.makeText(this,”Toast MSG”,Toast.LENGTH_SHORT);
toast.setGravity(Gravity.AXIS_PULL_AFTER,0,0);
toast.show();
```

Android TextView

The TextView view is used to display text to the user. This is the most basic view and one that you will frequently use when you develop Android applications. If you need to allow users to edit the text displayed, you should use the subclass of TextView, EditText. In some other platforms the TextView is commonly known as label view. It’s sole purpose to display a text on the screen. **For example:** the following example will display the text “Hello World” on the screen

```
<TextView
    android:id="@+id/txtview "
    android:layout_width="fill_parent "
    android:layout_height="wrap_content"
    android:text="Hello World"
/>
```

TextView code in JAVA:

```
TextView textView = (TextView) findViewById(R.id.textView);
```

```
textView.setText("Hello World"); //set text for text view
```

Attributes of TextView:

Some of the attributes of a TextView are:

- ✓ **id:** id is an attribute used to uniquely identify a text view.
- ✓ **gravity:** The gravity attribute is an optional attribute which is used to control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc.
- ✓ **text:** text attribute is used to set the text in a text view. We can set the text in xml as well as in the java class.
- ✓ **textColor:** textColor attribute is used to set the text color of a text view. Color value is in the form of "#argb", "#rgb", "#rrggbb", or "#aarrggbb".
- ✓ **textSize:** textSize attribute is used to set the size of text of a text view. We can set the text size in sp(scale independent pixel) or dp(density pixel).
- ✓ **textStyle:** textStyle attribute is used to set the text style of a text view. The possible text styles are bold, italic and normal. If we need to use two or more styles for a text view then "|" operator is used for that. Example: android:textStyle="bold|italics"
- ✓ **background:** background attribute is used to set the background of a text view. We can set a color or a drawable in the background of a text view.
- ✓ **padding:** padding attribute is used to set the padding from left, right, top or bottom. In above example code of background we also set the 10dp padding from all the side's of text view.

Example: Below is the example of TextView in which we display a text view and set the text in xml file and then change the text on button click event programmatically. Below is the final output and code:

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout . . .>
    <TextView
        android:id="@+id/simpleTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
        android:text="Before Clicking"
        android:textColor="#f00"
        android:textSize="25sp"
        android:textStyle="bold|italic"
        android:layout_marginTop="50dp"/>
    <Button
        android:id="@+id/btnChangeText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:background="#f00"
        android:padding="10dp"
        android:text="Change Text"
        android:textColor="#fff"
        android:textStyle="bold" />
```

MainActivity.java

```
import android.graphics.Color;
import
android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.*;
import android.widget.*;
public class MainActivity extends
AppCompatActivity {
    @Override
    protected void onCreate(Bundle
savedInstanceState) {
      super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
        final TextView simpleTextView =
(TextView)
findViewById(R.id.simpleTextView);
   Button changeText = (Button)
findViewById(R.id.btnChangeText);
changeText.setOnClickListener(new
View.OnClickListener() {
            @Override
   public void onClick(View view) {
```

`</RelativeLayout>`	`simpleTextView.setText("After Clicking");`
	`} });`
	`}}`

Android EditText

EditText view is a subclass of the TextView that allows users to edit its text content. It is also known as a text field in other platforms. It can be either single line or multi-line. Touching a text field places the cursor and automatically displays the keyboard. In addition to typing, text fields allow for a variety of activities such as text selection (cut, copy, paste) and data look-up via auto-completion. You can specify the type of keyboard you want for your EditText object with the *android:inputType* attribute. For example, if you want the user to input a phone number, you should use the phone input type.

Example:

```
<EditText
        android:id="@+id/search "
        android:layout_width="fill_parent "
        android:layout_height="wrap_content"
        android:hint="Search Tex"
        android:inputTYpe="text"
        android:imeOptions="actionSend"
    />
```

There are different input types, the most common are:

- ✓ ***text***- displays a normal text keyboard
- ✓ ***textEmailAddress***- displays a normal text keyboard with the @ character
- ✓ ***textUri***- displays a normal text keyboard with the / character
- ✓ ***number***- displays a basic number keypad
- ✓ ***phone***- displays a phone style keypad

The *android:inputType* property also allows you to specify certain keyboard behaviors such as whether to capitalize all new words or use features like auto-complete and spelling suggestions.

- ✓ ***textCapSentences***-displays normal text keyboard that capitalizes the first letter for each new sentence.
- ✓ ***textCapWords***- displays normal text keyboard that capitalizes every word.
- ✓ ***textAutoCorrect***- displays normal text keyboard that corrects commonly misspelled words
- ✓ ***textPassword***- displays normal text keyboard but the characters entered turn into dots.
- ✓ ***textMultiLine***- displays normal text keyboard that allow users to input long strings of text that include line breaks.(carriage return)

Retrieving / Getting the Value From EditText In Java Class:

Below is the example code of EditText in which we retrieve the value from an EditText in Java class. We have used this code in the example you will find at the end of this post.

```
EditText simpleEditText = (EditText) findViewById(R.id.search);
```

```
String editTextValue = simpleEditText.getText().toString();
```

Keyboard Actions

In addition to changing the keyboard's input type android allows you to specify an action to be made when users have completed their input. The action specifies the button that appears in place of the carriage return key and the action to be made, such as Search or Send. You can specify the action by setting the *android:imeOptions* attribute. For example, *android:imeOptions="actionSend"* in the above example specifies a send action.

Listening to Keyboard Actions

If you have specified a keyboard action for the input method using *android:imeOptions* attribute, you can listen for the specific action event using an *TextView.onEditorActionListener*.

The *TextView.onEditorActionListener* interface provides a callback method called *onEditorAction()* that indicates the action type invoked with an action ID such as IME_ACTION_SEND or IME_ACTION_SEARCH. **For example**, here is how you can listen when the user clicks the send button on the keyboard.

```
  EditText editText= (EditText)findViewById(R.id.search);
 editText.setOnEditorActionListener(new OnEditorActionListener(){

    @Override
    Public boolean onEditorAction(TextView v,int actionId, KeyEvent event){
     boolean handled=false;
     if(actionId==EditorInfo.IME_ACTION_SEND){
       sendMessage();
       handled=true;
      }
        return handled;
  }
});
```

Attributes of EditText:

EditText supports the attributes mentioned for TextView , the attribute described below and a lot more:

- ✓ **hint:** hint is an attribute used to set the hint i.e. what you want user to enter in this edit text. Whenever user start to type in edit text the hint will automatically disappear.

Example: Below is the example of edit text in which we get the value from multiple edittexts and on button click event the Toast will show the data defined in Edittext.

activity_main.xml	MainActivity.java

`<?xml version="1.0" encoding="utf-8"?>`
`<RelativeLayout ...>`
`<EditText`
`android:id="@+id/editText1"`
`android:layout_width="wrap_content"`
`android:layout_height="wrap_content"`
`android:layout_alignParentLeft="true"`
	`import`
	`android.support.v7.app.AppCompatActivity;`
	`import android.os.Bundle;`
	`import android.view.View;`
	`import android.widget.Button;`
	`import android.widget.EditText;`
	`import android.widget.Toast;`

```
android:layout_alignParentStart="true"
android:layout_alignParentTop="true"
android:layout_marginLeft="50dp"
android:layout_marginStart="50dp"
android:layout_marginTop="24dp"
android:ems="10"
android:hint="@string/name"
android:inputType="textPersonName"
android:selectAllOnFocus="true" />

    <EditText
 android:id="@+id/editText2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignLeft="@+id/editText1"
android:layout_alignStart="@+id/editText1"
android:layout_below="@+id/editText1"
android:layout_marginTop="19dp"
android:ems="10"
android:hint="@string/password_0_9"
android:inputType="numberPassword" />
    <EditText
 android:id="@+id/editText3"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignLeft="@+id/editText2"
android:layout_alignStart="@+id/editText2"
android:layout_below="@+id/editText2"
android:layout_marginTop="12dp"
android:ems="10"
android:hint="@string/e_mail"
android:inputType="textEmailAddress" />
    <EditText
        android:id="@+id/editText4"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignLeft="@+id/editText3"
android:layout_alignStart="@+id/editText3"
android:layout_below="@+id/editText3"
android:layout_marginTop="18dp"
android:ems="10"
android:hint="@string/date"
android:inputType="date" />
    <EditText
        android:id="@+id/editText5"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignLeft="@+id/editText4"
android:layout_alignStart="@+id/editText4"
android:layout_below="@+id/editText4"
android:layout_marginTop="18dp"
android:ems="10"
android:hint="@string/contact_number"
android:inputType="phone" />
    <Button
        android:id="@+id/button"
style="@android:style/Widget.Button"
android:layout_width="match_parent"
android:layout_height="wrap_content"
```

```
public class MainActivity extends
AppCompatActivity {
    Button submit;
    EditText name, password, email,
contact, date;
    @Override
    protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
name = (EditText)
findViewById(R.id.editText1);
password = (EditText)
findViewById(R.id.editText2);
 email = (EditText)
findViewById(R.id.editText3);
 date = (EditText)
findViewById(R.id.editText4);
 contact = (EditText)
findViewById(R.id.editText5);
  submit = (Button)
findViewById(R.id.button);
  submit.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if
(name.getText().toString().isEmpty() ||
password.getText().toString().isEmpty()
|| email.getText().toString().isEmpty()
|| date.getText().toString().isEmpty()
||contact.getText().toString().isEmpty())
{
Toast.makeText(getApplicationContext(),
"Enter the Data",
Toast.LENGTH_SHORT).show();
                } else {

Toast.makeText(getApplicationContext(),
"Name -  " + name.getText().toString() +
" \n" + "Password -  " +
password.getText().toString()
 + " \n" + "E-Mail -  " +
email.getText().toString() + " \n" +
"Date -  " + date.getText().toString()
 + " \n" + "Contact -  " +
contact.getText().toString(),
Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}
```

```
android:layout_alignParentLeft="true"
android:layout_alignParentStart="true"
android:layout_below="@+id/editText5"
android:layout_marginTop="62dp"
android:text="@string/submit"
android:textSize="16sp"
android:textStyle="normal|bold" />
</RelativeLayout>
```

AutoCompleteTextView

An AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing. The list of suggestions is displayed in drop down menu. The user can choose an item from there to replace the content of edit box with. It is the subclass of EditText class.

The list of suggestions is obtained from data adapter and appears only after a given number of characters defined by threshold.

Steps of creating AutoCompleteTextView

1. Create AutoCompleteTextView in the xml

```
<AutoCompleteTextView
     android:layout_width="match_parent "
     android:layout_height="wrap_content"
     android:id="@+id/course_list"
/>
```

2. Create the list in res/values/string.xml

```
<resources>
    <string-array name="course_array">
        <item>OOP in Java</item>
        <item>Data Structures in Java</item>
        <item>Advanced Progarmming using Java</item>
    </string-array>
</resources>
```

3. Get the reference of AutoCompleteTextView in java class

```
AutoCompleteTextView textView=( AutoCompleteTextView)findViewById(R.id.course_list);
String[] courses=getResources().getStringArray(R.array.courses_array);
ArrayAdapter<String> adapter=new
ArrayAdapter<String>(this.android.R.layout.simple_list_item_1,courses);
textView.setAdapter(adapter);
```

Retrieving the Value From AutoCompleteTextView In Java Class:

Below code retrieve the value from a AutoCompleteTextView in Java class.

```
AutoCompleteTextView simpleAutoCompleteTextView = (AutoCompleteTextView)
findViewById(R.id.course_list);
String AutoCompleteTextViewValue = simpleAutoCompleteTextView.getText().toString();
```

AutoCompleteTextView Attributes

In addition to the attributes described in EditText and TextView section AutoCompleteTextView has the following attributes:

- ✓ ***android:completionHint***- defines the hint displayed in the drop down menu.
- ✓ ***android:completionHintView***- defines the hint view displayed in the drop down menu.
- ✓ ***android:completionThreshold***- defines the number of characters that the user must type before completion suggestions are displayed in a drop down menu.
- ✓ ***android:dropDownAnchor***- is the View to anchor the auto-complete dropdown to.
- ✓ ***android:dropDownHeight***- specifies the basic height of the dropdown.
- ✓ ***android:dropDownHorizontalOffset***-The amount of pixels by which the drop down should be offset horizontally.
- ✓ ***android:dropDownSelector***- is the selector in a drop down list.
- ✓ ***android:dropDownVerticalOffset***-The amount of pixels by which the drop down should be offset vertically.
- ✓ ***android:dropDownWidth***- specifies the basic width of the dropdown.
- ✓ ***android:popupBackground***- sets the background.

Example: In the example of AutoCompleteTextView we display a auto complete text view with suggestion list which include country list. To fill the data in country list we implement an Array Adapter.

activity_main.xml

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

<RelativeLayout ...>

<AutoCompleteTextView
android:id="@+id/sACTV"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:background="#000"
android:hint="Enter Your Name Here"
android:padding="15dp"
android:textColorHint="#fff"
android:textStyle="bold|italic" />


</RelativeLayout>
```

MainActivity.java

```
import
android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.*;
import android.widget.*;
public class MainActivity extends
AppCompatActivity {
String[] countryNameList = {"Ethiopia",
"Eritrea", "Kenya", "Sudan", "England"};
    @Override
    protected void onCreate(Bundle
savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
 AutoCompleteTextView
simpleAutoCompleteTextView =
(AutoCompleteTextView)
findViewById(R.id.sACTV);
ArrayAdapter adapter = new ArrayAdapter(this,
android.R.layout.simple_list_item_1,
countryNameList);
simpleAutoCompleteTextView.setAdapter(adapter);
simpleAutoCompleteTextView.setThreshold(1);
simpleAutoCompleteTextView.setAdapter(adapter);
}}
```

CheckBox

In Android, CheckBox is a type of two state button either unchecked or checked in Android. Or you can say it is a type of on/off switch that can be toggled by the users. You should use checkbox when presenting a group of selectable options to users that are not mutually exclusive. CompoundButton is the parent class of CheckBoxclass.

CheckBox code in XML:

```
<CheckBox
android:id="@+id/simpleCheckBox"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Simple CheckBox"/>
```

You can check the current state of a check box programmatically by using isChecked() method. This method returns a Boolean value either true or false, if a check box is checked then it returns true otherwise it returns false. Below is an example code in which we checked the current state of a check box.

```
//initiate a check box
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);
//check current state of a check box (true or false)
Boolean checkBoxState = simpleCheckBox.isChecked();
```

CheckBox Attributes:

In addition to the above mentioned attributes CheckBox has the following attributes:

- ✓ ***android:autoText***- If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
- ✓ ***android:drawableBottom*** - is the drawable to be drawn below the text.
- ✓ ***android:drawableRight***- is the drawable to be drawn to the right of the text.
- ✓ ***android:editable***-If set, specifies that this TextView has an input method.
- ✓ ***android:text***- is the Text to display.
- ✓ ***android:background***- is a drawable to use as the background.
- ✓ ***android:contentDescription***- defines text that briefly describes content of the view.
- ✓ ***android:id***- supplies an identifier name for this view,
- ✓ ***android:onClick***- is the name of the method in this View's context to invoke when the view is clicked.
- ✓ ***android:visibility***- controls the initial visibility of the view.

Example: Below is the example of CheckBox in Android, in which we display five check boxes using background and other attributes we discusses earlier in this post. Every check box represents a different subject name and whenever you click on a check box then text of that checked check box is displayed in a toast.

activity_main.xml	MainActivity.java

`<?xml version="1.0" encoding="utf-8"?>`
`<RelativeLayout ...>`
`<TextView`
	`import android.graphics.Color;`
	`import`
	`android.support.v7.app.AppCompatActivity;`
	`import android.os.Bundle;`

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Select   Your   Programming
language:                                  "
android:textColor="#f00"
android:textSize="20sp"
android:textStyle="bold" />
    <LinearLayout
android:id="@+id/linearLayout"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_marginTop="30dp"
android:background="#e0e0e0"
android:orientation="vertical">
        <CheckBox
android:id="@+id/androidCheckBox"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:checked="false"
android:padding="20dp"
android:text="@string/android"
android:textColor="#44f"
android:textSize="20sp"
android:textStyle="bold|italic" />
        <CheckBox
android:id="@+id/javaCheckBox"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:checked="false"
android:padding="20dp"
android:text="@string/java"
android:textColor="#f44"
android:textSize="20sp"
android:textStyle="bold|italic" />
        <CheckBox
android:id="@+id/phpCheckBox"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:checked="false"
android:padding="20dp"
android:text="@string/php"
android:textColor="#444"
android:textSize="20sp"
android:textStyle="bold|italic" />
 </LinearLayout>

</RelativeLayout>
```

```
import android.view.*;
import android.widget.*;
public class MainActivity extends
AppCompatActivity implements
View.OnClickListener {
CheckBox android, java, php;
    @Override
    protected void onCreate(Bundle
savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
android = (CheckBox)
findViewById(R.id.androidCheckBox);
android.setOnClickListener(this);
java = (CheckBox)
findViewById(R.id.javaCheckBox);
java.setOnClickListener(this);
php = (CheckBox)
findViewById(R.id.phpCheckBox);
php.setOnClickListener(this);
    }
    @Override
    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.androidCheckBox:
                if (android.isChecked())
Toast.makeText(getApplicationContext(),
"Android", Toast.LENGTH_LONG).show();
                break;
            case R.id.javaCheckBox:
                if (java.isChecked())

Toast.makeText(getApplicationContext(),
"Java", Toast.LENGTH_LONG).show();
                break;
            case R.id.phpCheckBox:
                if (php.isChecked())

Toast.makeText(getApplicationContext(),
"PHP", Toast.LENGTH_LONG).show();
                break;
       }    }}
```

strings.xml

```
<resources>
<string
name="app_name">CheckBoxExample</string>
<string name="hello_world">Hello world!
</string><string
name="action_settings">Settings</string>
<string name="android">Android</string>
<string name="java">Java</string>
<string name="php">PHP</string>
</resources>
```

ToggleButton

ToggleButton allow the users to change the setting between two states like turn on/off your wifi, Bluetooth etc from your phone's setting menu. Since, Android 4.0 version (API level 14) there is an another kind of ToggleButton called Switch which provide the user slider control. You can learn more about it reading Switch tutorial. Android Switch and ToggleButton both are the subclasses of compoundButton class.

ToggleButton code in XML:

```
<ToggleButton
android:id="@+id/simpleToggleButton"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
```

How To Check Current State Of ToggleButton:

To check current state of a toggle button programmatically we use isChecked() method. This method returns a Boolean value either true or false. If a toggle button is checked then it returns true otherwise it returns false. Below is the code which checks the current state of a toggle button.

```
/*Add in Oncreate() funtion after setContentView()*/
ToggleButton simpleToggleButton = (ToggleButton)
findViewById(R.id.simpleToggleButton); // initiate a toggle button
Boolean ToggleButtonState = simpleToggleButton.isChecked(); // check current state of
a toggle button (true or false).
```

ToggleButton Attributes

In addition to the above mentioned attributes ToggleButton has the following attributes:

- ✓ ***android:disabledAlpha*** - is the alpha to apply to the indicator when disabled.
- ✓ ***android:textOff*** - is the text for the button when it is not checked.
- ✓ ***android:textOn*** - is the text for the button when it is checked.
- ✓ ***android:autoText*** - If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
- ✓ ***android:drawableBottom*** - is the drawable to be drawn below the text.
- ✓ ***android:drawableRight*** - is the drawable to be drawn to the right of the text
- ✓ ***android:editable***-If set, specifies that this TextView has an input method.
- ✓ ***android:text*** - is the Text to display. Inherited from android.view.View Class:
- ✓ ***android:background*** - is a drawable to use as the background.
- ✓ ***android:contentDescription*** - defines text that briefly describes content of the view.
- ✓ ***android:id*** - is supplies an identifier name for this view,
- ✓ ***android:onClick***- is the name of the method in this View's context to invoke when the view is clicked.
- ✓ ***android:visibility*** - controls the initial visibility of the view.

- ✓ ***android:checked*:** checked is an attribute of toggle button used to set the current state of a toggle button. The value should be true or false where true shows the checked state and false shows unchecked state of a toggle button.
- ✓ ***android:drawableBottom, drawableTop, drawableRight And drawableLeft:*** These attribute draw the drawable below, top, right and left of the text of ToggleButton

Example: Below is the example of ToggleButton in Android Studio. In this example we display two toggle button with background and one "submit" button using attributes discussed earlier in this post. Whenever user click on the submit button, the current state of both toggle button's is displayed in a Toast.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout . . . >
    <LinearLayout
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center_horizontal"
android:orientation="horizontal">
        <ToggleButton
android:id="@+id/simpleToggleButton1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center_horizontal"
android:checked="false"
android:drawablePadding="20dp"
android:drawableRight="@drawable/ic_launche
r"
android:textColor="#000" />
        <ToggleButton
android:id="@+id/simpleToggleButton2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center_horizontal"
android:layout_marginLeft="50dp"
android:checked="true"
android:drawableLeft="@drawable/ic_launcher
"
android:drawablePadding="20dp"
android:textColor="#000" />
    </LinearLayout>
    <Button
android:id="@+id/submitButton"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="50dp"
android:background="#0f0"
android:padding="10dp"
android:text="Submit"
android:textColor="#fff"
android:textSize="20sp"
android:textStyle="bold" />
</LinearLayout>
```

MainActivity.java

```
import
android.support.v7.app.AppCompatActivit
y;
import android.os.Bundle;
import android.view.*;
import android.widget.*;
public class MainActivity extends
AppCompatActivity {
ToggleButton simpleToggleButton1,
simpleToggleButton2;
    Button submit;
    @Override
    protected void onCreate(Bundle
savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
simpleToggleButton1 = (ToggleButton)
findViewById(R.id.simpleToggleButton1);
simpleToggleButton2 = (ToggleButton)
findViewById(R.id.simpleToggleButton2);
submit = (Button)
findViewById(R.id.submitButton);
submit.setOnClickListener(new
View.OnClickListener() {
            @Override
 public void onClick(View view) {
  String status = "ToggleButton1 : " +
simpleToggleButton1.getText() + "\n" +
"ToggleButton2 : " +
simpleToggleButton2.getText();
Toast.makeText(getApplicationContext(),
status, Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

RadioButton

In Android, RadioButton are mainly used together in a RadioGroup. In RadioGroup checking the one radio button out of several radio button added in it will automatically unchecked all the others. It means at one time we can checked only one radio button from a group of radio buttons which belong to same radio group. The most common use of radio button is in Quiz Android App code. RadioButon is a two state button that can be checked or unchecked. If a radio button is unchecked then a user can check it by simply clicking on it. Once a RadiaButton is checked by user it can't be unchecked by simply pressing on the same button. It will automatically unchecked when you press any other RadioButton within same RadioGroup.

RadioGroup is a widget used in Android for the grouping of radio buttons and provide the feature of selecting only one radio button from the set. When a user try to select any other radio button within same radio group the previously selected radio button will be automatically unchecked.

RadioGroup And RadioButton code in XML:

```
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
        <RadioButton
        android:id="@+id/simpleRadioButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
        <RadioButton
        android:id="@+id/simpleRadioButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</RadioGroup>
```

Checking Current State Of Radio Button:

You can check the current state of a radio button programmatically by using isChecked() method. This method returns a Boolean value either true or false. if it is checked then returns true otherwise returns false. Below is an example code with explanation in which we checked the current state of a radio button.

```
/*Add in Oncreate() funtion after setContentView()*/
RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton);
// initiate a radio button
Boolean RadioButtonState = simpleRadioButton.isChecked(); // check current state of a
radio button (true or false).
```

RadioButton Attributes: RadioButton supports the attributes mentioned for most UI widgets.

Example: Below is the example of Radiobutton in Android where we display true and false radio buttons with background and other attributes. The radio buttons are used to select your answer for the given question with one "submit" button.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout . . . >
    <LinearLayout
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:background="#e0e0e0"
android:orientation="vertical">
        <TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Java is the only
programming language used to develop a
mobile app"
android:textColor="#000"
android:textSize="20sp"
android:textStyle="bold" />
        <RadioGroup
android:layout_width="wrap_content"
android:layout_height="wrap_content">
       <RadioButton
android:id="@+id/rbtrue"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginLeft="20dp"
android:layout_marginTop="10dp"
android:checked="true"
android:text="@string/strtrue"
android:textColor="#154"
android:textSize="20sp"
android:textStyle="bold" />
     <RadioButton
android:id="@+id/rbfalse"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginLeft="20dp"
android:layout_marginTop="10dp"
android:checked="false"
android:text="@string/strfalse"
android:textColor="#154"
android:textSize="20sp"
android:textStyle="bold" />
          </RadioGroup>
        <Button
android:id="@+id/submitButton"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_margin="20dp"
android:background="#0f0"
android:padding="10dp"
```

MainActivity.java

```
import
android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.*;
import android.widget.*;
public class MainActivity extends
AppCompatActivity {
RadioButton bttrue, btfalse;
    String selectedanswer;
    Button submit;
    @Override
protected void onCreate(Bundle
savedInstanceState) {
 super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
 bttrue = (RadioButton)
findViewById(R.id.rbtrue);
 btfalse = (RadioButton)
findViewById(R.id.rbfalse);
findViewById(R.id.sheamus);
 submit = (Button)
findViewById(R.id.submitButton);
 submit.setOnClickListener(new
View.OnClickListener() {
            @Override
 public void onClick(View v) {
 if (bttrue.isChecked()) {
   selectedanswer =
bttrue.getText().toString(); }
 else if (btfalse.isChecked()) {
 selectedanswer =
btfalse.getText().toString();           }
Toast.makeText(getApplicationContext(),
selectedanswer, Toast.LENGTH_LONG).show();
}        });
    }}
```

strings.xml

```
<resources>
    <string
name="app_name">RadioButtonExample</string>
<string name="hello_world">Hello
world!</string><string
name="action_settings">Settings</string>
<string name="strtrue">True</string>
<string name="strfalse">False</string>
</resources>
```

```
android:text="Submit"
android:textColor="#fff"
android:textSize="20sp"
android:textStyle="bold" />
    </LinearLayout>

</LinearLayout>
```

Assignment1

2.4. Android - Event Handling

Events are a useful way to collect data about a user's interaction with interactive components of Applications. Like button presses or screen touch etc. The Android framework maintains an event queue as first-in, first-out (FIFO) basis. You can capture these events in your program and take appropriate action as per requirements.

There are three concepts related to Android Event Management −

- ✓ **Event Listeners** − An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.
- ✓ **Event Listeners Registration** − Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.
- ✓ **Event Handlers** − When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

Event Listeners & Event Handlers

Event Handler	Event Listener & Description
onClick()	OnClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.
onLongClick()	OnLongClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	OnFocusChangeListener() This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.
onKey()	OnFocusChangeListener() This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.

onTouch()	OnTouchListener() This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.
onMenuItemClick()	OnMenuItemClickListener() This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.
onCreateContextMenu()	onCreateContextMenuItemListener() This is called when the context menu is being built(as the result of a sustained "long click)

There are many more event listeners available as a part of **View** class like OnHoverListener, OnDragListener etc which may be needed for your application. So refer the official documentation for Android application development in for further information.

Event Listeners Registration

Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event. Though there are several tricky ways to register your event listener for any event, but some of them are explained below.

- ✓ Using an Anonymous Inner Class
- ✓ Activity class implements the Listener interface.
- ✓ Using Layout file activity_main.xml to specify event handler directly.

Event listeners registration using an anonymous inner class

Here you will create an anonymous implementation of the listener and will be useful if each class is applied to a single control only and you have advantage to pass arguments to event handler. In this approach event handler methods can access private data of Activity. No reference is needed to call to Activity. But if you applied the handler to more than one control, you would have to cut and paste the code for the handler and if the code for the handler is long, it makes the code harder to maintain.

Steps

1. Create an Android application and name it as *EventDemo* under a package *com.example.eventdemo* as explained in the *Hello World Example* chapter.
2. Modify *java/MainActivity.java* file to add click event listeners and handlers for a button defined.
3. Modify the default content of *res/layout/activity_main.xml* file to include Android UI controls.
4. Define required constants in *res/values/strings.xml* file
5. Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.eventdemo/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```
package com.example.eventdemo;
import android.os.Bundle;

import android.app.Activity;
import android.view.*;
import android.widget.*;
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //--- find the button---
        Button sButton = (Button) findViewById(R.id.button_s);
        // -- register click event with the button ---
        sButton.setOnClickListener(new View.OnClickListener() {
           public void onClick(View v) {
               // --- find the text view --
               TextView txtView = (TextView) findViewById(R.id.text_id);
               // -- change text size --
               txtView.setTextSize(14);
           }
        });

       }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

Following will be the content of **res/layout/activity_main.xml** file:

```
<?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" >
    <Button
    android:id="@+id/button_s"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:text="@string/button_small"/>

    <TextView
    android:id="@+id/text_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:capitalize="characters"
    android:text="@string/hello_world" />
</LinearLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">EventDemo</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
   <string name="button_small">Small Font</string>
</resources>
```

Registration using the activity implements listener interface

Here your Activity class implements the Listener interface and you put the handler method in the main Activity and then you call ***setOnClickListener(this)***. This approach is fine if your application has only a single control of that Listener type otherwise you will have to do further programming to check which control has generated event. Second you cannot pass arguments to the Listener so, again, works poorly for multiple controls.

Following are the simple steps to show how we will make use of separate Listener class to register and capture click event. Similar way you can implement your listener for any other required event type.

Following are the simple steps to show how we will implement Listener class to register and capture click event. Similar way you can implement your listener for any other required event type.

Steps

1. We do not need to create this application from scratch, so let's make use of above created Android application *EventDemo*.
2. Modify *java/MainActivity.java* file to add click event listeners and handlers for a button defined.
3. We are not making any change in *res/layout/activity_main.xml*, it will remain as shown above.
4. We are also not making any change in *res/values/strings.xml* file, it will also remain as shown above.
5. Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file

src/com.example.eventdemo/MainActivity.java. This filecan include each of the fundamental lifecycle methods.

```
package com.example.eventdemo;

import android.os.Bundle;
import android.app.Activity;
import android.view.*;
import android.widget.*;
public class MainActivity extends Activity implements OnClickListener {

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

        //--- find the button---
        Button sButton = (Button) findViewById(R.id.button_s);

        // -- register click event with the button ---
        sButton.setOnClickListener(this);
}

    //--- Implement the OnClickListener callback
    public void onClick(View v) {
       if(v.getId() == R.id.button_s)
       {
            // --- find the text view --
            TextView txtView = (TextView) findViewById(R.id.text_id);
            // -- change text size --
            txtView.setTextSize(14);
```

```
            return;
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

Registration using layout file activity_main.xml

Here you put your event handlers in Activity class without implementing a Listener interface or call to any listener method. Rather you will use the layout file (activity_main.xml) to specify the handler method via the ***android:onClick*** attribute for click event. You can control click events differently for different control by passing different event handler methods.

The event handler method must have a void return type and take a View as an argument. However, the method name is arbitrary, and the main class need not implement any particular interface.

This approach does not allow you to pass arguments to Listener and for the Android developers it will be difficult to know which method is the handler for which control until they look into activity_main.xml file. Second, you cannot handle any other event except click event using this approach.

Following are the simple steps to show how we can make use of layout file Main.xml to register and capture click event.

Step

1. We do not need to create this application from scratch, so let's make use of above created Android application *EventDemo*.
2. Modify *java/MainActivity.java* file to add click event listeners and handlers for the two buttons defined.
3. Modify layout file *res/layout/activity_main.xml*, to specify event handlers for the two buttons.
4. We are also not making any change in *res/values/strings.xml* file, it will also remain as shown above.
5. Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file

src/com.example.eventdemo/MainActivity.java. This file can include each of the fundamental lifecycle methods.

```
package com.example.eventdemo;
import android.os.Bundle;
import android.app.Activity;
import android.view.*;
import android.widget.*;

public class MainActivity extends Activity{

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
    }

    //--- Implement the event handler for the button.
    public void doSmall(View v)  {
       // --- find the text view --
       TextView txtView = (TextView) findViewById(R.id.text_id);
       // -- change text size --
       txtView.setTextSize(14);
       return;
   }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

Following will be the content of **res/layout/activity_main.xml** file. Here we have to add **android:onClick="methodName"** for the button, which will register given method names as click event handlers.

```
<?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" >
    <Button
    android:id="@+id/button_s"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:text="@string/button_small"
    android:onClick="doSmall"/>
   <TextView
    android:id="@+id/text_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:capitalize="characters"
    android:text="@string/hello_world" />

</LinearLayout>
```

2.5. Styles and Themes

If you already know about Cascading Style Sheet (CSS) in web design then to understand Android Style also works very similar way. There are number of attributes associated with each Android widget which you can set to change your application look and feel. A style can specify properties such as height, padding, font color, font size, background color, and much more.

You can specify these attributes in Layout file as follows:

```
<?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"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:capitalize="characters"
   android:textColor="#00FF00"
```

```
   android:typeface="monospace"
   android:text="@string/hello_world" />
</LinearLayout>
```

But this way we need to define style attributes for every attribute separately which is not good for source code maintenance point of view. So we work with styles by defining them in separate file as explained below.

Defining Styles

A style is defined in an XML resource that is separate from the XML that specifies the layout. This XML file resides under res/values/ directory of your project and will have <resources> as the root node which is mandatory for the style file. The name of the XML file is arbitrary, but it must use the .xml extension.

You can define multiple styles per file using <style> tag but each style will have its name that uniquely identifies the style. Android style attributes are set using <item> tag as shown below:

```
<?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>
```

The value for the <item> can be a keyword string, a hex color, a reference to another resource type, or other value depending on the style property.

Using Styles

Once your style is defined, you can use it in your XML Layout file using style attribute as follows:

```
<?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

Android supports style Inheritance in very much similar way as cascading style sheet in web design. You can use this to inherit properties from an existing style and then define only the properties that you want to change or add.

It is simple, to create a new style LargeFont that inherits the CustomFontStyle style defined above, but make the font size big, you can author the new style like this:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="CustomFontStyle.LargeFont">
      <item name="android:textSize">20ps</item>
   </style>
</resources>
```

You can reference this new style as @style/CustomFontStyle.LargeFont in your XML Layout file. You can continue inheriting like this as many times as you'd like, by chaining names with periods. For example, you can extend FontStyle.LargeFont to be Red, with:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="CustomFontStyle.LargeFont.Red">
      <item name="android:textColor">#FF0000</item>/>
</style>
</resources>
```

This technique for inheritance by chaining together names only works for styles defined by your own resources.

You can't inherit Android built-in styles this way. To reference an Android built-in style, such as TextAppearance, you must use the parent attribute as shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="CustomFontStyle" parent="@android:style/TextAppearance">
      <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>
```

Android Themes

Hope you understood the concept of Style, so now let's try to understand what is a Theme. A theme is nothing but an Android style applied to an entire Activity or application, rather than an individual View.

Thus, when a style is applied as a theme, every View in the Activity or application will apply each style property that it supports. For example, you can apply the same CustomFontStyle style as a theme for an Activity and then all text inside that Activity will have green monospace font.

To set a theme for all the activities of your application, open the AndroidManifest.xml file and edit the<application> tag to include the android:theme attribute with the style name. For example:

```
<application android:theme="@style/CustomFontStyle">
```

But if you want a theme applied to just one Activity in your application, then add the android:theme attribute to the <activity> tag only. For example:

```
<activity android:theme="@style/CustomFontStyle">
```

There are number of default themes defined by Android which you can use directly or i nherit them using parent attribute as follows:

```
<style name="CustomTheme" parent="android:Theme.Light">
    ...
</style>
```

Default Styles & Themes

The Android platform provides a large collection of styles and themes that you can use in your applications. You can find a reference of all available styles in the R.style class. To use the styles listed here, replace all underscores in the style name with a period. For example, you can apply the Theme_NoTitleBar theme with "@android:style/Theme.NoTitleBar". You can see the following source code for Android styles and themes:

- Android Styles (styles.xml)
- Android Themes (themes.xml)

Further Reading

- Read about the layouts which are not mentioned in this course.
- Read about the other UI widgets which are not covered in this course such as Spinner, RatingBar, ProgressBar, DatePicker, TimePicker, TextSwitcher, ImageSwitcher, AlertDialog, AdapterViewFlipper etc.
-

Chapter 3

3. Activity, Fragments, Intent and Services

3.1. Activity

In android, **Activity** represents a single screen with a user interface (UI) of an application and it will act as an entry point for users to interact with an app.

Android apps can contain multiple screens and each screen of the application is an extension of Activity class. By using activities, we can place all our android application UI components in a single screen.

From the multiple activities in android app, one activity can be marked as a **main activity** and that is the first screen to appear when we launch the application. In android app each activity can start another activity to perform different actions based on our requirements.

For example, a contacts app which is having a multiple activities, in that the main activity screen will show a list of contacts and from the main activity screen we can launch other activities that provides a screens to perform a tasks like add a new contact and search for the contacts. All these activities in contact app are loosely bound to other activities but will work together to provide a better user experience.

Generally, in android there is a minimal dependency between the activities in an app. To use activities in application we need to register those activities information in our app's manifest file (**AndroidMainfest.xml**) and need to manage activity life cycle properly.

To use activities in our application we need to define activities with required attributes in manifest file (**AndroidMainfest.xml**) like as shown below

```
<?xml version="1.0" encoding="utf-8"?>
<manifest …..>
    <application …..>
        <activity android:name=".MainActivity" >
          …….
          …….
        </activity>
     …….
</application>
</manifest>
```

The activity attribute **android:name** will represent the name of class and we can also add multiple attributes like icon, label, theme, permissions, etc. to an activity element based on our requirements.

In android application, activities can be implemented as a subclass of **Activity** class like as shown below.

```
import android.app.Activity;
import android.os.Bundle;
public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Android Activity Lifecycle

The activities in our android application will go through different stages in their life cycle. In android, **Activity** class have a 7 callback methods like onCreate(), onStart(), onPause(), onRestart(), onResume(), onStop() and onDestroy() to describe how the activity will behave at different stages.

- ✓ onCreate() — Called when the activity is first created
- ✓ onStart() — Called when the activity becomes visible to the user
- ✓ onResume() — Called when the activity starts interacting with the user
- ✓ onPause() — Called when the current activity is being paused and the previous activity is being resumed
- ✓ onStop() — Called when the activity is no longer visible to the user
- ✓ onDestroy() — Called before the activity is destroyed by the system (either manually or by the system to conserve memory)
- ✓ onRestart() — Called when the activity has been stopped and is restarting again

By default, the activity created for you contains the onCreate() event. Within this event handler is the code that helps to display the UI elements of your screen. The following figure shows life cycle of an activity.

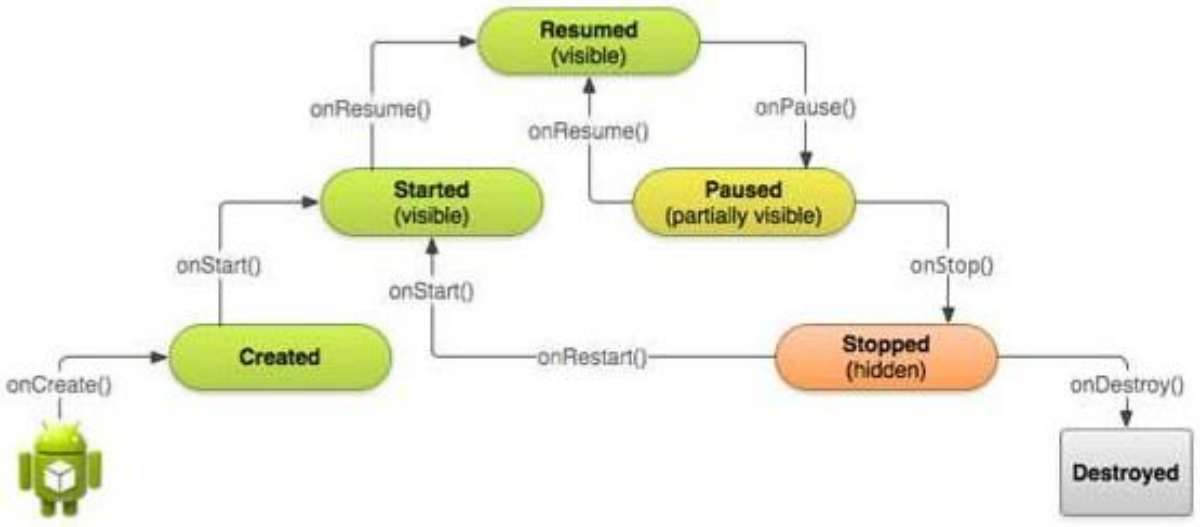


Figure 3.1 Life cycle of an activity

The best way to understand the various stages of an activity is to create a new project, implement the various events, and then subject the activity to various user interactions.

Understanding the Life Cycle of an Activity

1. Create a new Android project and name it **Activity101**.
2. In the Activity101Activity.java file, add the following statements in bold:

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
public class Activity101Activity extends Activity {
    String tag = “Lifecycle”;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Log.d(tag, “In the onCreate() event”);
    }
       public void onStart(){
                super.onStart();
                Log.d(tag, “In the onStart() event”);   }
       public void onRestart() {
                super.onRestart();
                Log.d(tag, “In the onRestart() event”);    }
       public void onResume(){
                super.onResume();
                Log.d(tag, “In the onResume() event”);    }
       public void onPause(){
                super.onPause();
                Log.d(tag, “In the onPause() event”);    }
       public void onStop(){
                super.onStop();
                Log.d(tag, “In the onStop() event”);    }
       public void onDestroy() {
                super.onDestroy();
                Log.d(tag, “In the onDestroy() event”);    }
}
```

3. Press F11 to debug the application on the Android emulator.
4. When the activity is first loaded, you should see something very similar to the following in the LogCat window (click the Debug perspective):

```
11-16 06:25:59.396: D/Lifecycle(559): In the onCreate() event
11-16 06:25:59.396: D/Lifecycle(559): In the onStart() event
11-16 06:25:59.396: D/Lifecycle(559): In the onResume() event
```

5. If you click the Back button on the Android emulator, the following is printed:

```
11-16 06:29:26.665: D/Lifecycle(559): In the onPause() event
11-16 06:29:28.465: D/Lifecycle(559): In the onStop() event
11-16 06:29:28.465: D/Lifecycle(559): In the onDestroy() event
```

6. Click the Home button and hold it there. Click the Activities icon and observe the following:

```
11-16 06:31:08.905: D/Lifecycle(559): In the onCreate() event
11-16 06:31:08.905: D/Lifecycle(559): In the onStart() event
11-16 06:31:08.925: D/Lifecycle(559): In the onResume() event
```

7. Click the Phone button on the Android emulator so that the activity is pushed to the background. Observe the output in the LogCat window:

```
11-16 06:32:00.585: D/Lifecycle(559): In the onPause() event
11-16 06:32:05.015: D/Lifecycle(559): In the onStop() event
```

8. Notice that the onDestroy() event is not called, indicating that the activity is still in memory. Exit the phone dialer by clicking the Back button. The activity is now visible again. Observe the output in the LogCat window:

```
11-16 06:32:50.515: D/Lifecycle(559): In the onRestart() event
11-16 06:32:50.515: D/Lifecycle(559): In the onStart() event
11-16 06:32:50.515: D/Lifecycle(559): In the onResume() event
```

The onRestart() event is now fi red, followed by the onStart() and onResume() methods.

How It Works

As you can see from this simple example, an activity is destroyed when you click the Back button. This is crucial to know, as whatever state the activity is currently in will be lost; hence, you need to write additional code in your activity to preserve its state when it is destroyed. At this point, note that the onPause() method is called in both scenarios — when an activity is sent to the background, as well as when it is killed when the user presses the Back button.

When an activity is started, the onStart() and onResume()methods are always called, regardless of whether the activity is restored from the background or newly created. When an activity is created for the first time, the onCreate() method is called.

From the preceding example, you can derive the following guidelines:

- ✓ Use the onCreate() method to create and instantiate the objects that you will be using in your application.
- ✓ Use the onResume() method to start any services or code that needs to run while your activity is in the foreground.
- ✓ Use the onPause() method to stop any services or code that does not need to run when your activity is not in the foreground.
- ✓ Use the onDestroy() method to free up resources before your activity is destroyed.

Note: *Even if an application has only one activity and the activity is killed, the application will still be running in memory.*

3.2. Fragments

In the previous section you learned what an activity is and how to use it. In a small-screen device (such as a smartphone), an activity typically fills the entire screen, displaying the various views that make up the user interface of an application. The activity is essentially a container for views.

However, when an activity is displayed in a large-screen device, such as on a tablet, it is somewhat out of place. Because the screen is much bigger, all the views in an activity must be arranged to make full use of

the increased space, resulting in complex changes to the view hierarchy. A better approach is to have "mini-activities," each containing its own set of views. During runtime, an activity can contain one or more of these mini-activities, depending on the screen orientation in which the device is held. In Android 3.0 and later, these mini-activities are known as **fragments.**

Think of a fragment as another form of activity. You create fragments to contain views, just like activities. Fragments are always embedded in an activity. For example, Figure 3.2.1 shows two fragments. Fragment 1 might contain a ListView showing a list of book titles. Fragment 2 might contain some TextViews and ImageViews showing some text and images.

Now imagine the application is running on an Android tablet in portrait mode (or on an Android smartphone). In this case, Fragment 1 may be embedded in one activity, while Fragment 2 may be embedded in another activity (see Figure 3.2.2). When users select an item in the list in Fragment 1,Activity 2 will be started.

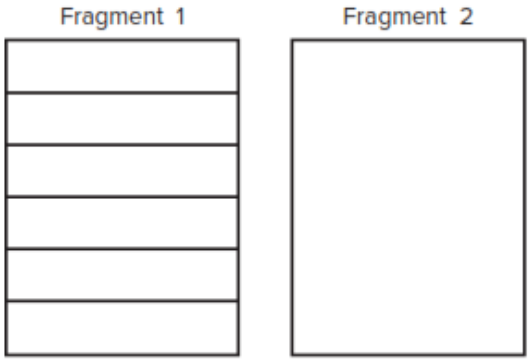


Figure 3.2.1 Fragments in an activity

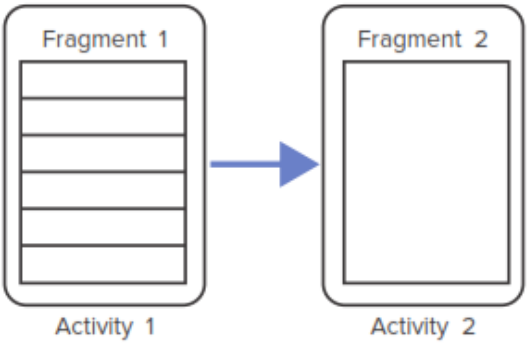


Figure 3.2.2 Fragments in different activities

If the application is now displayed in a tablet in landscape mode, both fragments can be embedded within a single activity, as shown in Figure 3.2.3.

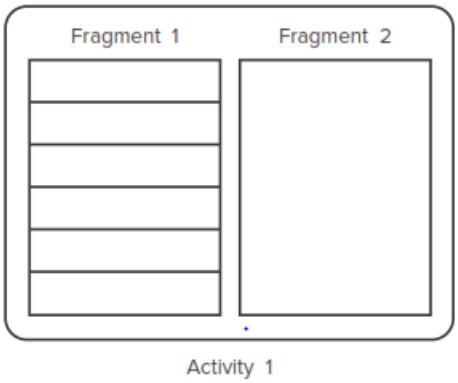


Figure 3.2.3 Fragments in landscape mode

From this discussion, it becomes apparent that fragments present a versatile way in which you can create the user interface of an Android application.

Fragments form the atomic unit of your user interface, and they can be dynamically added (or removed) to activities in order to create the best user experience possible for the target device.

Android Fragment Life Cycle

The following is pictorial representation of android fragment life cycle while its activity is running.

The following are the list of methods which will perform during the lifecycle of fragment in android applications.

Method	Description
onAttach()	It is called when the fragment has been associated with an activity.
onCreate()	It is used to initialize the fragment.
onCreateView()	It is used to create a view hierarchy associated with the fragment.
onActivityCreated()	It is called when the fragment activity has been created and the fragment view hierarchy instantiated.
onStart()	It is used to make the fragment visible.
onResume()	It is used to make the fragment visible in an activity.
onPause()	It is called when fragment is no longer visible and it indicates that the user is leaving the fragment.
onStop()	It is called to stop the fragment using onStop() method.
onDestoryView()	The view hierarchy which associated with the fragment is being removed after executing this method.
onDestroy()	It is called to perform a final clean up of the fragments state.
onDetach()	It is called immediately after the fragment disassociated from the activity.

Example:

Following is the example of creating a two fragments, two buttons and showing the respective fragment when click on button in android application.

1. Create a new android application using android studio and give name as **Fragments**.
2. Now we need to create our own custom fragment layout files (**listitems_info.xml**, **details_info.xml**) in **\res\layout** path to display those fragments in main layout for that right click on your layout folder → Go to **New** → select **Layout resource file** and give name as **listitems_info.xml**. Once we create a new file **listitems_info.xml**, open it and write the code like as shown below

listitems_info.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">
   <ListView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@android:id/list" />
</LinearLayout>
```

3. Same way create another file **details_info.xml**, open it and write the code like as shown below

details_info.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#0079D6">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#ffffff"
        android:layout_marginTop="200px"
        android:layout_marginLeft="200px"
        android:id="@+id/Name"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="200px"
        android:textColor="#ffffff"
        android:id="@+id/Location"/>
</LinearLayout>
```

4. Now we need to create our own custom fragment class files (**ListMenuFragment.java**, **DetailsFragment.java**) in **\java\com.example.fragmentsexample** path to bind and display data in fragments for that right click on your application folder →Go to New → select **Java Class** and give name as **DetailsFragment.java**. Once we create a new file **DetailsFragment.java**, open it and write the code like as shown below

DetailsFragment.java

```
import android.app.Fragment;
import android.os.Bundle;
```

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
public class DetailsFragment extends Fragment {
    TextView name,location;
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View view = inflater.inflate(R.layout.details_info, container, false);
        name = (TextView)view.findViewById(R.id.Name);
        location = (TextView)view.findViewById(R.id.Location);
        return view;
    }
    public void change(String uname, String ulocation){
        name.setText(uname);
        location.setText(ulocation);
    }
}
```

5. If you observe above code we extended class with Fragment and used **LayoutInflater** to show the details of fragment. We defined a function **change()** to change the text in textview. Same way create another file **ListMenuFragment.java**, open it and write the code like as shown below

ListMenuFragment.java

```
import android.app.ListFragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ListView;
public class ListMenuFragment extends ListFragment {
    String[] users = new String[] { "Abebe","Samson","Kirubel","Helen" };
    String[] location = new String[]{"Gondar","Bahir Dar","Addis Ababa","Debre
Markos"};
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View view =inflater.inflate(R.layout.listitems_info, container, false);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(getActivity(),
                android.R.layout.simple_list_item_1, users);
        setListAdapter(adapter);
        return view;
    }
```

```
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        DetailsFragment txt =
(DetailsFragment)getFragmentManager().findFragmentById(R.id.fragment2);
        txt.change("Name: "+ users[position],"Location : "+ location[position]);
        getListView().setSelector(android.R.color.holo_blue_dark);
    }
}
```

6. If you observe above code we extended our class using **ListFragment** and we defined two arrays of strings users, location which contains names and locations. We defined **onListItemClick** event to update the name and location in **DetailsFragment** based on the list item which we clicked. Now we need to display our fragments horizontally side by side in main layout for that open activity_main.xml file and write code like as shown below

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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:orientation="horizontal"
    tools:context="com.example.fragmentsexample.MainActivity">
    <fragment
        android:layout_height="match_parent"
        android:layout_width="350px"
        class="com.example.fragmentsexample.ListMenuFragment"
        android:id="@+id/fragment"/>
    <fragment
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        class="com.example.fragmentsexample.DetailsFragment"
        android:id="@+id/fragment2"/>
</LinearLayout>
```

We are not going to make any modifications for our main activity file (**MainActivity.java**) and manifest file (**AndroidMainfest.xml**).

3.3. Intents

Android uses Intent for communicating between the components (such as activities, services, broadcast receivers and content providers) of an Application and also from one application to another application.

For example: Intent facilitates you to redirect your activity to another activity on occurrence of any event. By calling, startActivity() you can perform this task.

```
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
startActivity(intent);
```

In the above example, foreground activity is getting redirected to another activity i.e. SecondActivity.java. getApplicationContext() returns the context for your foreground activity.

In android, Intents are the objects of **android.content.Intent** type and intents are mainly useful to perform following things.

Component	Description
Starting an Activity	By passing an **Intent** object to `startActivity()` method we can start a new Activity or existing Activity to perform required things.
Starting a Service	By passing an **Intent** object to `startService()` method we can start a new Service or send required instructions to an existing Service.
Delivering a Broadcast	By passing an **Intent** object to `sendBroadcast()` method we can deliver our messages to other app broadcast receivers.

Types of Intents:

There are two types of intents: **Explicit Intent** and **Implicit Intent**

Explicit Intent:

Explicit Intents are used to connect the application internally. In Explicit we use the name of component which will be affected by Intent. For Example: If we know class name then we can navigate the app from One Activity to another activity using Intent. In the similar way we can start a service to download a file in background process. Explicit Intent works internally within an application to perform navigation and data transfer. The below given code snippet will help you understand the concept of Explicit Intents

```
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
startActivity(intent);
```

Here **SecondActivity** is the JAVA class name where the activity will now be navigated.

Implicit Intent:

In Implicit Intents we do need to specify the name of the component. We just specify the Action which has to be performed and further this action is handled by the component of another application. The basic example of implicit Intent is to open any web page.

Let's take an example to understand Implicit Intents more clearly. We have to open a website using intent in your application. See the code snippet given below

```
Intent intentObj = new Intent(Intent.ACTION_VIEW);
intentObj.setData(Uri.parse("https://www.google.com"));
```

```
startActivity(intentObj);
```

Unlike Explicit Intent you do not use any class name to pass through Intent(). In this example we have just specified an action. Now when we run this code then Android will automatically start your web browser and it will open Google home page.

Example:

The example will show you both implicit and explicit Intent together. Let's implement Intent for a very basic use. In the below example we will Navigate from one Activity to another and open a web homepage of Google using Intent.

Create a project in Android Studio and named it "Intents". Make an activity, which would consists Java file; **MainActivity.java** and an xml file for User interface which would be **activity_main.xml**

Step 1: Let's design the UI of **activity_main.xml**:

- ✓ First design the text view displaying basic details of the App
- ✓ Second design the two button of Explicit Intent Example and Implicit Intent Example

Below is the complete code of **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="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="If you click on Explicit example we will navigate to second
        activity within App and if you click on Implicit example Google Homepage will
        open in Browser"
        android:id="@+id/textView2"
        android:clickable="false"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="42dp"
        android:background="#3e7d02"
        android:textColor="#ffffff" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="Explicit Intent Example"
        android:id="@+id/explicit_Intent"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="147dp" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Implicit Intent Example"
        android:id="@+id/implicit_Intent"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Step 2: Design the UI of second activity **activity_second.xml**

Now let's design UI of another activity where user will navigate after he click on Explicit Example button. Go to layout folder, create a new activity and name it activity_second.xml.

- ✓ In this activity we will simply use TextView to tell user he is now on second activity.

Below is the complete code of **activity_second.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<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="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:background="#CCEEAA"
    tools:context="com.example.android.intents.SecondActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="This is Second Activity"
        android:id="@+id/textView"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Step 3: Implement onClick event for Implicit and Explicit Button inside **MainActivity.java**

Now we will use setOnClickListener() method to implement OnClick event on both the button. Implicit button will open AbhiAndroid.com homepage in browser and Explicit button will move to **SecondActivity.java**.

Below is the complete code of **MainActivity**.java

```
import android.content.Intent;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
public class MainActivity extends AppCompatActivity {
    Button explicit_btn, implicit_btn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        explicit_btn = (Button)findViewById(R.id.explicit_Intent);
        implicit_btn = (Button) findViewById(R.id.implicit_Intent);
        //implement Onclick event for Explicit Intent
        explicit_btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new  Intent(getBaseContext(), SecondActivity.class);
                startActivity(intent);
            }
        });
        //implement onClick event for Implicit Intent
        implicit_btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(Intent.ACTION_VIEW);
                intent.setData(Uri.parse("https://www.google.com"));
                startActivity(intent);
            }
        });
    } }
```

Step 4: Create A New JAVA class name **SecondActivity**

Now we need to create another SecondActivity.java which will simply open the layout of **activity_second.xml**. Also we will use Toast to display message that he is on second activity.

Below is the complete code of SecondActivity.java:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;
public class SecondActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        Toast.makeText(getApplicationContext(), "We are moved to second Activity”,
Toast.LENGTH_LONG).show();
    }
}
```

Step 5: Manifest file:

Make sure Manifest file has both the **MainActivity** and **SecondActivity** listed it. Also here **MainActivity** is our main activity which will be launched first. So make sure intent-filter is correctly added just below **MainActivity**.

Below is the code of Manifest file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.intents" >
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SecondActivity" >
        </activity>
    </application>
</manifest>
```

Testing:

- ✓ Now run the above program in your Emulator.
- ✓ First Click on Explicit Intent Example. The SecondActivity will be open within the App.
- ✓ Now go back in Emulator and click on Implicit Intent Example. The google.com homepage will open in Browser (make sure you have internet)

Benefits of Intent in Android

Android uses Intents for facilitating communication between its components like Activities, Services and Broadcast Receivers.

For an Activity: Every screen in Android application represents an activity. To start a new activity you need to pass an Intent object to startActivity() method. This Intent object helps to start a new activity and passing data to the second activity.

For Services: Services work in background of an Android application and it does not require any user Interface. Intents could be used to start a Service that performs one-time task(for example: Downloading some file) or for starting a Service you need to pass Intent to startService() method.

For Broadcast Receivers: There are various messages that an app receives; these messages are called as Broadcast Receivers. (For example, a broadcast message could be initiated to intimate that the file downloading is completed and ready to use). Android system initiates some broadcast message on several events, such as System Reboot, Low Battery warning message etc.

For Android Applications: Whenever you need to navigate to another activity of your app or you need to send some information to next activity then we can always prefer to Intents for doing so.

3.4. Android Services

Android service is a component that is *used to perform operations on the background* such as playing music, handle network transactions, interacting content providers etc. It doesn't have any UI (user interface). The service runs in the background indefinitely even if application is destroyed. Moreover, service can be bounded by a component to perform interactivity and inter process communication (IPC). The android.app.Service is subclass of ContextWrapper class.

Life Cycle of Android Service

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

- ✓ Started
- ✓ Bound

1) **Started Service**

A service is started when component (like activity) calls **startService()** method, now it runs in the background indefinitely. It is stopped by **stopService()** method. The service can stop itself by calling the **stopSelf()** method.

2) **Bound Service**

A service is bound when another component (e.g. client) calls **bindService()** method. The client can unbind the service by calling the **unbindService()** method. The service cannot be stopped until all clients unbind the service.

Example:

Let's see the example of service in android that plays an audio in the background. Audio will not be stopped even if you switch to another activity. To stop the audio, you need to stop the service.

1. Create an android project and design the **activity_main.xml as follows**

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

```
    tools:context="example.com.androidservice.MainActivity">
      <Button
        android:id="@+id/buttonStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="74dp"
        android:text="Start Service" />
      <Button
        android:id="@+id/buttonStop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="Stop Service" />
      <Button
        android:id="@+id/buttonNext"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="63dp"
        android:text="Next Page" />
</RelativeLayout>
```

2. Create the **activity_next.xml** file

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/androi
d"
    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.com.androidservice.NextPage">
      <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
```

```
    android:layout_marginTop="200dp"
    android:text="Next Page"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

3. Now create the service implementation class by inheriting the Service class and overriding its callback methods. **MyService.java**

```
import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;
import android.support.annotation.Nullable;
import android.widget.Toast;
  public class MyService extends Service {
    MediaPlayer myPlayer;
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {          return null;      }
    @Override
    public void onCreate() {
        Toast.makeText(this, "Service Created", Toast.LENGTH_LONG).show();
        myPlayer = MediaPlayer.create(this, R.raw.sun);
        myPlayer.setLooping(false); // Set looping
    }
    @Override
    public void onStart(Intent intent, int startid) {
        Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
        myPlayer.start();
    }
    @Override
    public void onDestroy() {
        Toast.makeText(this, "Service Stopped", Toast.LENGTH_LONG).show();
        myPlayer.stop();
    }  }
```

4. Now create the MainActivity class to perform event handling. Here, we are writing the code to start and stop service. Additionally, calling the second activity on buttonNext. **MainActivity.java**

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
import android.view.View;
import android.widget.Button;
  public class MainActivity extends AppCompatActivity implements View.OnClickListener
{
    Button buttonStart, buttonStop,buttonNext;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        buttonStart = findViewById(R.id.buttonStart);
        buttonStop = findViewById(R.id.buttonStop);
        buttonNext =  findViewById(R.id.buttonNext);
        buttonStart.setOnClickListener(this);
        buttonStop.setOnClickListener(this);
        buttonNext.setOnClickListener(this);
      }
    public void onClick(View src) {
        switch (src.getId()) {
            case R.id.buttonStart:
                startService(new Intent(this, MyService.class));
                break;
            case R.id.buttonStop:
                stopService(new Intent(this, MyService.class));
                break;
            case R.id.buttonNext:
                Intent intent=new Intent(this,NextPage.class);
                startActivity(intent);
                break;
        }
    }
}
```

5. Now, create another activity.

NextPage.java

```
 import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class NextPage extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_next);
    }
}
```

6. Finally, declare the service in the manifest file. Let's see the complete AndroidManifest.xml file

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="example.com.androidservice">
      <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                  <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".NextPage"></activity>
        <service
            android:name=".MyService"
            android:enabled="true" />
    </application>
  </manifest>
```

Chapter 4

4. Android Menu

In android, **Menu** is a part of user interface (UI) component which is used to handle some common functionality around the application. By using Menus in our applications, we can provide better and consistent user experience throughout the application. Menus are useful for displaying additional options that are not directly visible on the main UI of an application.

We can use Menu APIs to represent user actions and other options in our android application activities. In android, we can define a Menu in separate XML file and use that file in our activities or fragments based on our requirements. In android, we can define a Menu in separate XML file and use that file in our activities or fragments based on our requirements.

Define Android Menu in XML File

For all menu types, Android provides a standard XML format to define menu items. Instead of building a menu in our activity's code, we should define a menu and all its items in an XML menu resource and load menu resource as a Menu object in our activity or fragment.

In android, to define menu, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML file to build the menu with the following elements.

Element	Description
<menu>	It's a root element to define a Menu in XML file and it will hold one or more elements.
<item>	It is used to create a menu item and it represent a single item in menu. This element may contain a nested <menu> element in order to create a submenu.
<group>	It's an optional and invisible for <item> elements. It is used to categorize the menu items so they share properties such as active state and visibility.

Following is the example of defining a menu in XML file (**menu_example.xml**).

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/inbox"
      android:icon="@drawable/ic_inbox"
      android:title="@string/inbox" />
   <item android:id="@+id/compose"
      android:icon="@drawable/ic_compose"
      android:title="@string/compose"
      android:showAsAction="ifRoom" />
```

```
    <item android:id="@+id/outbox"
        android:icon="@drawable/ic_outbox"
        android:title="@string/outbox" />
</menu>
```

The **<item>** element in **menu** supports different type of attributes to define item's behaviour and appearance. Following are some of commonly used **<item>** attributes in android applications.

Attribute	Description
android:id	It is used to uniquely identify element in application.
android:icon	It is used to set the item's icon from drawable folder.
android:title	It is used to set the item's title
android:showAsAction	It is used to specify how the item should appear as an action item in the app bar.

In case if we want to add **submenu** in **menu** item, then we need to add a **<menu>** element as the child of an **<item>**. Following is the example of defining a submenu in menu item.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/file"
        android:title="@string/file" >
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/create_new"
                android:title="@string/create_new" />
            <item android:id="@+id/open"
                android:title="@string/open" />
        </menu>
    </item>
</menu>
```

Load Android Menu from an Activity

Once we are done with creation of menu, we need to load the menu resource from our activity using **MenuInflater.inflate()** like as shown below.

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
```

```
    inflater.inflate(R.menu.menu_example, menu);
}
```

If you observe above code we are calling our menu using MenuInflater.inflate() method in the form of **R.menu.menu_file_name**. Here our xml file name is **menu_example.xml** so we used file name **menu_example**.

Handle Android Menu Click Events

In android, we can handle a menu item click events using **ItemSelected()** event based on the menu type. Following is the example of handling a context menu item click event using **onContextItemSelected()**.

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.mail:
            // do something
            return true;
        case R.id.share:
            // do something
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

If you observe above code, the **getItemId()** method will get the id of selected menu item based on that we can perform our actions.

Types of Menus

In android, we have three fundamental type of Menus available to define a set of options and actions in our android applications.

- Options Menu
- Context Menu
- Popup Menu

4.1. Option Menu

In android, **Options Menu** is a primary collection of menu items for an activity and it is useful to implement actions that have a global impact on the app, such as Settings, Search, etc.

By using Options Menu, we can combine multiple actions and other options that are relevant to our current activity. We can define items for the options menu from either our Activity or Fragment class.

In case, if we define items for the options menu in both activity or fragment, then those items will be combined and display in UI.

Create Android Options Menu in XML File

In android, to define **options menu**, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML (**menu_example**) file to build the menu.

Following is the example of defining a menu in XML file (**menu_example.xml**).

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/inbox"
      android:icon="@drawable/ic_inbox"
      android:title="@string/inbox" />
   <item android:id="@+id/compose"
      android:icon="@drawable/ic_compose"
      android:title="@string/compose"
      android:showAsAction="ifRoom" />
   <item android:id="@+id/outbox"
      android:icon="@drawable/ic_outbox"
      android:title="@string/outbox" />
</menu>
```

Load Android Options Menu from an Activity

To specify the options menu for an activity, we need to override **onCreateOptionsMenu()** method and load the defined menu resource using **MenuInflater.inflate()** like as shown below.

```
@Override
public void onCreateOptionsMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
   super.onCreateContextMenu(menu, v, menuInfo);
   MenuInflater inflater = getMenuInflater();
   inflater.inflate(R.menu.menu_example, menu);
}
```

If you observe above code we are calling our menu using **MenuInflater.inflate()** method in the form of **R.menu.menu_file_name**. Here our xml file name is **menu_example.xml** so we used file name **menu_example**.

Handle Android Options Menu Click Events

In android, we can handle a options menu item click events using **onOptionsItemSelected()** event method.

Following is the example of handling a options menu item click event using **onOptionsItemSelected()**.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
      case R.id.inbox:
```

```
            // do something
            return true;
        case R.id.compose:
            // do something
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

Android Options Menu Attributes

Following are some of commonly used attributes related to options menu control in android applications.

Attribute	Description
android:id	It is used to uniquely identify element in application.
android:icon	It is used to set the item's icon from drawable folder.
android:title	It is used to set the item's title
android:showAsAction	It is used to specify how the item should appear as an action item in the app bar.

Note: If you are using Android 3.0 +, the Options Menu won't support any item shortcuts and item icons in the menu.

Android Options Menu Example

Following is the example of implementing an **Options Menu** in android application.

Create a new android application using android studio and give names as **OptionsMenu**. In android, to define **options menu**, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML (**options_menu.xml**) file to build the menu. Now open newly created xml (**options_menu.xml**) file and write the code like as shown below.

options_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/inbox_item"
        android:title="Inbox" />
    <item android:id="@+id/ compose_item"
        android:title="Compose" />
    <item android:id="@+id/outbox_item"
        android:title="Outbox" />
    <item android:id="@+id/spam_item"
        android:title="Spam" />
```

```
    <item android:id="@+id/logout_item"
        android:title="Logout" />
</menu>
```

Once we are done with creation of menu, we need to load this menu XML resource from our activity using **onCreateOptionsMenu()** callback method, for that open main activity file **MainActivity.java** from **\java\com.example.optionsmenu** path and write the code like as shown below.

MainActivity.java

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;


public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.options_menu, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        Toast.makeText(this, "Selected Item: " +item.getTitle(), Toast.LENGTH_SHORT).show();
        switch (item.getItemId()) {
            case R.id.inbox_item:
              // do your code
               return true;
            case R.id.compose_item:
               // do your code
```

```
            return true;
        case R.id.outbox_item:
            // do your code
            return true;
        case R.id.spam_item:
            // do your code
            return true;
        case R.id.logout_item:
            // do your code
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
  }
}
```

If you observe above code we are overriding **onCreateOptionsMenu()** method in activity to create options menu and loaded defined menu resource using **MenuInflater.inflate()**.

Generally, during the launch of our activity, **onCreate()** callback method will be called by android framework to get the required layout for an activity.

4.2. Context Menu

In android, **Context Menu** is a floating menu that appears when the user performs a long click on an element and it is useful to implement an action that effect the selected content or context frame.

The android Context Menu is more like the menu which displayed on right click in Windows or Linux.

In android, the Context Menu offers an actions that effect a specific item or context frame in the UI and we can provide a context menu for any view. The context menu won't support any item shortcuts and item icons.

Create Android Context Menu in Activity

The views which we used to show the context menu on long press, we need to register that views using **registerForContextMenu(View)** in our activity and we need to override **onCreateContextMenu()** in our activity or fragment. When the registered view receives a long click event, the system calls our **onCreateContextMenu()** method. By using **onCreateContextMenu()** method, we can create our menu items like as shown below.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.activity_main);
    Button btn = (Button) findViewById(R.id.btnShow);
    registerForContextMenu(btn);
}
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo
menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    menu.setHeaderTitle("Context Menu");
    menu.add(0, v.getId(), 0, "Inbox");
    menu.add(0, v.getId(), 0, "Outbox");
}
```

If you observe above code, we registered our Button control using **registerForContextMenu()** to show the context menu on button long-click and binding the Context Menu items using **onCreateContextMenu()** method.

Handle Android Context Menu Click Events

In android, we can handle a context menu item click events using **onContextItemSelected()** method.

Following is the example of handling a context menu item click event using **onContextItemSelected()** method.

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    if (item.getTitle() == "Inbox") {
        // do your coding
    }
    else {
        return  false;
    }
    return true;
}
```

Note: If you are using Android 3.0 +, the Context Menu won't support any item shortcuts and item icons in the menu.

Android Context Menu Example

Following is the example of implementing a **Context Menu** in android application.

Create a new android application using android studio and give names as **ContextMenuExample**. Now open an **activity_main.xml** file from **\res\layout** path and write the code like as shown below

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
```

```
    <Button
        android:id="@+id/btnShow"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Long press me"
        android:layout_marginTop="200dp" android:layout_marginLeft="100dp"/>
</LinearLayout>
```

If you observe above code we created a one Button control in XML Layout file to show the context menu when we do long press on Button. Once we are done with creation of layout with required control, we need to load the XML layout resource from our activity **onCreate()** callback method, for that open main activity file **MainActivity.java** from **\java\com.example.contextmenuexample** path and write the code like as shown below.

MainActivity.java

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btn = (Button) findViewById(R.id.btnShow);
        registerForContextMenu(btn);
    }
    @Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo
menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);
        menu.setHeaderTitle("Context Menu");
        menu.add(0, v.getId(), 0, "Inbox");
        menu.add(0, v.getId(), 0, "Compose");
        menu.add(0, v.getId(), 0, "Outbox");
        menu.add(0, v.getId(), 0, "Logout");
    }
    @Override
    public boolean onContextItemSelected(MenuItem item) {
        Toast.makeText(this, "Selected Item: " +item.getTitle(), Toast.LENGTH_SHORT).show();
        return true;
    }
}
```

If you observe above code we are overriding **onCreateContextMenu()** method in activity to create context menu and registered view for context menu using **registerForContextMenu()**.

Generally, during the launch of our activity, **onCreate()** callback method will be called by android framework to get the required layout for an activity.

4.3. Popup Menu

In android, **Popup Menu** displays a list of items in a vertical list that's anchored to the view that invoked the menu and it's useful for providing an overflow of actions that related to specific content.

In android, the Popup Menu provides an overflow of actions that are related to specific content and the actions in popup menu won't affect the corresponding content. The popup menu won't support any item shortcuts and item icons.

In android, Popup menu is available with API level 11 (Android 3.0) and higher versions. If you are using Android 3.0 +, the Popup Menu won't support any item shortcuts and item icons in the menu.

Create Android Popup Menu in XML File

In android, to define **popup menu**, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML (menu_example) file to build the menu.

Following is the example of defining a menu in XML file (**menu_example.xml**).

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/inbox"
      android:icon="@drawable/ic_inbox"
      android:title="@string/inbox" />
   <item android:id="@+id/compose"
      android:icon="@drawable/ic_xompose"
      android:title="@string/compose"
      android:showAsAction="ifRoom" />
   <item android:id="@+id/outbox"
      android:icon="@drawable/ic_outbox"
      android:title="@string/outbox" />
</menu>
```

Once we are done with the creation of menu, we need to create a view element which anchored the menu.

```
<Button
   android:id="@+id/btnShow"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Show Popup Menu"
   android:onClick="showPopup" />
```

Now in our activity we need to implement showPopup method to show the popup menu.

Load Android Popup Menu from an Activity

To show the popup menu for the view, we need to instantiate **Popup** constructor and use **MenuInflater** to load the defined menu resource using **MenuInflater.inflate()** like as shown below.

```
public void showPopup(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    MenuInflater inflater = popup.getMenuInflater();
    inflater.inflate(R.menu.menu_example, popup.getMenu());
    popup.show();
}
```

Handle Android Popup Menu Click Events

To perform an action when the user selects a menu item, we need to implement the **PopupMenu.OnMenuItemClickListener** interface and register it with our **PopupMenu** by calling **setOnMenuItemclickListener()**. When the user selects an item, the system calls the **onMenuItemClick()** callback in your interface.

Following is the example of handling a popup menu item click event using **onMenuItemClick()**.

```
public void showMenu(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    popup.setOnMenuItemClickListener(this);
    popup.inflate(R.menu.actions);
    popup.show();
}

@Override
public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.inbox:
            inbox(item);
            return true;
        case R.id.compose:
            compose(item);
            return true;
        default:
            return false;
    }
}
```

Note: If you are using Android 3.0 +, the Popup Menu won't support any item shortcuts and item icons in the menu.

Android Popup Menu Example

Following is the example of implementing a **Popup Menu** in android application.

Create a new android application using android studio and give names as **PopupMenuExample**.

Now open an **activity_main.xml** file from **\res\layout** path and write the code like as shown below

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >

   <Button
      android:id="@+id/btnShow"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Show Popup Menu"
      android:layout_marginTop="200dp" android:layout_marginLeft="100dp"/>
</LinearLayout>
```

If you observe above code we created a one Button control in XML Layout file to show the popup menu when we click on Button. In android, to define **popup menu**, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML (**popup_menu.xml**) file to build the menu. Now open newly created xml (**popup_menu.xml**) file and write the code like as shown below.

popup_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
   <item android:id="@+id/inbox_item"
      android:title="Inbox" />
   <item android:id="@+id/compose_item"
      android:title="Compose" />
   <item android:id="@+id/outbox_item"
      android:title="Outbox" />
   <item android:id="@+id/spam_item"
      android:title="Spam" />
   <item android:id="@+id/logout_item"
      android:title="Logout" />
  </menu>
```

Once we are done with creation of **menu**, we need to load this menu XML resource from our activity by instantiating a **Popup** constructor, for that open

main activity file **MainActivity.java** from **\java\com.example.popupmenuexample**path and write the code like as shown below.

MainActivity.java

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.PopupMenu;
import android.widget.Toast;
```

```
public
class MainActivity extends AppCompatActivity implements PopupMenu.OnMenuItemClickListener {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btn = (Button) findViewById(R.id.btnShow);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                PopupMenu popup = new PopupMenu(MainActivity.this, v);
                popup.setOnMenuItemClickListener(MainActivity.this);
                popup.inflate(R.menu.popup_menu);
                popup.show();
            }
        });
    }
    @Override
    public boolean onMenuItemClick(MenuItem item) {
        Toast.makeText(this, "Selected Item: " +item.getTitle(), Toast.LENGTH_SHORT).show();
        switch (item.getItemId()) {
            case R.id.inbox_item:
                // do your code
                return true;
            case R.id.compose_item:
                // do your code
                return true;
            case R.id.outbox_item:
                // do your code
                return true;
            case R.id.spam_item:
                // do your code
                return true;
            case R.id.logout_item:
                // do your code
                return true;
            default:
                return false;
        }
    }
}
```

If you observe above code we are trying to show popup menu on Button click, loaded defined menu resource using **Popup.inflate()** and implement popup menu items click event.

Generally, during the launch of our activity, **onCreate()** callback method will be called by android framework to get the required layout for an activity.

Chapter 5

5. Android Data Storage

5.1. Shared Preferences

In android, **Shared Preferences** are used to save and retrieve the primitive data types (integer, float, Boolean, string, long) data in the form of key-value pair from a file within an apps file structure.The **Shared Preferences** object will point to a file that contains a key-value pairs and provides a simple read and write methods to save and retrieve the key-value pairs from a file. The Shared Preferences file is managed by an android framework and it can be accessed anywhere within the app to read or write data into the file, but it's not possible to access the file from any other app so it's secured. The Shared Preferences are useful to store the small collection of key-values such as user's login information, app preferences related to users, etc. to maintain the state of app, next time when they login again to the app.

Handle a Shared Preferences

In android, we can save the preferences data either in single or multiple files based on our requirements.

In case if we use single file to save the preferences, then we need to use **getPreferences()** method to get the values from Shared Preferences file and for multiple files we need to call a **getSharedPreferences()** method and pass a file name as a parameter.

Method	Description
getPreferences()	This method is for activity level preferences and each activity will have it's own preference file and by default this method retrieves a default shared preference file that belongs to the activity.
getSharedPreferences()	This method is useful to get the values from multiple shared preference files by passing the name as parameter to identify the file. We can call this from any Context in our app.

Following are the different ways to initialize the Shared Preferences in our application.

If we are using single shared preference file for our activity, then we need to initialize the **SharedPreferences** object by using **getPreferences()** method like as shown below.

```
SharedPreferences sharedPref = getPreferences(Context.MODE_PRIVATE);
```

In case, if we are using multiple shared preference files, then we need to initialize the **SharedPreferences** object by using **getSharedPreferences()** method like as shown below.

```
SharedPreferences sharedPref = getSharedPreferences("filename1",Context.MODE_PRIVATE);
```

Here, the name "**filename1**" is the preference file, which we want to read the values based on our requirements and the context mode **MODE_PRIVATE**, will make sure that the file can be accessed only within our application.

Write to Shared Preferences

To store a data in shared preference file, we need an **editor** to edit and save the changes in **SharedPreferences** object. Following is the code snippet to store the data in shared preference file using **editor**.

```
SharedPreferences sharedPref = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putBoolean("keyname",true);editor.putString("keyname","string value");
editor.putInt("keyname","int value");editor.putFloat("keyname","float value");
editor.putLong("keyname","long value");editor.commit();
```

If you observe above code snippet, we created a **SharedPreferences.Editor** by calling the **edit()** method of **SharedPreferences** object. We added a primitive data type values such as integer, float, long, string and Boolean by passing the keys and values to the methods **putInt()**, **putString()**, etc. based on our requirements. After that, to save all the changes we are calling **commit()** method.

Read from Shared Preferences

To read or retrieve a values from Shared Preferences file, we need to call a methods such as **getInt()**, **getString()**, etc. by providing the key for the value which we want to get like as shown below.

```
SharedPreferences pref = getPreferences(Context.MODE_PRIVATE);
pref.getString("keyname",null);pref.getInt("keyname",0);pref.getFloat("keyname",0);
pref.getBoolean("keyname",true); pref.getLong("keyname",0);
```

If you observe above code snippet, we are getting the values from shared preferences using a methods such as **getInt()**, **getFloat()**, etc. by providing the key for the value which we want to get.

Deleting from Shared Preferences

To delete a value from Shared Preferences file, we need to call a **remove()** method by providing the key for the value which we want to delete like as shown below.

```
SharedPreferences pref = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = pref.edit();
editor.remove("keyname");editor.commit();
```

If you observe above code snippet, we are deleting the values from shared preferences using a method called **remove()** by providing the key for the value which we want to delete and committing the changes to shared preferences file using **commit()** method.

Clearing from Shared Preferences

We can clear all the data from Shared Preferences file using **clear()** method like as shown below.

```
SharedPreferences pref = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = pref.edit();
editor.clear();editor.commit();
```

If you observe above code snippet, we are clearing all the values from shared preferences using a method called **clear()** and committing the changes to shared preferences file using **commit()** method.

Now we will see how to store and retrieve primitive data type key-value pairs in shared preferences file using **SharedPreferences** object in android application with examples.

Android Shared Preferences Example

Following is the example of storing and retrieving the primitive data type values from shared preferences file using **SharedPreferences**.

Create a new android application using android studio and give names as **SharedPreferencesExample**. In case if you are not aware of creating an app in android studio check this article Android Hello World App.

Once we create an application, open **activity_main.xml** file from **\res\layout** folder path and write the code like as shown below.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
      android:id="@+id/fstTxt"        android:layout_width="wrap_content"
      android:layout_height="wrap_content"        android:layout_marginLeft="100dp"
      android:layout_marginTop="150dp"        android:text="UserName" />
   <EditText
      android:id="@+id/txtName"        android:layout_width="wrap_content"
      android:layout_height="wrap_content"        android:layout_marginLeft="100dp"      android:ems="10"/>
   <TextView
      android:id="@+id/secTxt"        android:layout_width="wrap_content"
      android:layout_height="wrap_content"  android:text="Password"  android:layout_marginLeft="100dp" />
   <EditText
      android:id="@+id/txtPwd"        android:inputType="textPassword"
      android:layout_width="wrap_content"        android:layout_height="wrap_content"
      android:layout_marginLeft="100dp"        android:ems="10" />
   <Button
      android:id="@+id/btnLogin"        android:layout_width="wrap_content"
      android:layout_height="wrap_content"    android:layout_marginLeft="100dp"      android:text="Login" />
</LinearLayout>
```

Now we will create another layout resource file **details.xml** in **\res\layout** path to get the first activity (**activity_main.xml**) details in second activity file for that right click on your **layout** folder Go to **New** select **Layout Resource File** and give name as **details.xml**.

Once we create a new layout resource file **details.xml**, open it and write the code like as shown below

details.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
      android:layout_width="wrap_content"      android:layout_height="wrap_content"
      android:id="@+id/resultView"       android:layout_gravity="center"
      android:layout_marginTop="170dp"       android:textSize="20dp"/>
   <Button
      android:id="@+id/btnLogOut"       android:layout_width="wrap_content"
      android:layout_height="wrap_content"       android:layout_gravity="center"
      android:layout_marginTop="20dp"       android:text="Log Out" />
</LinearLayout>
```

Now open your main activity file **MainActivity.java** from **\java\com.example.sharedpreferencesexample** path and write the code like as shown below

MainActivity.java

```
import android.content.*;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.*;
public class MainActivity extends AppCompatActivity {
   EditText uname, pwd;
   Button loginBtn;
   SharedPreferences pref;
   Intent intent;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      uname = (EditText)findViewById(R.id.txtName);
      pwd = (EditText)findViewById(R.id.txtPwd);
      loginBtn = (Button)findViewById(R.id.btnLogin);
      pref = getSharedPreferences("user_details",MODE_PRIVATE);
      intent = new Intent(MainActivity.this,DetailsActivity.class);
      if(pref.contains("username") && pref.contains("password")){
         startActivity(intent);
      }
      loginBtn.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            String username = uname.getText().toString();
            String password = pwd.getText().toString();
            if(username.equals("abebe") && password.equals("abc123")){
               SharedPreferences.Editor editor = pref.edit();
               editor.putString("username",username);
               editor.putString("password",password);
               editor.commit();
```

```
                Toast.makeText(getApplicationContext(), "Login Successful",Toast.LENGTH_SHORT).show();
                startActivity(intent);
            }
            else    {
                Toast.makeText(getApplicationContext(),"Credentials are not valid",Toast.LENGTH_SHORT).show();
            }   }    });
    }
}
```

If you observe above code, we are checking whether the entered username and password details matching or not based on that we are saving the details in shared preferences file and redirecting the user to another activity.

Now we will create another activity file **DetailsActivity.java** in **\java\com.examplw.sharedpreferencesexample** path to show the details from shared preference file for that right click on your application folder Go to **New** select **JavaClass** and give name as **DetailsActivity.java**.

Once we create a new activity file **DetailsActivity.java**, open it and write the code like as shown below

DetailsActivity.java

```
import android.content.*;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.*;
public class DetailsActivity extends AppCompatActivity {
    SharedPreferences prf;
    Intent intent;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.details);
        TextView result = (TextView)findViewById(R.id.resultView);
        Button btnLogOut = (Button)findViewById(R.id.btnLogOut);
        prf = getSharedPreferences("user_details",MODE_PRIVATE);
        intent = new Intent(DetailsActivity.this,MainActivity.class);
        result.setText("Hello, "+prf.getString("username",null));
        btnLogOut.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                SharedPreferences.Editor editor = prf.edit();
                editor.clear();
                editor.commit();
                startActivity(intent);
            }
        });
    }
}
```

Now we need to add this newly created activity in **AndroidManifest.xml** file in like as shown below.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.sharedpreferencesexample">
  <application
    android:allowBackup="true"  android:icon="@mipmap/ic_launcher" android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"  android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".DetailsActivity" android:label="Shared Preferences - Details"></activity>
  </application>
</manifest>
```

If you observe above example, we are checking whether the entered user details matching or not based on that we are saving the user details in shared preferences file and redirecting the user to another activity file (**DetailsActivity.java**) to show the users details and added all the activities in **AndroidManifest.xml** file.

Effective session management is helps ensure that only authenticated users can access the application features and that their sessions remain secure throughout their usage.

Android Session Management

In android, **Session Management** is a process which is used to maintain the required values in a session to use it in application. In android we can manage the logged in user details in session either by storing it in global variables or in Shared Preferences. In case, if we store the values in global variables, those will be lost whenever the user closes the application but if we store the values in Shared Preferences, those will be persisted even if the application is closed by a user.

In android, Shared Preferences are used to save and retrieve the primitive data types (integer, float, Boolean, string, long) in the form of key-value pair from a file within an apps file structure.

To know more about Shared Preferences, check this Android Shared Preferences with Examples.

Now we will see how to store and retrieve logged in user details from shared preferences file using SharedPreferences object and clear or delete the stored session values from Shared Preferences file whenever the user clicks on logout button in android application with examples.

Android Session Management Example

Following is the example of storing and retrieving the logged in user details from shared preferences file using SharedPreferences and clear the stored session values on logout button click.

Create a new android application using android studio and give names as **SharedPreferencesExample**. Once we create an application, open **activity_main.xml** file from **\res\layout** folder path and write the code like as shown below.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
      android:id="@+id/fstTxt" android:layout_width="wrap_content"   android:layout_height="wrap_content"
      android:layout_marginLeft="100dp" android:layout_marginTop="150dp"     android:text="UserName" />
   <EditText
      android:id="@+id/txtName"    android:layout_width="wrap_content"
      android:layout_height="wrap_content"  android:layout_marginLeft="100dp"      android:ems="10"/>
   <TextView
      android:id="@+id/secTxt"   android:layout_width="wrap_content"
      android:layout_height="wrap_content"  android:text="Password"  android:layout_marginLeft="100dp" />
   <EditText
      android:id="@+id/txtPwd"  android:inputType="textPassword"  android:layout_width="wrap_content"
android:layout_height="wrap_content"     android:layout_marginLeft="100dp"  android:ems="10" />
   <Button
      android:id="@+id/btnLogin" android:layout_width="wrap_content"ndroid:layout_height="wrap_content"
      android:layout_marginLeft="100dp"      android:text="Login" />
</LinearLayout>
```

Now we will create another layout resource file **details.xml** in **\res\layout** path to get the first activity (**activity_main.xml**) details in second activity file for that right click on your **layout** folder Go to **New** select **Layout Resource File** and give name as **details.xml**.

Once we create a new layout resource file **details.xml**, open it and write the code like as shown below

details.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
      android:layout_width="wrap_content"  android:layout_height="wrap_content"android:textSize="20dp"
      android:id="@+id/resultView"  android:layout_gravity="center"  android:layout_marginTop="170dp"/>
   <Button
      android:id="@+id/btnLogOut"       android:layout_width="wrap_content"
      android:layout_height="wrap_content"        android:layout_gravity="center"
      android:layout_marginTop="20dp"       android:text="Log Out" />
</LinearLayout>
```

Now open your main activity file **MainActivity.java** from **\java\com.example.sharedpreferencesexample** path and write the code like as shown below

MainActivity.java

```
import android.content.*;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
import android.view.View;
import android.widget.*;
public class MainActivity extends AppCompatActivity {
    EditText uname, pwd;
    Button loginBtn;
    SharedPreferences pref;
    Intent intent;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        uname = (EditText)findViewById(R.id.txtName);
        pwd = (EditText)findViewById(R.id.txtPwd);
        loginBtn = (Button)findViewById(R.id.btnLogin);
        pref = getSharedPreferences("user_details",MODE_PRIVATE);
        intent = new Intent(MainActivity.this,DetailsActivity.class);
        if(pref.contains("username") && pref.contains("password")){
            startActivity(intent);
        }
        loginBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String username = uname.getText().toString();
                String password = pwd.getText().toString();
                if(username.equals("abebe") && password.equals("abc123")){
                    SharedPreferences.Editor editor = pref.edit();
                    editor.putString("username",username);
                    editor.putString("password",password);
                    editor.commit();
                    Toast.makeText(getApplicationContext(), "Login Successful",Toast.LENGTH_SHORT).show();
                    startActivity(intent);
                }
                else    {
                    Toast.makeText(getApplicationContext(),"Credentials are not valid",Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}
```

If you observe above code, we are checking whether the entered username and password details matching or not based on that we are saving the details in shared preferences file and redirecting the user to another activity.Now we will create another activity file **DetailsActivity.java** in **\java\com.example.sharedpreferencesexample** path to show the details from shared preference file for that right click on your application folder → Go to **New** → select **JavaClass** and give name as **DetailsActivity.java**.

Once we create a new activity file **DetailsActivity.java**, open it and write the code like as shown below

DetailsActivity.java

```
import android.content.*;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.*;
public class DetailsActivity extends AppCompatActivity {
    SharedPreferences prf;
    Intent intent;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.details);
        TextView result = (TextView)findViewById(R.id.resultView);
        Button btnLogOut = (Button)findViewById(R.id.btnLogOut);
        prf = getSharedPreferences("user_details",MODE_PRIVATE);
        intent = new Intent(DetailsActivity.this,MainActivity.class);
        result.setText("Hello, "+prf.getString("username",null));
        btnLogOut.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                SharedPreferences.Editor editor = prf.edit();
                editor.clear();
                editor.commit();
                startActivity(intent);
            }
        });
    }
}
```

Now we need to add this newly created activity in **AndroidManifest.xml** file in like as shown below.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sharedpreferencesexample">
    <application
        android:allowBackup="true"   android:icon="@mipmap/ic_launcher" android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"   android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".DetailsActivity" android:label="Shared Preferences - Details"></activity>
    </application>
</manifest>
```

If you observe above example, we are checking whether the entered user details matching or not based on that we are saving the user details in shared preferences file and redirecting the user to another activity file (**DetailsActivity.java**) to show the users details and added all the activities in **AndroidManifest.xml** file.

5.2. Internal Storage

we use object of
FileOutputStream =for storing or writing to internal storage
FileInputStream =read the data file from device

In android, **Internal Storage** is useful to store the data files locally on the device's internal memory using **FileOutputStream** object. After storing the data files in device internal storage, we can read the data file from device using **FileInputStream** object.

The data files saved in internal storage is managed by an android framework and it can be accessed anywhere within the app to read or write data into the file, but it's not possible to access the file from any other app so it's secured. When the user uninstall the app, automatically these data files will be removed from the device internal storage.

Write a File to Internal Storage

By using android **FileOutputStream** object **openFileOutput** method, we can easily create and write a data to a file in device's internal storage.

Following is the code snippet to create and write a private file to the device internal memory.

```
String FILENAME = "user_details";
String name = "abebe";
FileOutputStream fstream = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fstream.write(name.getBytes());
fstream.close();
```

If you observe above code, we are creating and writing a file in device internal storage by using **FileOutputStream**object **openFileOutput** method with **file name** and **MODE_PRIVATE** mode to make the file private to our application. We used **write()** method to write the data in file and used **close()** method to close the stream.In android, we have a different mode such as **MODE_APPEND**, **MODE_WORLD_READBLE**, **MODE_WORLD_WRITEABLE**, etc. to use it in our application based on your requirements.

Apart from above methods **write()** and **close()**, **FileOutputStream** object is having other methods, those are

Method	Description
getChannel()	It returns the unique FileChannel object associated with this file output stream.
getFD()	It returns the file descriptor which is associated with the stream.
write(byte[] b, int off, int len)	It writes **len** bytes from the specified byte array starting at offset **off** to the file output stream.

Read a File from Internal Storage

By using android **FileInputStream** object **openFileInput** method, we can easily read the file from device's internal storage.

Following is the code snippet to read a data from file which is in device's internal memory.

```
String FILENAME = "user_details";
FileInputStream fstream = openFileInput(FILENAME);
StringBuffer sbuffer = new StringBuffer();
int i;
while ((i = fstream.read())!= -1){
   sbuffer.append((char)i);
}
fstream.close();
```

If you observe above code, we are reading a file from device internal storage by using FileInputStream object openFileInput method with file name. We used read() method to read one character at a time from the file and used close() method to close the stream.

Apart from above methods **read()** and **close()**, **FileInputStream** object is having other methods, those are

Method	Description
getChannel()	It returns the unique FileChannel object associated with this file output stream.
getFD()	It returns the file descriptor which is associated with the stream.
read(byte[] b, int off, int len)	It reads **len** bytes of data from the specified file input stream into an array of bytes.

Now we will see how to save files directly on the device's internal memory and read the data files from device internal memory by using **FileOutputStream** and **FileInputStream** objects in android application with examples.

Android Internal Storage Example

Following is the example of storing and retrieving the data files from device's internal memory by using **FileOutputStream** and **FileInputStream** objects.

Create a new android application using android studio and give names as **InternalStorageExample**. In case if you are not aware of creating an app in android studio check this article Android Hello World App.

Once we create an application, open **activity_main.xml** file from **\res\layout** folder path and write the code like as shown below.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical" android:layout_width="match_parent"
  android:layout_height="match_parent">
  <TextView
    android:id="@+id/fstTxt"  android:layout_width="wrap_content"  android:layout_height="wrap_content"
    android:layout_marginLeft="100dp"   android:layout_marginTop="150dp"   android:text="UserName" />
  <EditText
    android:id="@+id/txtName"  android:layout_width="wrap_content"

     android:layout_height="wrap_content"   android:layout_marginLeft="100dp"   android:ems="10"/>
  <TextView
    android:id="@+id/secTxt"  android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:text="Password"   android:layout_marginLeft="100dp" />
  <EditText
    android:id="@+id/txtPwd"   android:inputType="textPassword"    android:layout_width="wrap_content"
    android:layout_height="wrap_content"  android:layout_marginLeft="100dp"   android:ems="10" />
  <Button
    android:id="@+id/btnSave"      android:layout_width="wrap_content"
    android:layout_height="wrap_content"    android:layout_marginLeft="100dp"       android:text="Save" />
</LinearLayout>
```

Now we will create another layout resource file **details.xml** in **\res\layout** path to get the first activity (**activity_main.xml**) details in second activity file for that right click on your **layout** folder → Go to **New** → select **Layout Resource File** and give name as **details.xml**.

Once we create a new layout resource file **details.xml**, open it and write the code like as shown below

details.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical" android:layout_width="match_parent"
  android:layout_height="match_parent">
  <TextView
    android:layout_width="wrap_content"   android:layout_height="wrap_content"android:textSize="20dp"
    android:id="@+id/resultView"  android:layout_gravity="center"  android:layout_marginTop="170dp"/>
  <Button
    android:id="@+id/btnBack"        android:layout_width="wrap_content"
    android:layout_height="wrap_content"  android:layout_gravity="center"
```

```
android:layout_marginTop="20dp"        android:text="Back" />
</LinearLayout>
```

Now open your main activity file **MainActivity.java** from **\java\com.example.internalstorageexample** path and write the code like as shown below

MainActivity.java

```
import android.content.*;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.*;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
public class MainActivity extends AppCompatActivity {
    EditText uname, pwd;
    Button saveBtn;
    FileOutputStream fstream;
    Intent intent;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        uname = (EditText)findViewById(R.id.txtName);
        pwd = (EditText)findViewById(R.id.txtPwd);
        saveBtn = (Button)findViewById(R.id.btnSave);
        saveBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String username = uname.getText().toString()+"\n";
                String password = pwd.getText().toString();
                try {
                    fstream = openFileOutput("user_details", Context.MODE_PRIVATE);
                    fstream.write(username.getBytes());
                    fstream.write(password.getBytes());
                    fstream.close();
```

```
            Toast.makeText(getApplicationContext(), "Details Saved Successfully",Toast.LENGTH_SHORT).show();
                intent = new Intent(MainActivity.this,DetailsActivity.class);
                startActivity(intent);
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    });
  }
}
```

If you observe above code, we are taking entered username and password details and saving it in device local file and redirecting the user to another activity.

Now we will create another activity file **DetailsActivity.java** in **\java\com.example.internalstorageexample** path to show the details from device internal storage for that right click on your application folder → Go to **New** → select **Java Class** and give name as **DetailsActivity.java**. Once we create a new activity file **DetailsActivity.java**, open it and write the code like as shown below

DetailsActivity.java

```
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
public class DetailsActivity extends AppCompatActivity {
  FileInputStream fstream;
  Intent intent;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.details);
        TextView result = (TextView)findViewById(R.id.resultView);
        Button back = (Button)findViewById(R.id.btnBack);
        try {
            fstream = openFileInput("user_details");
            StringBuffer sbuffer = new StringBuffer();
            int i;
            while ((i = fstream.read())!= -1){
                sbuffer.append((char)i);
            }
            fstream.close();
            String details[] = sbuffer.toString().split("\n");
            result.setText("Name: "+ details[0]+"\nPassword: "+details[1]);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        back.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                intent = new Intent(DetailsActivity.this,MainActivity.class);
                startActivity(intent);
            }       });    }
}
```

Now we need to add this newly created activity in **AndroidManifest.xml** file in like as shown below.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.internalstorageexample">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
```

```
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".DetailsActivity" android:label="Internal Storage - Details"></activity>
    </application>
</manifest>
```

If you observe above example, we are saving entered details file and redirecting the user to another activity file (**DetailsActivity.java**) to show the users details and added all the activities in **AndroidManifest.xml** file.

5.3. External Storage

In android, **External Storage** is useful to store the data files publicly on the shared external storage using **FileOutputStream** object. After storing the data files on external storage, we can read the data file from external storage media using **FileInputStream** object. The data files saved in external storage are world-readable and can be modified by the user when they enable USB mass storage to transfer files on a computer.

Grant Access to External Storage

To read or write files on the external storage, our app must acquire the **WRITE_EXTERNAL_STORAGE** and **READ_EXTERNAL_STORAGE** system permissions. For that, we need to add following permissions in android manifest file like as shown below.

```
<manifest>
    ....
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    ....
</manifest>
```

Checking External Storage Availability

Before we do any work with external storage, first we need to check whether the media is available or not by calling **getExternalStorageState()**. The media might be mounted to a computer, missing, read-only or in some other state. To get the media status, we need to write the code like as shown below.

```
boolean Available= false;
boolean Readable= false;
String state = Environment.getExternalStorageState();
if(Environment.MEDIA_MOUNTED.equals(state)){
   // Both Read and write operations available
   Available= true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)){
   // Only Read operation available
   Available= true;
   Readable= true;
} else {
   // SD card not mounted
   Available = false;
}
```

If you observe above code snippet, we used **getExternalStorageState()** method to know the status of media mounted to a computer, such as missing, read-only or in some other state.

In android, by using **getExternalStoragePublicDirectory()** method we can access the files from appropriate public directory by passing the type of directory we want, such as DIRECTORY_MUSIC, DIRECTORY_PICTURES, DIRECTORY_RINGTONES, etc. based on our requirements.

If we save our files to corresponding media-type public directory, the system's media scanner can properly categorize our files in the system.

Following is the example to create a directory for new photo album in the public pictures directory.

```
// Get the directory for the user's public pictures directory.
File file = new File(Environment.getExternalStoragePublicDirectory(
      Environment.DIRECTORY_PICTURES), albumName);
if (!file.mkdirs()) {
   Log.e(LOG_TAG, "Directory not created");
}
```

In case, if we are handling the files that are not intended for other apps to use, then we should use a private storage directory on the external storage by calling **getExternalFilesDir()**.

Write a File to External Storage

By using android **FileOutputStream** object and **getExternalStoragePublicDirectory** method, we can easily create and write a data to the file in external storage public folders.

Following is the code snippet to create and write a public file in device **Downloads** folder.

```
String FILENAME = "user_details";
String name = "abebe";
```

```
File folder = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);
File myFile = new File(folder, FILENAME);
FileOutputStream fstream = new FileOutputStream(myFile);

fstream.write(name.getBytes());
fstream.close();
```

If you observe above code, we are creating and writing a file in device public **Downloads** folder by using **getExternalStoragePublicDirectory** method. We used **write()** method to write the data in file and used **close()**method to close the stream.

Read a File from External Storage

By using android **FileInputStream** object and **getExternalStoragePublicDirectory** method, we can easily read the file from external storage.

Following is the code snippet to read a data from file which is in **downloads** folder.

```
String FILENAME = "user_details";
File folder = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);
File myFile = new File(folder, FILENAME);
FileInputStream fstream = new FileInputStream(myFile);
StringBuffer sbuffer = new StringBuffer();
int i;
while ((i = fstream.read())!= -1){
   sbuffer.append((char)i);
}
fstream.close();
```

If you observe above code, we are reading a file from device Downloads folder storage by using **FileInputStream**object **getExternalStoragePublicDirectory** method with **file name**. We used **read()** method to read one character at a time from the file and used **close()** method to close the stream.

Now we will see how to save or write file to external memory and read the file data from external storage using android **FileInputStream** and **FileOutputStream** objects in android applications with examples.

Android External Storage Example

Following is the example of storing and retrieving the data files from external memory by using **FileOutputStream** and **FileInputStream** objects.

Create a new android application using android studio and give names as **ExternalStorageExample**. In case if you are not aware of creating an app in android studio check this article Android Hello World App.

Once we create an application, open **activity_main.xml** file from **\res\layout** folder path and write the code like as shown below.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical" android:layout_width="match_parent"
  android:layout_height="match_parent">
  <TextView
    android:id="@+id/fstTxt" android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:layout_marginLeft="100dp"  android:layout_marginTop="150dp"  android:text="UserName" />
  <EditText
  android:id="@+id/txtName"  android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:layout_marginLeft="100dp"      android:ems="10"/>
  <TextView
    android:id="@+id/secTxt"  android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:text="Password"     android:layout_marginLeft="100dp" />
  <EditText
    android:id="@+id/txtPwd" android:inputType="textPassword"     android:layout_width="wrap_content"
    android:layout_height="wrap_content"  android:layout_marginLeft="100dp"      android:ems="10" />
  <Button
   android:id="@+id/btnSave" android:layout_width="wrap_content"android:layout_height="wrap_content"
    android:layout_marginLeft="100dp"       android:text="Save" />
</LinearLayout>
```

Now we will create another layout resource file **details.xml** in **\res\layout** path to get the first activity(**activity_main.xml**) details in second activity file for that right click on your **layout** folder → Go to **New** → select **Layout Resource File** and give name as **details.xml**.

Once we create a new layout resource file **details.xml**, open it and write the code like as shown below

details.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical" android:layout_width="match_parent"
  android:layout_height="match_parent">
  <TextView
    android:layout_width="wrap_content"  android:layout_height="wrap_content"android:textSize="20dp"
    android:id="@+id/resultView"  android:layout_gravity="center" android:layout_marginTop="170dp" />
  <Button
   android:id="@+id/btnBack"  android:layout_width="wrap_content" android:layout_height="wrap_content"
```

```
    android:layout_gravity="center"    android:layout_marginTop="20dp"    android:text="Back" />
</LinearLayout>
```

Now open your main activity file **MainActivity.java** from **\java\com.example.externalstorageexample** path and write the code like as shown below

MainActivity.java

```
package com.example.externalstorageexample;
import android.content.Intent;
import android.os.Environment;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class MainActivity extends AppCompatActivity {
    EditText uname, pwd;
    Button saveBtn;
    FileOutputStream fstream;
    Intent intent;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        uname = (EditText)findViewById(R.id.txtName);
        pwd = (EditText)findViewById(R.id.txtPwd);
        saveBtn = (Button)findViewById(R.id.btnSave);
        saveBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String username = uname.getText().toString()+"\n";
                String password = pwd.getText().toString();
```

```
            try {

ActivityCompat.requestPermissions(MainActivity.this, new String[]{android.Manifest.permission.READ_EXTERNAL_STORAGE},23);
                File folder =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);
                File myFile = new File(folder,"user_details");
                fstream = new FileOutputStream(myFile);
                fstream.write(username.getBytes());
                fstream.write(password.getBytes());
                fstream.close();
                Toast.makeText(getApplicationContext(), "Details Saved in
"+myFile.getAbsolutePath(),Toast.LENGTH_SHORT).show();
                intent = new Intent(MainActivity.this,DetailsActivity.class);
                startActivity(intent);
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    });
  }
}
```

If you observe above code, we are taking entered username and password details and saving it in device external memory and redirecting the user to another activity.

Now we will create another activity file DetailsActivity.java in \java\com.example.externalstorageexample path to show the details from external storage for that right click on your application folder → Go to **New** → select **Java Class** and give name as **DetailsActivity.java**.

Once we create a new activity file **DetailsActivity.java**, open it and write the code like as shown below

DetailsActivity.java

```
package com.example.externalstorageexample;
import android.content.Intent;
import android.os.Bundle;
import android.os.Environment;
```

```
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.*;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
public class DetailsActivity extends AppCompatActivity {
    FileInputStream fstream;
    Intent intent;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.details);
        TextView result = (TextView)findViewById(R.id.resultView);
        Button back = (Button)findViewById(R.id.btnBack);
        try {
            File folder = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);
            File myFile = new File(folder,"user_details");
            fstream = new FileInputStream(myFile);
            StringBuffer sbuffer = new StringBuffer();
            int i;
            while ((i = fstream.read())!= -1){
                sbuffer.append((char)i);
            }
            fstream.close();
            String details[] = sbuffer.toString().split("\n");
            result.setText("Name: "+ details[0]+"\nPassword: "+details[1]);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        back.setOnClickListener(new View.OnClickListener() {
```

```
            @Override
            public void onClick(View v) {
                intent = new Intent(DetailsActivity.this,MainActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

Now we need to add this newly created activity in **AndroidManifest.xml** file in like as shown below.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.externalstorageexample">
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".DetailsActivity" android:label="External Storage - Details"></activity>
    </application>
</manifest>
```

If you observe above example, we are saving entered details file and redirecting the user to another activity file (**DetailsActivity.java**) to show the users details and added all the activities in **AndroidManifest.xml** file.

5.4. SQLite

SQLite is an open source light weight relational database management system (RDBMS) used to perform database operations, such as storing, updating, retrieving data from database. In our android applications Shared Preferences, Internal Storage and External Storage options are useful to store and maintain the small amount of data. In case, if we want to deal with large amount of data, then **SQLite database** is the preferable option to store and maintain the data in structured format.

By default, Android comes with built-in SQLite Database support so we don't need to do any configurations.

Just like we save the files on device's **internal storage**, Android stores our database in a private disk space that's associated to our application and the data is secure, because by default this area is not accessible to other applications.

The package **android.database.sqlite** contains all the required API's to use SQLite database in our android applications.

Now we will see how to create database and required tables in SQLite and perform CRUD (insert, update, delete and select) operations in android applications.

Create Database and Tables using SQLite Helper

In android, by using **SQLiteOpenHelper** class we can easily create required database and tables for our application. To use **SQLiteOpenHelper**, we need to create a subclass that overrides the **onCreate()** and **onUpgrade()** call-back methods.

Following is the code snippet of creating database and tables using **SQLiteOpenHelper** class in our android application.

```
public class DbHandler extends SQLiteOpenHelper {
    private static final int DB_VERSION = 1;
    private static final String DB_NAME = "usersdb";
    private static final String TABLE_Users = "userdetails";
    private static final String KEY_ID = "id";
    private static final String KEY_NAME = "name";
    private static final String KEY_LOC = "location";
    private static final String KEY_DESG = "designation";
    public DbHandler(Context context){
        super(context,DB_NAME, null, DB_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db){
        String CREATE_TABLE = "CREATE TABLE " + TABLE_Users + "("
                + KEY_ID + " INTEGER PRIMARY KEY AUTOINCREMENT," + KEY_NAME + " TEXT,"
                + KEY_LOC + " TEXT,"
                + KEY_DESG + " TEXT"+ ")";
        db.execSQL(CREATE_TABLE);
    }
```

```
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
        // Drop older table if exist
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_Users);
        // Create tables again
        onCreate(db);
    }
}
```

If you observe above code snippet, we are creating database "**usersdb**" and table "**userdetails**" using **SQLiteOpenHelper** class by overriding **onCreate** and **onUpgrade** methods.

Method	Description
onCreate()	This method is called only once throughout the application after database is created and the table creation statements can be written in this method.
onUpgrade()	This method is called whenever there is an updation in database like modifying the table structure, adding constraints to database, etc.

Now we will see how to perform CRUD (create, read, delete and update) operations in android applications.

Insert Data into SQLite Database

In android, we can insert data into SQLite database by passing **ContentValues** to **insert()** method.The following code snippet shows how to insert data into SQLite database using **insert()** method.

```
//Get the Data Repository in write mode
SQLiteDatabase db = this.getWritableDatabase();
//Create a new map of values, where column names are the keys
ContentValues cValues = new ContentValues();
cValues.put(KEY_NAME, name);
cValues.put(KEY_LOC, location);
cValues.put(KEY_DESG, designation);
// Insert the new row, returning the primary key value of the new row
long newRowId = db.insert(TABLE_Users,null, cValues);
```

If you observe above code, we are getting the data repository in write mode and adding required values to columns and inserting into database.

Read the Data from SQLite Database

In android, we can read the data from SQLite database using **query()** method.The following code snippet shows how to read the data from SQLite Database using **query()** method.

```
//Get the Data Repository in write mode
SQLiteDatabase db = this.getWritableDatabase();
Cursor cursor =
db.query(TABLE_Users, new String[]{KEY_NAME, KEY_LOC, KEY_DESG}, KEY_ID+ "=?",new String[]{String.value
Of(userid)},null, null, null, null);
```

If you observe above code, we are getting the details from required table using **query()** method based on our requirements.

Update Data in SQLite Database

In android, we can update the data in SQLite database using **update()** method in android applications.The following code snippet shows how to update the data in SQLite database using **update()** method.

```
//Get the Data Repository in write mode
SQLiteDatabase db = this.getWritableDatabase();
ContentValues cVals = new ContentValues();
cVals.put(KEY_LOC, location);
cVals.put(KEY_DESG, designation);
int count = db.update(TABLE_Users, cVals, KEY_ID+" = ?",new String[]{String.valueOf(id)});
```

If you observe above code, we are updating the details using **update()** method based on our requirements.

Delete Data from SQLite Database

In android, we can delete data from SQLite database using **delete()** method.The following code snippet shows how to delete the data from SQLite database using **delete()** method.

```
//Get the Data Repository in write mode
SQLiteDatabase db = this.getWritableDatabase();
db.delete(TABLE_Users, KEY_ID+" = ?",new String[]{String.valueOf(userid)});
```

If you observe above code, we are deleting the details using **delete()** method based on our requirements.Now we will see how to create SQLite database and perform CRUD (insert, update, delete, select) operations on SQLite Database in android application with examples.

Android SQLite Database Example

Following is the example of creating the SQLite database, insert and show the details from SQLite database into android listview using **SQLiteOpenHelper** class.

Create a new android application using android studio and give names as **SQLiteExample**. Once we create an application, create a class file **DbHandler.java** in **\java\com.example.sqliteexample** path to implement SQLite

database related activities for that right click on your application folder → Go to **New** → select **Java Class** and give name as **DbHandler.java**.

Once we create a new class file **DbHandler.java**, open it and write the code like as shown below

DbHandler.java

```
import android.content.*;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import java.util.ArrayList;
import java.util.HashMap;
public class DbHandler extends SQLiteOpenHelper {
   private static final int DB_VERSION = 1;
   private static final String DB_NAME = "usersdb";
   private static final String TABLE_Users = "userdetails";
   private static final String KEY_ID = "id";
   private static final String KEY_NAME = "name";
   private static final String KEY_LOC = "location";
   private static final String KEY_DESG = "designation";
   public DbHandler(Context context){
      super(context,DB_NAME, null, DB_VERSION);
   }
   @Override
   public void onCreate(SQLiteDatabase db){
      String CREATE_TABLE = "CREATE TABLE " + TABLE_Users + "("
            + KEY_ID + " INTEGER PRIMARY KEY AUTOINCREMENT," + KEY_NAME + " TEXT,"
            + KEY_LOC + " TEXT,"
            + KEY_DESG + " TEXT"+ ")";
      db.execSQL(CREATE_TABLE);
   }
   @Override
   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
      // Drop older table if exist
      db.execSQL("DROP TABLE IF EXISTS " + TABLE_Users);
      // Create tables again
      onCreate(db);
   }
   // **** CRUD (Create, Read, Update, Delete) Operations ***** //

   // Adding new User Details
   void insertUserDetails(String name, String location, String designation){
      //Get the Data Repository in write mode
      SQLiteDatabase db = this.getWritableDatabase();
      //Create a new map of values, where column names are the keys
      ContentValues cValues = new ContentValues();
      cValues.put(KEY_NAME, name);
      cValues.put(KEY_LOC, location);
      cValues.put(KEY_DESG, designation);
      // Insert the new row, returning the primary key value of the new row
```

```
        long newRowId = db.insert(TABLE_Users,null, cValues);
        db.close();
    }
    // Get User Details
    public ArrayList<HashMap<String, String>> GetUsers(){
        SQLiteDatabase db = this.getWritableDatabase();
        ArrayList<HashMap<String, String>> userList = new ArrayList<>();
        String query = "SELECT name, location, designation FROM "+ TABLE_Users;
        Cursor cursor = db.rawQuery(query,null);
        while (cursor.moveToNext()){
            HashMap<String,String> user = new HashMap<>();
            user.put("name",cursor.getString(cursor.getColumnIndex(KEY_NAME)));
            user.put("designation",cursor.getString(cursor.getColumnIndex(KEY_DESG)));
            user.put("location",cursor.getString(cursor.getColumnIndex(KEY_LOC)));
            userList.add(user);
        }
        return  userList;
    }
    // Get User Details based on userid
    public ArrayList<HashMap<String, String>> GetUserByUserId(int userid){
        SQLiteDatabase db = this.getWritableDatabase();
        ArrayList<HashMap<String, String>> userList = new ArrayList<>();
        String query = "SELECT name, location, designation FROM "+ TABLE_Users;
        Cursor cursor =
db.query(TABLE_Users, new String[]{KEY_NAME, KEY_LOC, KEY_DESG}, KEY_ID+ "=?",new String[]{String.value
Of(userid)},null, null, null, null);
        if (cursor.moveToNext()){
            HashMap<String,String> user = new HashMap<>();
            user.put("name",cursor.getString(cursor.getColumnIndex(KEY_NAME)));
            user.put("designation",cursor.getString(cursor.getColumnIndex(KEY_DESG)));
            user.put("location",cursor.getString(cursor.getColumnIndex(KEY_LOC)));
            userList.add(user);
        }
        return  userList;
    }
    // Delete User Details
    public void DeleteUser(int userid){
        SQLiteDatabase db = this.getWritableDatabase();
        db.delete(TABLE_Users, KEY_ID+" = ?",new String[]{String.valueOf(userid)});
        db.close();
    }
    // Update User Details
    public int UpdateUserDetails(String location, String designation, int id){
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues cVals = new ContentValues();
        cVals.put(KEY_LOC, location);
        cVals.put(KEY_DESG, designation);
        int count = db.update(TABLE_Users, cVals, KEY_ID+" = ?",new String[]{String.valueOf(id)});
        return  count;
```

```
    }
}
```

If you observe above code, we implemented all SQLite Database related activities to perform CRUD operations in android application.Now open **activity_main.xml** file from **\res\layout** folder path and write the code like as shown below.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/fstTxt"  android:layout_width="wrap_content"
        android:layout_height="wrap_content"    android:layout_marginLeft="100dp"
        android:layout_marginTop="150dp"     android:text="Name" />
    <EditText
        android:id="@+id/txtName"       android:layout_width="wrap_content"
        android:layout_height="wrap_content"   android:layout_marginLeft="100dp"      android:ems="10"/>
    <TextView
        android:id="@+id/secTxt"     android:layout_width="wrap_content"
        android:layout_height="wrap_content"   android:text="Location" android:layout_marginLeft="100dp" />
    <EditText
        android:id="@+id/txtLocation"  android:layout_width="wrap_content"
        android:layout_height="wrap_content"    android:layout_marginLeft="100dp"       android:ems="10" />
    <TextView
        android:id="@+id/thirdTxt"   android:layout_width="wrap_content"
  android:layout_height="wrap_content"   android:text="Designation" android:layout_marginLeft="100dp" />
    <EditText
        android:id="@+id/txtDesignation"      android:layout_width="wrap_content"
        android:layout_height="wrap_content"  android:layout_marginLeft="100dp"       android:ems="10" />
    <Button
        android:id="@+id/btnSave"  android:layout_width="wrap_content"
        android:layout_height="wrap_content"   android:layout_marginLeft="100dp"      android:text="Save" />
</LinearLayout>
```

Now we will create another layout resource file **details.xml** in **\res\layout** path to show the details in custom listviewfrom SQLite Database for that right click on your layout folder → Go to **New** → select **Layout Resource File** and give name as **details.xml**.Once we create a new layout resource file details.xml, open it and write the code like as shown below

details.xml

```
<?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" >
    <ListView
        android:id="@+id/user_list"   android:layout_width="fill_parent"
        android:layout_height="wrap_content"    android:dividerHeight="1dp" />
```

```
  <Button
     android:id="@+id/btnBack"     android:layout_width="wrap_content"android:text="Back"
 android:layout_height="wrap_content" android:layout_gravity="center"android:layout_marginTop="20dp"/>
</LinearLayout>
```

Create an another layout file (**list_row.xml**) in **/res/layout** folder to show the data in listview, for that right click on **layout** folder → add new **Layout resource file** → Give name as **list_row.xml** and write the code like as shown below.

list_row.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"    android:layout_height="wrap_content"
   android:orientation="horizontal"    android:padding="5dip" >
   <TextView
      android:id="@+id/name"  android:layout_width="wrap_content"  android:layout_height="wrap_content"
      android:textStyle="bold"   android:textSize="17dp" />
   <TextView
      android:id="@+id/designation"  android:layout_width="wrap_content"
      android:layout_height="wrap_content"   android:layout_below="@id/name"
      android:layout_marginTop="7dp" android:textColor="#343434"   android:textSize="14dp" />
   <TextView
      android:id="@+id/location" android:layout_width="wrap_content"
      android:layout_height="wrap_content"  android:layout_alignBaseline="@+id/designation"
      android:layout_alignBottom="@+id/designation"   android:layout_alignParentRight="true"
      android:textColor="#343434"      android:textSize="14dp" />
</RelativeLayout>
```

Now open your main activity file **MainActivity.java** from **\java\com.example.sqliteexample** path and write the code like as shown below

MainActivity.java

```
package com.example.sqliteexample;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
   EditText name, loc, desig;
   Button saveBtn;
   Intent intent;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      name = (EditText)findViewById(R.id.txtName);
      loc = (EditText)findViewById(R.id.txtLocation);
```

```
        desig = (EditText)findViewById(R.id.txtDesignation);
        saveBtn = (Button)findViewById(R.id.btnSave);
        saveBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String username = name.getText().toString()+"\n";
                String location = loc.getText().toString();
                String designation = desig.getText().toString();
                DbHandler dbHandler = new DbHandler(MainActivity.this);
                dbHandler.insertUserDetails(username,location,designation);
                intent = new Intent(MainActivity.this,DetailsActivity.class);
                startActivity(intent);
           Toast.makeText(getApplicationContext(), "Details Inserted Successfully",Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

If you observe above code, we are taking entered user details and inserting into SQLite database and redirecting the user to another activity.Now we will create another activity file **DetailsActivity.java** in **\java\com.example.sqliteexample** path to show the details from SQLite database for that right click on your application folder → Go to **New** → select **Java Class** and give name as **DetailsActivity.java**.Once we create a new activity file **DetailsActivity.java**, open it and write the code like as shown below

DetailsActivity.java

```
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.*;
import java.util.ArrayList;
import java.util.HashMap;

public class DetailsActivity extends AppCompatActivity {
    Intent intent;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.details);
        DbHandler db = new DbHandler(this);
        ArrayList<HashMap<String, String>> userList = db.GetUsers();
        ListView lv = (ListView) findViewById(R.id.user_list);
        ListAdapter adapter = new SimpleAdapter(DetailsActivity.this, userList,
R.layout.list_row,new String[]{"name","designation","location"}, new int[]{R.id.name, R.id.designation,
R.id.location});
        lv.setAdapter(adapter);
        Button back = (Button)findViewById(R.id.btnBack);
        back.setOnClickListener(new View.OnClickListener() {
```

```
            @Override
            public void onClick(View v) {
                intent = new Intent(DetailsActivity.this,MainActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

If you observe above code, we are getting the details from SQLite database and binding the details to android listview. Now we need to add this newly created activity in **AndroidManifest.xml** file in like as shown below.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sqliteexample">
    <application
        android:allowBackup="true"        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".DetailsActivity" android:label="SQLite Example - Details"></activity>
    </application>
</manifest>
```

If you observe above example, we are saving entered details in SQLite database and redirecting the user to another activity file (**DetailsActivity.java**) to show the users details and added all the activities in **AndroidManifest.xml** file

5.5. Web Service with PHP/Java &MySQL

External Databases are also the data repository for android apps for example apps that provide weather information, exchange rates, world news etc. MySQL is one of the database tool. Connection between an android app and mysql can be done through

- Web Services
- JDBC without web services

A web service is a standard used for exchanging information between applications or systems of heterogeneous type. Software applications written in various programming languages and running on various platforms can use web services to exchange information over Internet using http protocol.This inter-operability can be achieved between Java and Dot net applications, or PHP and Java applications.

A WebService is a Consumer_Machine-to-Provider_Machinecollaboration schema that operates over a computer network. The data exchanges occur independently of the OS, browser, platform,and programming languages used by the provider and the consumers.A provider may expose multiple **EndPoints** (sets of WebServices), each offering any number of typically related functions. WebServices expect the computer network to support standard Web protocols such as XML, HTTP, HTTPS, FTP, and SMTP. **Example:** Weather information, money exchange rates, world news, stock market quotation are examples of applications that can be modeledaround the notion of a remote data-services provider surrounded bycountless consumers tapping on the server's resources.

Why Web Service?

Here are the reasons for using Web Service:

- **Expose method as a service over network:** Web service is a chunk of code written in some programming language (Say C# or Java) that can be invoked remotely through http protocol. Once the Web methods are exposed publically, any applications can invoke it and use the functionality of the web methods.
- **Application Inter-operability – Connect heterogeneous applications:** With the help of web service, heterogeneos applications (Java and Dot Net / Java and PHP application) can communicate with each other. Web service written in Java can be invoked from PHP by passing the required parameters to web methods. This makes the applications platform and technology independant.
- **Standardized protocol:** Web Services uses standardized industry standard protocol for the communication which must be adhered and followed by the applications creating Web Service.
- **Cost effective communication:** Web service is using SOAP over HTTP protocol for communication which is much cheaper than systems like B2B.

What is the need of using Web Service in Android applications?

Existing Web applications are in a need of creating mobile applications to show their presence in mobile platform as well. Almost all web applications are having their Mobile applications created in Android, iOS or Windows platform.

Exposing the existing functionalities of the applications is bit tough as all the functionalities have to re-written in the respective platforms.

But it can be easily achieved with much ease by creating Web Service and expose the existing functionalities as web methods to Mobile platforms.

Here are the few advantages of using Web Service in Android:

- **Make client more lightweight:**Adding a web service layer makes the client more lightweight, both in terms of the required CPU power and the bandwidth used during the processing. Most of

the processing to be done in client end can be separated and put inside a web service layer which will be extremely helpful for end-users in terms of:

1. Using less CPU increases the battery life
2. Using less bandwidth reduces monthly payments over data charge

- **Re-usage of existing functionalities:**While designing the web service, we could also get significant benefits by reusing the existing functionalities by exposing them as web methods.
- **Remote DB hit made simple:** DB residing remotely can be hit from inside Android applications through Web Service calls. Making Web Service call from Android applications allows us to add functionality outside the scope of a DB like caching data, applying business rules over the data etc.,

Why should the Android developer learn how to create a WebService?

- Simple apps are usually self-contained and do not need to collaborate with other parties to obtain additional data or services (for instance, think of a scientific calculator)
- However, there are many cases in which the data needed to work with is very extensive, or changes very often and cannot (should not) be hardcoded into the app. Instead, this kind of data should be requested from a reliable external source (for instance, what is the Euro-to-Dollar rate of change right now?)
- Another class of apps requires a very high computing power perhaps not available in the mobile device (think of problems such as finding the shortest/fastest route between to mapped locations, or best air-fare & route selection for a traveler)
- It is wise for an Android developer to learn how to solve typical problems that exceed the capacities of the handheld devices. Understanding the possibilities offered by the client-server computing model will make the developer be a more complete and better professional.

Web Service Architecture

An ideal *Webservice* provider is designed around four logical layers which define the ways in which data is to be transported, encoded, exposed and discovered by the users.

Layers	Responsibility
Transport	Move messages through the network, using HTTP, SMTP, FTP, …
Messaging	Encoding of data to be exchanged (XML)
Description	WSDL (Web Service Desc. Lang) used for describing public methods available from the endpoint
Discovery	UDDI (Universal Description & Discovery Integration) facilitates location and publishing of services through a common registry

The Client Side - Consuming WebServices

There are two widely used forms of invoking and consuming Web Services:

Representational State Transfer (REST)

Closely tie to the HTTP protocol by associating its operation to the common methods: GET, POST, PUT, DELETE for HTTP/HTTPS.

This model has a simple invocation mode and little overhead. Service calls rely on a URL which may also carry arguments. Sender & receiver must understand how they pass data items from one another. As an example: Google Maps API uses the REST model.

Remote Procedure Call (RPC).

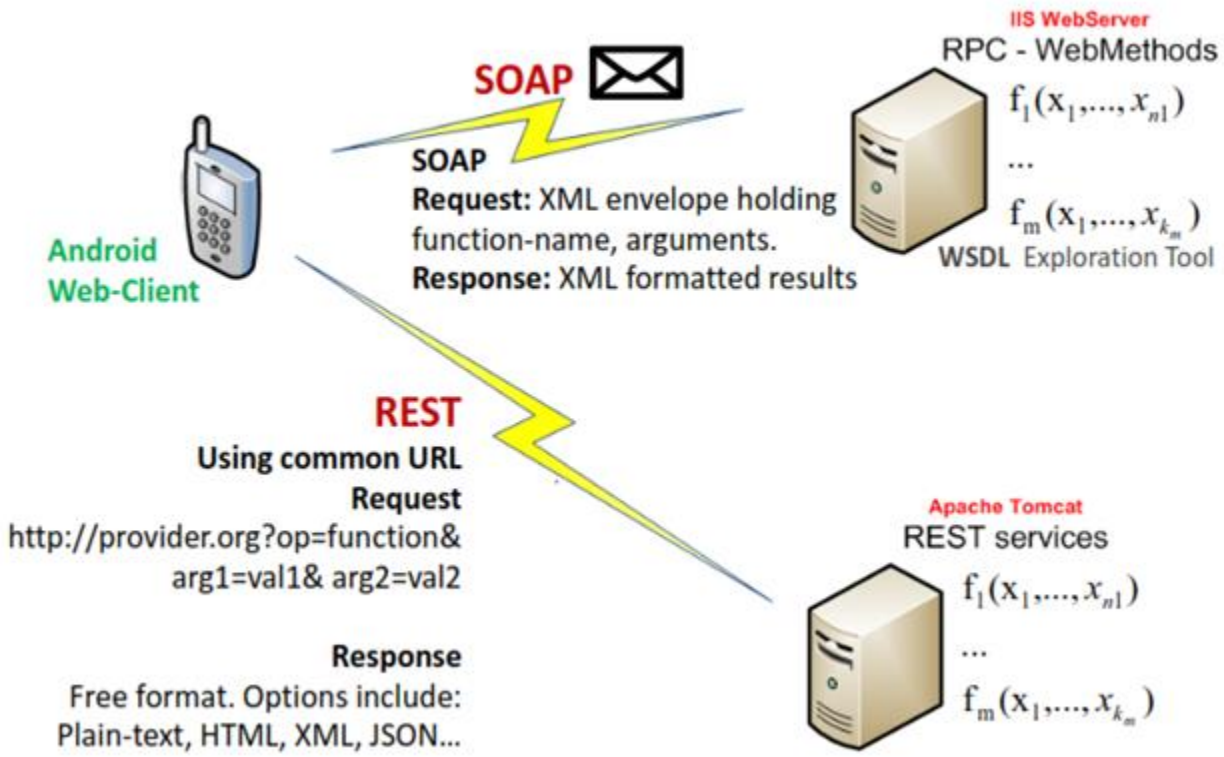
Remote services are seen as coherent collections of discoverable functions (or method calls) stored and exposed by EndPoint providers. Some implementations of this category include: Simple Object Access Protocol (SOAP), Common Object Request Broker Architecture (CORBA), Microsoft's Distributed Component Object Model (DCOM) and Sun Microsystems's Java/Remote Method Invocation (RMI).

REST vs. SOAP

Although SOAP and REST technologies accomplish the same final goal, that is request and receive a service, there are various differences between them.

- REST users refer to their remote services through a conventional URL that commonly includes the location of the (stateless) server, the service name, the function to be executed and the parameters needed by the function to operate (if any). Data is transported using HTTP/HTTPS.
- SOAP requires some scripting effort to create an XML envelop in which data travels. An additional overhead on the receiving end is needed to extract data from the XML envelope. SOAP accepts a variety of transport

mechanisms, among them HTTP, HTTPS, FTP, SMTP, etc. SOAP uses WSDL (Web Service Description Language) for exposing the format and operation of the services. REST lacks an equivalent exploratory tool.



<u>Example 1:</u>Android User Registration with PHP and MySQL (RESTful Web Service)

1. Create the backend web server. Download xampp or wamp.
2. Test your server by opening http://localhost and check mysql database server by opening http://localhost/phpmyadmin
3. Open the mysql database server and create a database called androidtest and table called users.

	#	Name	Type	Collation	Attributes
☐	1	name	varchar(30)	latin1_swedish_ci	
☐	2	fname	varchar(30)	latin1_swedish_ci	
☐	3	email	varchar(30)	latin1_swedish_ci	
☐	4	password	varchar(30)	latin1_swedish_ci	

4. Create a project folder named as test-android under xampp root directory i.e. c:\xampp\htdocs and create the following php file under your project folder
   a. Create database connection using php and save it as db-connect.php

```
<?php
$host="localhost";
$user="root";
$password="";
$db="androidtest";
$con=mysqli_connect($host,$user,$password,$db);
if(!$con) {
  echo '<h1>MySQL Server is not connected</h1>';
```

```
    }
    ?>
```

b. Create a php file that allow you to insert data to users table and name it register.php.

```
<?php
include("db-connect.php");
if(!$con) {
  echo '<h1>MySQL Server is not connected</h1>'.mysqli_connect_error($con);
}else{
            $name = $_POST["name"];
            $fname = $_POST["fname"];
            $email = $_POST["email"];
            $password = $_POST["password"];
   // Registration
   if(!empty($name) && !empty($fname) && !empty($email)&& !empty($password)){
      $json_registration=createNewRegisterUser($name,$fname,$email,$password);
      echo json_encode($json_registration);
   }
}
function createNewRegisterUser($name,$fname,$email,$password){
include("db-connect.php");
        $isExisting=isEmailExist($email);
        if($isExisting){
          $json['success'] = 0;
          $json['message'] = "Error in registering. Probably the username/email already exists";
}else{
          $query="insert into users(name,fname,email,password) values ('$name', '$fname',
'$email','$password')";
          $inserted = mysqli_query($con, $query);
if($inserted == 1){
            $json['success'] = 1;
            $json['message'] = "Successfully registered the user";
}else{
            $json['success'] = 0;
            $json['message'] = "Unable to register the user";
          }
          mysqli_close($con);
        }
        return $json;
      }
```

```
function isEmailExist($email){
include("db-connect.php");
    $query="select * from users where email='$email'";
  $result=mysqli_query($con, $query);
  if(mysqli_num_rows($result) >0){
    mysqli_close($con);
     return true;
  }
  return false;
 }
?>
```

5. Open Android Studio and add the following to your app's build gradle file

```
android{
…
useLibrary 'org.apache.http.legacy'

}
dependencies{ …
compile "org.apache.httpcomponents:httpcore:4.3.2"
}
```

6. Allow remote connection permission in AndroidManifest.xml file.

```
<manifest ...>

<uses-permission android:name="android.permission.INTERNET" />
```

7. Create an activity with the name **activity_jsonregistration.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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"
tools:context=".JSONREgistration"
android:orientation="vertical">

<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/dbresultmysql"/>
<EditText
android:id="@+id/usernameidmysql"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:ems="10"
android:inputType="textPersonName"
android:hint="Name" />

<EditText
android:id="@+id/userfathernameidmysql"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:ems="10"
android:inputType="textPersonName"
android:hint="Father Name" />
```

```
<EditText
android:id="@+id/useremailidmysql"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:ems="10"
android:inputType="textEmailAddress"
android:hint="Email"/>

<EditText
android:id="@+id/userpasswordidmysql"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:ems="10"
android:inputType="textPassword"
android:hint="PAssword"/>

<Button
android:id="@+id/registeruseridmysql"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Register User"
tools:layout_editor_absoluteX="81dp"
tools:layout_editor_absoluteY="73dp" />

<Button
android:id="@+id/viewuseridmysql"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="View User Profile"
tools:layout_editor_absoluteX="81dp"
tools:layout_editor_absoluteY="151dp" />
<Button
android:id="@+id/updateuseridmysql"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Update User Profile"
tools:layout_editor_absoluteX="81dp"
tools:layout_editor_absoluteY="151dp" />
<Button
android:id="@+id/deleteuseridmysql"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Delete User Profile"
tools:layout_editor_absoluteX="81dp"
tools:layout_editor_absoluteY="151dp" />
</LinearLayout>
```

JSONREgistration.java

```
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONException;
import org.json.JSONObject;
import java.util.ArrayList;

public class JSONRegistration extends AppCompatActivity {
    Button registerbtn;
    EditText emailmysql,namemysql,fnamemysql,passwordmysql;
```

```
    String name,fname,email,password;
    String URL= "http://10.0.2.2/test-android/register.php";
    JSONParser jsonParser=new JSONParser();
int i=0;

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

registerbtn = findViewById(R.id.registeruseridmysql);
emailmysql = findViewById(R.id.useremailidmysql);
namemysql = findViewById(R.id.usernameidmysql);
fnamemysql = findViewById(R.id.userfathernameidmysql);
passwordmysql = findViewById(R.id.userpasswordidmysql);
registerbtn.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
name = namemysql.getText().toString();
fname = fnamemysql.getText().toString();
email = emailmysql.getText().toString();
password = passwordmysql.getText().toString();
                AttemptRegistration attemptRegistration= new AttemptRegistration();
                    attemptRegistration.execute(name,fname,email,password);

                }

        });
    }
private class AttemptRegistration extends AsyncTask<String, String, JSONObject> {
@Override
protected void onPreExecute() {
super.onPreExecute();
        }
@Override
protected JSONObject doInBackground(String... args) {
            String name = args[0];
            String fname=args[1];
            String email= args[2];
            String password= args[3];
            ArrayList params = new ArrayList();
            params.add(new BasicNameValuePair("name", name));
            params.add(new BasicNameValuePair("fname", fname));
            params.add(new BasicNameValuePair("email", email));
            params.add(new BasicNameValuePair("password", password));
            JSONObject json = jsonParser.makeHttpRequest(URL, "POST", params);
return json;
        }
protected void onPostExecute(JSONObject result) {
try {
if (result != null) {

Toast.makeText(getApplicationContext(),result.getString("message"),Toast.LENGTH_LONG).show();
                } else {
                    Toast.makeText(getApplicationContext(), "Unable to retrieve any data from
server", Toast.LENGTH_LONG).show();
                }
            } catch (JSONException e) {
                Toast.makeText(JSONRegistration.this, e.getMessage(),
Toast.LENGTH_SHORT).show();
            }
        }

    }

}
```

8. Create JSONParser class (JSONParser.java)

```
import android.util.Log;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.utils.URLEncodedUtils;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
public class JSONParser {

static InputStream is = null;
static JSONObject jObj = null;
static JSONArray jArr = null;
static String json = "";
static String error = "";

// constructor
public JSONParser() {

    }

// function get json from url
    // by making HTTP POST or GET mehtod
public JSONObject makeHttpRequest(String url, String method,
                                    ArrayList params) {

// Making HTTP request
try {

// check for request method
if(method.equals("POST")){
// request method is POST
                // defaultHttpClient
HttpClient httpClient = new DefaultHttpClient();
                HttpPost httpPost = new HttpPost(url);
                httpPost.setEntity(new UrlEncodedFormEntity(params));
try {
                    Log.e("API123", " "
+convertStreamToString(httpPost.getEntity().getContent()));
                    Log.e("API123",httpPost.getURI().toString());
                } catch (Exception e) {
                    e.printStackTrace();
                }

                HttpResponse httpResponse = httpClient.execute(httpPost);
                Log.e("API123",""+httpResponse.getStatusLine().getStatusCode());
error= String.valueOf(httpResponse.getStatusLine().getStatusCode());
                HttpEntity httpEntity = httpResponse.getEntity();
is = httpEntity.getContent();

            }else if(method.equals("GET")){
// request method is GET
```

```
DefaultHttpClient httpClient = new DefaultHttpClient();
                String paramString = URLEncodedUtils.format(params, "utf-8");
                url += "?" + paramString;
                HttpGet httpGet = new HttpGet(url);

                HttpResponse httpResponse = httpClient.execute(httpGet);
                HttpEntity httpEntity = httpResponse.getEntity();
is = httpEntity.getContent();
            }

        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

try {
            BufferedReader reader = new BufferedReader(new InputStreamReader(
is, "utf-8"), 8);
            StringBuilder sb = new StringBuilder();
            String line = null;
while ((line = reader.readLine()) != null) {
                sb.append(line + "\n");
            }
is.close();
json = sb.toString();
            Log.d("API123",json);
        } catch (Exception e) {
            Log.e("Buffer Error", "Error converting result " + e.toString());
        }

// try to parse the string to a JSON object
try {
jObj = new JSONObject(json);
jObj.put("error_code",error);
        } catch (JSONException e) {
            Log.e("JSON Parser", "Error parsing data " + e.toString());
        }

// return JSON String
return jObj;

    }

private String convertStreamToString(InputStream is) throws Exception {
        BufferedReader reader = new BufferedReader(new InputStreamReader(is));
        StringBuilder sb = new StringBuilder();
        String line = null;
while ((line = reader.readLine()) != null) {
            sb.append(line);
        }
        is.close();
return sb.toString();
    }
}
```

9. Run the application and test the result.

Note:

- ***Use 10.0.2.2 address in the url(JSONREgistration.java) if you want to test the application using AVD.***

- ***Use the ip address of the server in the url(JSONREgistration.java) if you want to test the app on your device(mobile).***

<u>Example2:</u> Android User Registration with JDBC (No Web Service)

1. **Create an activity called UserRegistrationWithoutWebService**

activity_user_registration_without_web_service.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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"
tools:context=".UserRegistrationWithoutWebService"
android:orientation="vertical">

<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/dbresultmysqlnophp"/>
<EditText
android:id="@+id/usernameidmysqlnophp"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:ems="10"
android:inputType="textPersonName"
android:hint="Name" />

<EditText
android:id="@+id/userfathernameidmysqlnophp"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:ems="10"
android:inputType="textPersonName"
android:hint="Father Name" />

<EditText
android:id="@+id/useremailidmysqlnophp"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:ems="10"
android:inputType="textEmailAddress"
android:hint="Email"/>

<EditText
android:id="@+id/userpasswordidmysqlnophp"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:ems="10"
android:inputType="textPassword"
android:hint="Password"/>

<Button
android:id="@+id/registeruseridmysqlnophp"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Register User"
tools:layout_editor_absoluteX="81dp"
tools:layout_editor_absoluteY="73dp" />
</LinearLayout>
```

UserRegistrationWithoutWebService.java

```
import android.os.AsyncTask;
import android.os.Bundle;
```

```
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;

public class UserRegistrationWithoutWebServiceextends AppCompatActivity {

private static final String url = "jdbc:mysql://10.0.2.2:3306/androidtest";
private static final String user = "root";
private static final String pass = "";
    Button btnregister;
    EditText nametxt,fnametxt,emailtxt,passwordtxt;
    TextView resulttv;
    String name,fname,email,password;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user_registration_without_web_service);
resulttv = findViewById(R.id.dbresultmysqlnophp);
btnregister = findViewById(R.id.registeruseridmysqlnophp);
nametxt = findViewById(R.id.usernameidmysqlnophp);
fnametxt = findViewById(R.id.userfathernameidmysqlnophp);
emailtxt = findViewById(R.id.useremailidmysqlnophp);
passwordtxt = findViewById(R.id.userpasswordidmysqlnophp);
btnregister.setOnClickListener(new View.OnClickListener() {

@Override
public void onClick(View v) {
// TODO Auto-generated method stub
name=nametxt.getText().toString();
fname=fnametxt.getText().toString();
email=emailtxt.getText().toString();
password=passwordtxt.getText().toString();
                RegisterNewUser registernewuser = new RegisterNewUser();
                registernewuser.execute("");
            }
        });
    }

private class RegisterNewUser extends AsyncTask<String, Void, String> {
        String res = "";

@Override
protected void onPreExecute() {
super.onPreExecute();
            Toast.makeText(getApplicationContext(), "Please wait...",
Toast.LENGTH_SHORT).show();
          }

@Override
protected String doInBackground(String... params) {
try {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con = DriverManager.getConnection(url, user, pass);
                System.out.println("Database Connected!");
                String result = "Database Connection Successful\n";
                String sql="insert into users values(?,?,?,?)";
                PreparedStatement pstmt = con.prepareStatement(sql);
                pstmt.setString(1,name);
```

```
                pstmt.setString(2,fname);
                pstmt.setString(3,email);
                pstmt.setString(4,password);
if(pstmt.executeUpdate()>0)
res="User Registered Successfully";
else
res="User not registered!";
            } catch (Exception e) {
                e.printStackTrace();
res = e.toString();
            }
return res;
        }

@Override
protected void onPostExecute(String result) {
resulttv.setText(result);
        }
    }

}
```

Chapter 6

6. Android Telephony

6.1. Telephony Manager

The **android.telephony.TelephonyManager** class provides information about the telephony services such as subscriber id, sim serial number, phone network type etc. Moreover, you can determine the phone state etc.

Example: the following example shows how to display information of your telephony services using TelephoneManager class.

activity_main.xml

```
<RelativeLayout xmlns:androclass="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:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context=".MainActivity" >

   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_alignParentTop="true"
      android:layout_marginLeft="38dp"
      android:layout_marginTop="30dp"
      android:text="Phone Details:" />

</RelativeLayout>
```

MainActivity.java

```
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.telephony.TelephonyManager;
import android.view.Menu;
import android.widget.TextView;

public class MainActivity extends Activity {
  TextView textView1;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
            textView1=(TextView)findViewById(R.id.textView1);

        //Get the instance of TelephonyManager
TelephonyManager  tm=(TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
         //Calling the methods of TelephonyManager the returns the information
        String IMEINumber=tm.getIMEINumber();
        String subscriberID=tm.getDeviceId();
        String SIMSerialNumber=tm.getSimSerialNumber();
        String networkCountryISO=tm.getNetworkCountryIso();
        String SIMCountryISO=tm.getSimCountryIso();
        String softwareVersion=tm.getDeviceSoftwareVersion();
        String voiceMailNumber=tm.getVoiceMailNumber();

        //Get the phone type
        String strphoneType="";

        int phoneType=tm.getPhoneType();

        switch (phoneType)
        {
                case (TelephonyManager.PHONE_TYPE_CDMA):
                        strphoneType="CDMA";
                            break;
                case (TelephonyManager.PHONE_TYPE_GSM):
                        strphoneType="GSM";
                            break;
                case (TelephonyManager.PHONE_TYPE_NONE):
                         strphoneType="NONE";
                             break;
         }

        //getting information if phone is in roaming
        boolean isRoaming=tm.isNetworkRoaming();

        String info="Phone Details:\n";
        info+="\n IMEI Number:"+IMEINumber;
        info+="\n SubscriberID:"+subscriberID;
        info+="\n Sim Serial Number:"+SIMSerialNumber;
        info+="\n Network Country ISO:"+networkCountryISO;
        info+="\n SIM Country ISO:"+SIMCountryISO;
        info+="\n Software Version:"+softwareVersion;
        info+="\n Voice Mail Number:"+voiceMailNumber;
        info+="\n Phone Network Type:"+strphoneType;
        info+="\n In Roaming? :"+isRoaming;

        textView1.setText(info);//displaying the information in the textView
    }
}
```

AndroidManifest.xml

You need to provide **READ_PHONE_STATE** permission in the AndroidManifest.xml file.

AndroidManifest.xml

```
<manifest ...>
   <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
        ...
</manifest>
```

6.2. Phone Call

In android, we can easily make a phone call from our android applications by invoking built-in phone calls app using Intents action (**ACTION_CALL**). To make a phone call using Intent object in android application, we need to write the code like as shown below.

```
Intent callIntent = new Intent(Intent.ACTION_CALL);
callIntent.setData(Uri.parse("tel:" + txtPhone.getText().toString()));
startActivity(callIntent);
```

If you observe above code, we are using Intent object **ACTION_CALL** action to make a phone call based on our requirements.

Example

Following is the example of making a phone call by invoking default phone calls app using Intent object in android application. Create a new android application using android studio and give names as **PhoneCallExample**.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
      android:id="@+id/fstTxt"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginLeft="100dp"
      android:layout_marginTop="150dp"
      android:text="Mobile No"
      />
   <EditText
      android:id="@+id/mblTxt"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginLeft="100dp"
      android:ems="10">
   </EditText>
   <Button
      android:id="@+id/btnCall"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="100dp"
        android:text="Call" />
</LinearLayout>
```

Now open our main activity

file **MainActivity.java** from **\src\main\java\com.example.phonecallexample** path and write the code like as shown below

MainActivity.java

```
import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.net.Uri;
import android.os.Build;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {
    private EditText txtPhone;
    private Button btn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtPhone = (EditText)findViewById(R.id.mblTxt);
        btn = (Button)findViewById(R.id.btnCall);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                callPhoneNumber();
            }
        });
    }
    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults)
    {
        if(requestCode == 101)
        {
            if(grantResults[0] == PackageManager.PERMISSION_GRANTED)
            {               callPhoneNumber();           }
        }
    }

    public void callPhoneNumber()
```

```
  {
    try
    {
      if(Build.VERSION.SDK_INT > 22)
      {
        if (ActivityCompat.checkSelfPermission(this, Manifest.permission.CALL_PHONE) != Package
Manager.PERMISSION_GRANTED) {
          ActivityCompat.requestPermissions(MainActivity.this, new String[]{Manifest.permission.CA
LL_PHONE}, 101);
          return;
        }

        Intent callIntent = new Intent(Intent.ACTION_CALL);
        callIntent.setData(Uri.parse("tel:" + txtPhone.getText().toString()));
        startActivity(callIntent);

      }
      else {
        Intent callIntent = new Intent(Intent.ACTION_CALL);
        callIntent.setData(Uri.parse("tel:" + txtPhone.getText().toString()));
        startActivity(callIntent);
      }
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }
}
```

If you observe above code we are adding **Runtime** permissions to make sure our application work in both old / latest android OS versions and we used Intent action (**ACTION_CALL**) to make phone call on button click using default phone calls app. As discussed, we need to add a **CALL_PHONE** permission in our android manifest.

```
<manifest ...>
  <uses-permission android:name="android.permission.CALL_PHONE" />
...
</manifest>
```

6.3. Send SMS

In android, we can send SMS from our android application in two ways either by using **SMSManager** api or Intents based on our requirements. If we use **SMSManager** api, it will directly send SMS from our application. In case if we use Intent with proper action (**ACTION_VIEW**), it will invoke built-in SMS app to send SMS from our application.

Sending SMS using SMSManager API

In android, to send SMS using SMSManager API we need to write the code like as shown below.

```
SmsManager smgr = SmsManager.getDefault();
smgr.sendTextMessage(MobileNumber,null,Message,null,null);
```

SMSManager API required **SEND_SMS** permission in our android manifest to send SMS. Following is the code snippet to set **SEND_SMS** permissions in manifest file.

```
<uses-permission android:name="android.permission.SEND_SMS"/>
```

Android Send SMS using Intent

To send SMS using Intent object, we need to write the code like as shown below.

```
Intent sInt = new Intent(Intent.ACTION_VIEW);
sInt.putExtra("address", new String[]{txtMobile.getText().toString()});
sInt.putExtra("sms_body",txtMessage.getText().toString());
sInt.setType("vnd.android-dir/mms-sms");
```

Even for Intent, it required a **SEND_SMS** permission in our android manifest to send SMS. Following is the code snippet to set **SEND_SMS** permissions in manifest file.

```
<uses-permission android:name="android.permission.SEND_SMS"/>
```

Now we will see how to send SMS in android application using **SMSManager** API with examples.

Example: The following example shows how to send sms from our android app using **SMSManager** api.

Create a new android application using android studio and give names as **SendSMSExample**.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
      android:id="@+id/fstTxt"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginLeft="100dp"
      android:layout_marginTop="150dp"
      android:text="Mobile No" />
   <EditText
      android:id="@+id/mblTxt"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginLeft="100dp"
      android:ems="10"/>
   <TextView
      android:id="@+id/secTxt"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Message"
      android:layout_marginLeft="100dp" />
```

```
    <EditText
        android:id="@+id/msgTxt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="100dp"
        android:ems="10" />
    <Button
        android:id="@+id/btnSend"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="100dp"
        android:text="Send SMS" />
</LinearLayout>
```

Now open our main activity

file **MainActivity.java** from **\src\main\java\com.example.sendsmsexample** path and write the code like as shown below

MainActivity.java

```
import android.content.Intent;
import android.net.Uri;
import android.provider.Telephony;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    private EditText txtMobile;
    private EditText txtMessage;
    private Button btnSms;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtMobile = (EditText)findViewById(R.id.mblTxt);
        txtMessage = (EditText)findViewById(R.id.msgTxt);
        btnSms = (Button)findViewById(R.id.btnSend);
        btnSms.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                try{
                    SmsManager smgr = SmsManager.getDefault();
                    smgr.sendTextMessage(txtMobile.getText().toString(),null,txtMessage.getText().toString(),
null,null);
                    Toast.makeText(MainActivity.this, "SMS Sent Successfully", Toast.LENGTH_SHORT).show();
```

```
            }
            catch (Exception e){
Toast.makeText(MainActivity.this, "SMS Failed to Send, Please try again", Toast.LENGTH_SHORT).show(
);
            }
        }
    });
  }
}
```

Do not forget to add a **SEND_SMS** permission in our android manifest.

```
<manifest ...>
  <uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
...
</manifest>
```

if we want to use Intents to send SMS replace button click code like as shown below.

```
btnSms.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    try{
      Intent i = new Intent(Intent.ACTION_VIEW);
      i.setData(Uri.parse("smsto:"));
      i.setType("vnd.android-dir/mms-sms");
      i.putExtra("address", new String(txtMobile.getText().toString()));
      i.putExtra("sms_body",txtMessage.getText().toString());
      startActivity(Intent.createChooser(i, "Send sms via:"));
    }
    catch(Exception e){
Toast.makeText(MainActivity.this, "SMS Failed to Send, Please try again", Toast.LENGTH_SHORT).show(
);
    }
  }
});
```

6.4. Send Email

In android, we can easily send an email from our android application using existing email clients such as **GMAIL**, **Outlook**, etc. instead of building an email client from scratch. The Intent object in android with proper action (**ACTION_SEND**) and data will help us to launch the available email clients to send an

email in our application. To send an email using Intent object in android application, we need to write the code like as shown below.

```
Intent it = new Intent(Intent.ACTION_SEND);
it.putExtra(Intent.EXTRA_EMAIL, new String[]{"abc@gmail.com "});
it.putExtra(Intent.EXTRA_SUBJECT, "test");
it.putExtra(Intent.EXTRA_TEXT, "Hello this is to inform you that....");
it.setType("message/rfc822");
```

If you observe above code we used multiple components to send email, those are

- ✓ **it** - Our local implicit intent
- ✓ **ACTION_SEND** - It's an activity action which specifies that we are sending some data.
- ✓ **putExtra** - we use this **putExtra()** method to add extra information to our Intent. Here we can add following things.
  - EXTRA_EMAIL - It's an array of email addresses
  - EXTRA_SUBJECT - The subject of the email that we want to send
  - EXTRA_TEXT - The body of the email

The android Intent object is having different options such as EXTRA_CC, EXTRA_BCC, EXTRA_HTML_TEXT, EXTRA_STREAM, etc. to add different options for email client.

- ✓ **setType** - We use this property to set the MIME type of data that we want to send. Here we used "**message/rfc822**" and other MIME types are "**text/plain**" and "**image/jpg**".

Now we will see how to send an email in android application using Intent object with examples.

Example: Following is the example of sending an email with existing email clients using Intent in android application. Create a new android application using android studio and give names as **SendMailExample**. In case if you are not aware of creating an app in android studio check this article Android Hello World App.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="20dp"
  android:paddingRight="20dp"
  android:orientation="vertical" >
  <EditText
    android:id="@+id/txtTo"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="To"/>
  <EditText
```

```
        android:id="@+id/txtSub"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Subject"/>
    <EditText
        android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="Message"/>
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="Send"
        android:id="@+id/btnSend"/>
</LinearLayout>
```

Now open our

main activity file **MainActivity.java** from **\src\main\java\com.example.sendmailexample** path and write the code like as shown below

MainActivity.java

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

    private EditText eTo;
    private EditText eSubject;
    private EditText eMsg;
    private Button btn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        eTo = (EditText)findViewById(R.id.txtTo);
        eSubject = (EditText)findViewById(R.id.txtSub);
        eMsg = (EditText)findViewById(R.id.txtMsg);
        btn = (Button)findViewById(R.id.btnSend);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent it = new Intent(Intent.ACTION_SEND);
                it.putExtra(Intent.EXTRA_EMAIL, new String[]{eTo.getText().toString()});
```

```
            it.putExtra(Intent.EXTRA_SUBJECT,eSubject.getText().toString());
            it.putExtra(Intent.EXTRA_TEXT,eMsg.getText());
            it.setType("message/rfc822");
            startActivity(Intent.createChooser(it,"Choose Mail App"));
        }
    });
  }
}
```

We need to add **MIME type** in our android manifest file for that open android manifest file (**AndroidManifest.xml**) and write the code like as shown below

```
<manifest ...>
  <application
    ...
      <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
            <action android:name="android.intent.action.SEND"/>
            <category android:name="android.intent.category.DEFAULT"/>
            <data android:mimeType="message/rfc822"/>
        </intent-filter>
    </activity>
  </application>
</manifest>
```

If you observe above **AndroidManifest.xml** file we added following extra fields of Intent filters.

- **action** - we use this property to define that the activity can perform **SEND** action.
- **category** - we included the **DEFAULT** category for this activity to be able to receive implicit intents.
- **data** - the type of data the activity can send.

Chapter 7

7. Location in Android

7.1. Location-Based Service and Geocoding

What makes mobile so different is we can build advanced services that only mobile device with sensing can deliver. We do this all the time on our phone: do a location-based search for say supermarkets, cafe, cinema, users etc. Users take their devices everywhere and are constantly using them on the go, and as developers we can capitalize on that by providing a more contextual experience based on their current location. Accessing the current location of an Android device is easier than ever, but it can still be a little tricky, especially for the first time. Android phone uses the location of the phone as an input to the search. This is called location-based service. Examples DarkSky, MapMyFitness, Glympse, Uber etc. Android provides a number of building blocks for location-based services.

Android has two basic ways to determine a user's location.

The first is to use the **built-in location APIs** that have been available since Android was first introduced. These still work, but not as well as the newer location APIs bundled as part of the best known as **Google Play Services**

The Google Location Services API, part of Google Play Services, provides a more powerful, high-level framework that automates tasks such as location provider choice and power management. Location Services also provides new features such as activity detection that aren't available in the framework API. Developers who are using the framework API, as well as developers who are just now adding location-awareness to their apps, should strongly consider using the Location Services API.

Google Play Services is a bundle of services and APIs from Google that we can use for lots of features in Android apps. They are installed on a majority of devices and run on Android 2.3 and up. There are a lot of different parts of Google Play Services, but we will see how to include them in a project and use them for detecting location in a quick and effective way.

We need to download the Google Play Services SDK using the SDK Manager. It's a separate download just like a new version of the Android SDK. If you don't have it already, check out the Setting Up Google Play Services page on the Android developer site for details on how to download it. You may also want to download an emulator (AVD) image that uses the Google APIs.

Location manager

The Android location manager gives location in terms of longitude and latitude for the location of the phone. Depending on the location provider selected (could be based on GPS, WiFi or Cellular) the accuracy of the location will vary.

The key Android plumbing for location is:

- Location Manager, which provides the coordinates
- Location Providers, which can make a number of tradeoffs to offer the user the capability they want

A number of services can be built using these simple components:

- get the user's current location
- periodically get the user location as the move around -- provides a trail of bread crumbs
- use proximity alerts when you move in and out of a predefined area (e.g., Time Square)

The user gets a location manager by specifying the LOCATION_SERVICE as input to the getSystemService() of the activity. A location manager is returned.

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  LocationManager locationManager;
  String svcName = Context.LOCATION_SERVICE;
  locationManager = (LocationManager)getSystemService(svcName);
```

In the Manifest you will see that it is necessary to get the user's permission to track their location or get a location reading:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

There are fine and coarse permissions that the developer can use. The defaults is: if you ask for *ACCESS_FINE_LOCATION* then by default you get coarse. FINE is typically associated with GPS and coarse location with Cellular or network.

Location Provider

Mobile phones can provide location from a set of providers that make a number of tradeoffs. For example, GPS has good accuracy outdoors but is costly in terms of battery consumption for using the GPS chips on the phone. In contrast, Cellular is cheap in terms of energy consumption but could provide very coarse location information (say in the upper valley) because of the lack of cell tower density but could be great in the city. There are a number of tradeoffs that the user might want to make when selecting a location provider. Basically, depending on what location device, the user selects there are a number of different tradeoffs in:

- power consumption
- longitude/latitude accuracy
- altitude accuracy
- speed
- direction information

The user can specify the location provider explicitly in the code using a number of constants:

- LocationManager.GPS_PROVIDER
- LocationManager.NETWORK_PROVIDER
- LocationManager.PASSIVE_PROVIDER

It would be poor programming to rigidly specify the provider -- for example, the user might turn off GPS. It is better to let the Android systems match the user's needs to what providers are on offer. To do this we use *Criteria* as shown below. The code states that the user requires:

- coarse accuracy
- low power consumption

- no altitude, bearing or speed

The code snippet is taken again from onCreate(). The user specifies the level of location information and then asks the system for the best provider given what is currently available and the user's needs.

```
Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_FINE);
criteria.setPowerRequirement(Criteria.POWER_LOW);
criteria.setAltitudeRequired(false);
criteria.setBearingRequired(false);
criteria.setSpeedRequired(false);
criteria.setCostAllowed(true);

String provider = locationManager.getBestProvider(criteria, true);
```

Geocoding

Geocoding is the process of transforming a street address or other description of a location into a (latitude, longitude) coordinate. Geocoder supports two services:

- forward geocoding: from address to longitude/latitude
- reverse geocoding: from longitude/latitude to address. Reverse geocoding is the process of transforming a (latitude, longitude) coordinate into a (partial) address. The amount of detail in a reverse geocoded location description may vary, for example one might contain the full street address of the closest building, while another might contain only a city name and postal code. The Geocoder class requires a backend service that is not included in the core android framework.

The Geocoder class comes with the Google Maps library. To use the library you have to import it into the application. In addition, the Geocoder class uses a server to translate over the Internet so you need to add the following permission to the Manifest:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Example: Getting current location. The following example displays longitude and latitude of your current location.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<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"
      tools:context=".MainActivity">

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Get Current Location"
        android:id="@+id/getLocationBtn"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/locationText"
        android:layout_below="@id/getLocationBtn"/>
</RelativeLayout>
```

MainActivity.java

```
import android.content.Context;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity implements LocationListener {
    Button getLocationBtn;
    TextView locationText;

    LocationManager locationManager;

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

        getLocationBtn = (Button)findViewById(R.id.getLocationBtn);
        locationText = (TextView)findViewById(R.id.locationText);

        getLocationBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                getLocation();
            }
        });
    }

    void getLocation() {
        try {
            locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
            locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 5000,
5, this);
        }
        catch(SecurityException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onLocationChanged(Location location) {
        locationText.setText("Current Location: " + location.getLatitude() + ", " +
location.getLongitude());
    }

    @Override
    public void onProviderDisabled(String provider) {
        Toast.makeText(MainActivity.this, "Please Enable GPS and Internet",
Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {

    }

    @Override
    public void onProviderEnabled(String provider) {
```

```
    }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.currentlocationtest">
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

7.2. Integrating Google Map with android application

Google Maps is one of the many applications bundled with the Android platform. Android provides facility to integrate Google map in our application. Google map displays your current location, navigate location direction, search location etc. We can also customize Google map according to our requirement.

Types of Google Maps

There are four different types of Google maps, as well as an optional to no map at all. Each of them gives different view on map. These maps are as follow:

- **Normal:** This type of map displays typical road map, natural features like river and some features build by humans.
- **Hybrid:** This type of map displays satellite photograph data with typical road maps. It also displays road and feature labels.
- **Satellite:** Satellite type displays satellite photograph data, but doesn't display road and feature labels.
- **Terrain:** This type displays photographic data. This includes colors, contour lines and labels and perspective shading.
- **None:** This type displays an empty grid with no tiles loaded.

Syntax of different types of map

```
googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
```

Methods of Google map

Google map API provides several methods that help to customize Google map. These methods are as following:

Methods	Description
addCircle(CircleOptions options)	This method add circle to map.
addPolygon(PolygonOptions options)	This method add polygon to map.
addTileOverlay(TileOverlayOptions options)	This method add tile overlay to the map.
animateCamera(CameraUpdate update)	This method moves the map according to the update with an animation.
clear()	This method removes everything from the map.
getMyLocation()	This method returns the currently displayed user location.
moveCamera(CameraUpdate update)	This method reposition the camera according to the instructions defined in the update.
setTrafficEnabled(boolean enabled)	This method set the traffic layer on or off.
snapshot(GoogleMap.SnapshotReadyCallback callback)	This method takes a snapshot of the map.
stopAnimation()	This method stops the camera animation if there is any progress.

To use google maps in our android applications we need to install **Google Play Services** SDK in our Android Studio because google made **Google Mas API** as a part of Google Play Services SDK.

To install Google Play Services, open **Android Studio** → Go to **Tools** menu → **Android** → click **SDK Manager**, then new window will open in that select **SDK Tools** tab → Select **Google Play Services** → click **OK.** Once we are done with **Google Play Services** installation in android studio, now we will see how to integrate google map in android app.

1. Create an Android project and select Google maps activity.

2. Get a Google Map API key: Once project is created, Android Studio will open **google_maps_api.xml** and **MapsActivity.java** files in the editor. The **google_maps_api.xml** file will contains an instructions to generate a Google Maps API key to access Google Maps servers. Copy the link provided in the **google_maps_api.xml** file and paste the console URL in browser and it will take you to **Google API Console**. Follow the instructions to create a **new project** on **Google API Console**. Once we click on **continue** it will create a project and Google Maps Android API will be enabled. Now we need to create an API key to call the API for that click on **Create API Key**. Once we click on **Create API Key**, it will create an API key.

3. Now copy the API Key, go back to android studio and paste the API key into the **<string>** element in **google_maps_api.xml** file like as shown below.

```
<string name="google_maps_key" templateMergeStrategy="preserve" translatable="false">AIzaSyC
KPTaBv41DKqr9qxMPWOQAsqp0Q4NHMER</string>
```

4. We need to modify AndroidManifest.xml file by adding some user permission like:

- INTERNET: To determine if we are connected to the internet or not.
- ACCESS_FINE_LOCATION: To determine a user's location using GPS.
- ACCESS_COARSE_LOCATION: To determine user"s location using Wi-Fi and mobile data.

AndroidManifext.xml

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

    <!--
         The ACCESS_COARSE/FINE_LOCATION permissions are not required to use
         Google Maps Android API v2, but you must specify either coarse or fine
         location permissions for the 'MyLocation' functionality.
    -->
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
     <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_key" />

        <activity
            android:name=".MapsActivity"
            android:label="@string/title_activity_maps">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

activity_maps.xml

We are going to implement the android MapView in a Fragment and subsequently add the Fragment to the MainActivity class.

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

```
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MapsActivity"/>
```

MapsActivity.java

In MapsActivity.java class to get the Google map object, we need to implement the OnMapReadyCallback interface and override the onMapReady() callback method, these are called when the map is ready to be used.

Zoom Controls: We can control Zooming gestures on the map by using built-in *setZoomControlsEnabled()* function. These can be enabled by calling:

```
googleMap.getUiSettings().setZoomGesturesEnabled(true); // false to disable
```

Add Marker on Map: **We can place a marker to display location on the map by** addMarker()**method.**

```
LatLng addisababa= new LatLng(9.02497, 38.74689);
mMap.addMarker(new MarkerOptions().position(addisababa).title("Marker in
Addis Ababa"));
```

MapsActivity.java

```
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {

    private GoogleMap mMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
// Obtain the SupportMapFragment and get notified when the map is ready to be used.
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }

    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        LatLng addisababa = new LatLng(9.02497, 38.74689);
        mMap.addMarker(new MarkerOptions().position(addisababa).title("Addis Ababa in
Ethiopia"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(addisababa));

    }
}
```

build.gradle

In the app build.gradle file, we will add a Map dependency library. The modified version of the file is as shown:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.example.mygooglemapapplication"
        minSdkVersion 14
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.google.android.gms:play-services-maps:16.0.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'

}
```

Generally, during the launch of our activity, **onCreate()** callback method will be called by android framework to get the required layout for an activity. The output is as shown below.

Chapter 8

8. Publishing, Distributing, Monetizing, and Promoting Applications

8.1. Signing and Publishing Android Application

Android applications are distributed as Android package files (.APK). In order to be installed ona device or emulator, Android packages need to be signed.

During development, your applications will be signed using a debug key that is automaticallygenerated by the ADT tools. Before distributing your application beyond your testing environment,you must compile it as a release build and sign it using a private release key — typically using a selfsignedcertificate.

The JDK includes theKeytool and Jarsigner command-line tools necessary to create a newkeystore/signing certificate, and to sign your APK, respectively. Alternatively, you can use the Generate signed bundle/apk wizard, as described in the next section.

The importance of maintaining the security of your signing certificate can't beoverstated. Android usesthis certificate as the means of identifying the authenticityof applicationupdates,andapplyinginter-processsecurityboundariesbetweeninstalledapplications.Using a stolen key, a third party could sign and distribute applications thatmaliciously replace your authentic applications.Similarly, your certificate is the only way you can upgrade your applications.If you lose your certificate, it is impossible to perform a seamless update on adevice or from within Google Play. In the latter case, you would need to createa new listing, losing all the reviews, ratings, and comments associated with yourprevious package, as well as making it impossible to provide updates to theexisting users of your application.

1. Create Your Android App

First thing for uploading your App on Play Store is to Develop your App and make it Store-Ready.

Use Proper theme, colors, graphics, strings and icons.

2. Make its Signed Apk(Using Release Keystore)

To generate keystores for signing Android apps at the command line, use:

$ keytool -genkey -v -keystore my-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 10000

A debug keystore which is used to sign an Android app during development needs a specific alias and password combination as dictated by Google. To create a debug keystore, use:

$ keytool -genkey -v -keystoredebug.keystore -storepass android -alias androiddebugkey -keypass android -keyalg RSA -keysize 2048 -validity 10000

Keystore name: "debug.keystore"

Keystore password: "android"

Key alias: "androiddebugkey"

Key password: "android"

To generate signed bundle/apk using Android Application Wizard

- Go to Build→Generate Signed Bundle/APK→APK→ and the go through the steps to finish the signing process.

3. Login to your Gmail Account and visit this link: https://play.google.com/apps/publish/

4. Create a Merchant Account

Note: It will charge you once in a lifetime fee i.e 25$. Just do it to start uploading your first app.

5. Mandatory Fields needed in App Listing: Title, Short Desc, Full Desc, App Screenshots(JPEG or 24-bit PNG (no alpha))(Min-2,Max-8)(Min-320px,Max-3840px), Hi-res icon(512 x 512)(32-bit PNG (with alpha)), Feature Graphic(1024 w x 500 h)(JPG or 24-bit PNG (no alpha)), App Type, Category, Content Rating, Developer/Company Email, Privacy Policy Url, And some other details

How to Publish an App on Google Play?

With over 1 billion monthly active users, Google Play is arguably one of the largest platforms for distributing, promoting, and selling Android apps.

For newbies and first-timers, publishing an app on Google Play can seem like an intimidating process. Like any other app store, the platform comes with its own set of rules, regulations, and procedures, and you need to understand how it works in order to avoid any future issues.

Here is a step-by-step guide on how to successfully publish your app on the Google Play Store.

Step 1: Create a Developer Account

Before you can publish any app on Google Play, you need to create a Developer Account. You can easily sign up for one using your existing Google Account.The signup process is fairly straightforward, and you'll need to pay a one-time registration fee of $25. After you've reviewed and accepted the Developer Distribution Agreement, you can proceed to make the payment using your credit or debit card.To finish the sign up process, fill out all your necessary account details, including your Developer Name, which will be visible to your customers on Google Play. You can always add more details later.Also, do remember that it can take up to 48 hours for your registration to be fully processed.

Step 2: Plan to Sell? Link Your Merchant Account

If you want to publish a paid app or plan to sell in-app purchases, you need to create a payments center profile, i.e. a merchant account. Here's how you can do that:

1. Sign in to your Play Console
2. Click on Download Reports – Financial
3. Select 'Set up a merchant account now'
4. Fill out your business information

Once you create the profile, it will be automatically linked to your developer account.A merchant account will let you manage your app sales and monthly payouts, as well as analyze your sales reports right in your Play Console.

Step 3: Create an App

Now that you have set up your Play Console, you can finally add your app. Here's how to do that:

1. Navigate to the 'All applications' tab in the menu
2. Click on 'Create Application'

3. Select your app's default language from the drop-down menu
4. Type in a title for your app
5. Click on "Create"

The title of your app will show on Google Play after you've published. Don't worry too much about it at this stage; you can always change the name later.

After you've created your app, you'll be taken to the store entry page. Here, you will need to fill out all the details for your app's store listing.

Step 4: Prepare Store Listing

Before you can publish your app, you need to prepare its store listing. These are all the details that will show up to customers on your app's listing on Google Play.

Note: You don't necessarily have to complete this step before moving on to the next one. You can always save a draft and revisit it later when you're ready to publish.

The information required for your store listing is divided into several categories:

Product Details

There are three fields here that you need to fill out:

Field	Description	Character Limit	Notes
Title	Your app's name on Google Play.	50 character limit	You can add one localized title per language.
Short description	The first text users see when looking at your app's detail page on the Play Store app.	80 character limit	Users can expand this text to view your app's full description.
Full description	Your app's description on Google Play.	4000 character limit	

Your app's title and description should be written with a great user experience in mind.Use the right keywords, but don't overdo it. Make sure your app doesn't come across as spamor promotional, or it will risk getting suspended on the Play Store.

Graphic Assets

Under graphic assets, you can add screenshots, images, videos, promotional graphics, and icons that showcase your app's features and functionality.Some parts under graphic assets are mandatory, like screenshots, a feature graphic, and a high-resolution icon. Others are optional, but you can add them to make your app look more attractive to users.There are specific requirements for each graphic asset that you upload, such as the file format and dimensions. You can read more about each requirement here.

Languages & Translations

You can also add translations of your app's information in the store listing details, along with in-language screenshots and other localized images.There's also an option for users to view automated translations of your app's information using Google Translate, in case you don't add your own translations.

Categorization

This part requires you to select the appropriate type and category your app belongs to. From the drop-down menu, you can pick either app or game for the application type.There are various categories for each type of app available on the Play Store. Pick the one your app fits into best. In order to rate your content, you'll need to upload an APK first. You can skip this step for later.

Contact Details

This part requires you to enter contact details to offer your customers access to support regarding your app.

You can add multiple contact channels here, like an email, website, and phone number, but providing a contact email is mandatory for publishing an app.

Privacy Policy

For apps that request access to sensitive user data or permissions, you need to enter a comprehensive privacy policy that effectively discloses how your app collects, uses, and shares that data.You must add a URL linking to your privacy policy in your store listing and within your app. Make sure the link is active and relevant to your app.

You're now done with the store listing. Go ahead and click on 'Save Draft' to save your details. You can always skip some steps and come back to them later before you publish your app.

Step 5: Upload APK to an App Release

Now that you have prepared the ground to finally upload your app, it's time to dig out your APK file.

The Android Package Kit (or APK, for short) is the file format used by the Android operating system to distribute and install apps. Simply put, your APK file contains all the elements needed for your app to actually work on a device.Google offers you multiple ways to upload and release your APK. Before you upload the file, however, you need to create an app release.

To create a release, select the app you created in Step 3. Then, from the menu on the left side, navigate to 'Release management' -> 'App releases.'

Here, you need to select the type of release you want to upload your first app version to. You can choose between an internal test, a closed test, an open test, and a production release.The first three releases allow you to test out your app among a select group of users before you make it go live for everyone to access.

This is a safer option because you can analyze the test results and optimize or fix your app accordingly if you need to before rolling it out to all users.However, if you create a production release, your uploaded app version will become accessible to everyone in the countries you choose to distribute it in.

Once you've picked an option, click on 'Create release.'Next, follow the on-screen instructions to add your APK files, and name and describe your release.After you're done, press Save.

Step 6: Provide an Appropriate Content Rating

If you don't assign a rating to your app, it will be listed as 'Unrated'. Apps that are 'Unrated' may get removed from Google Play.To rate your app, you need to fill out a content rating questionnaire. You can access it when you select your app in the Play Console, and navigate to 'Store presence' – 'Content rating' on the left menu.Make sure you enter accurate information. Misrepresentation of your app's

content can lead to suspension or removal from the Play Store.An appropriate content rating will also help you get to the right audience, which will eventually improve your engagement rates.

Step 7: Set Up Pricing & Distribution

Before you can fill out the details required in this step, you need to determine your app's monetization strategy.Once you know how your app is going to make money, you can go ahead and set up your app as free or paid.Remember, you can always change your app from paid to free later, but you cannot change a free app to paid. For that, you'll need to create a new app and set its price.

You can also choose the countries you wish to distribute your app in, and opt-in to distribute to specific Android devices and programs too.

Step 8: Rollout Release to Publish Your App

You're almost done. The final step involves reviewing and rolling out your release after making sure you've taken care of everything else.Before you review and rollout your release, make sure the store listing, content rating, and pricing and distribution sections of your app each have a green check mark next to them.

Once you're sure you've filled out those details, select your app and navigate to 'Release management' – 'App releases.' Press 'Edit release' next to your desired release, and review it.Next, click on 'Review' to be taken to the 'Review and rollout release' screen. Here, you can see if there are any issues or warnings you might have missed out on.Finally, select 'Confirm rollout.' This will also publish your app to all users in your target countries on Google Play.

8.2. Distributing Applications

One of the advantages of Android's open ecosystem is the freedom to publish and distribute your applications however, and wherever, you choose. The most common and popular distribution channel is Google Play; however, you are free to distribute your applications using alternative markets, your own website, social media, or any other distribution channel.

In addition to Google Play, there are several alternatives of varying reach, including OEM and carrier pre-installs, the Amazon App Store, and carrier-specific stores.

When distributing your application, it's important to note that application package names are usedas unique identifiers for each application. As a result, each application — including variations thatyou plan to distribute separately — must each have a unique package name. Note that the filenameof your APK does not have to be unique — it will be discarded during the installation process (onlythe package name is used).

8.3. MonetizingApplication

As an open ecosystem, Android enables you to monetize your applications using whatever mechanism you choose.

If you choose to distribute and monetize your applications using Google Play, three options typically are available:

- ✓ **Paid applications** — Charge users an upfront fee before they download and install your application.
- ✓ **Free applications with In-App Billing (IAB)** — Make the download and installation of the application free, but charge within the application for virtual goods, upgrades, and other value-adds.
- ✓ **Advertising-supported applications** — Distribute the application for free, and monetize it by displaying advertising.

Although paid applications and advertising-supported applications are the traditional mechanisms for monetizing mobile applications, IAB has emerged as an extremely effective alternative. If you choose to charge for your applications on Google Play, either through upfront charges or IAB,the revenue is split between you and Google Play in the form of a transaction fee. The revenue split is set at 70 percent for the developer and 30% for Google play.

In order to use either approach, you must first create a Google Checkout Merchant Account, youcan do this from your Android publisher account. Your application listings will then include the option to set a price for the application and the items sold using IAB.

In each case you are the application distributor and merchant of record, so you are responsible for any legal or taxation obligations associated with the sale of your application, subject to the terms described in the DDA.

You can also monetize your application using in-app advertising. The specific process required to setup advertising within your application will vary depending on the ads provider you choose. The general process could be described as follows:

a) Create a publisher account.
b) Download and install the associated ads SDK.
c) Update your Fragment or Activity layouts to include an add banner.

It's important to ensure that any ads included within your application are as unobtrusive as possible and don't detract significantly from the user experience of your application. It's also important toensure that your user interaction model doesn't encourage accidental clicks on the ad banner. In many cases, developers have chosen to offer a paid alternative (either using up-front paymentor IAB) to allow users to eliminate ad banners from their applications.

8.4. Application Marketing and Promotion Strategies

The first step in effectively marketing and promoting your application is ensuring that you providethe full set of high-quality assets for your Google Play listing. Several promotional opportunities are available within Google Play.However, with more than 2.6million other applications available, it's important that you consideralternative avenues for marketing and promotion rather than simply launching your application.

While your marketing and promotion strategies will vary widely depending on your goals andbudget, the following list details some of the most effective techniques to consider:

- ✓ **Offline cross promotion** — If you have a significant offline presence (such as a stores orbranches), or a large media presence (such as within newspapers, magazines, or on TV),cross promoting your application through those channels can be a particularly effective wayto increase awareness and help to ensure users trust the download. Traditional advertisingtechniques such as TV and newspaper advertisements can be extremely effective in raisingawareness of your application.
- ✓ **Online cross promotion** — If you have a significant web presence, promoting your applicationthrough direct links to Google Play can be an effective way to drive downloads. Ifyourapplication provides a better user experience than your mobile website, you can detectbrowservisitors from Android devices and direct them to Google Play to download yournativeapp.
- ✓ **Third-party promotion** — Distributing a promotional video on YouTube and leveragingsocial networks, blogs, press releases, and online review sites can help provide positive wordof mouth.
- ✓ **Online advertising** — Online advertising using in-app ad networks (such as AdMob) or traditionalsearch-based advertising (such as Google AdWords) can drive significant impressionsanddownloads for your application.

Application Launch Strategies

Ratings and reviews can have a significant impact on your application's ranking in category lists andwithin Google Play search results. As a result, it can be difficult to recover from a poor launch. Thefollowing list describes some of the strategies you can use to ensure a successful launch:

- ✓ **Iterate on features not quality**- A poorly implemented but feature-rich application willreceive worse reviews than a well-polished application that doesn't do everything. If you areusing an agile approach of releasing early and often, ensure each release is of the same highquality, adding new features as part of each release. Similarly, each release should be morepolished and stable than the last.
- ✓ **Create high quality Google Play assets -** The first impression your application makesis through its appearance in Google Play. Maximize the likelihood of that impressionresulting in an installation by creating assets that represent the quality of yourapplication.
- ✓ **Be honest and descriptive**- Disappointed users who find your application is not as it wasdescribed are likely to uninstall it, rate it poorly, and leave negative comments.

Promotion Within Google Play

In addition to the effect of reviews, downloads, and installs will have on your Google Play listing,there are several automated and curated lists that are used by the Google Play editorial team tohighlight high quality applications.Additionally, a small number of applications are chosen as "featured" applications, receiving priorityplacementinGooglePlay.

Featuredapplicationstypicallyreceivea significant boostindownloadnumbers,makingfeaturinghighlyprizedgoal.While the criteria used to determine which applications become featured is not publicly available,therearecertaincriteriathathavegenerallybecomeassociatedwithfeaturedapplications,including:

- ✓ **High quality and innovative**- The featured applications in Google Play act as a showcasefor the platform. As a result, the easiest way to be featured is by creating a high qualityapplication that is useful and innovative.
- ✓ **A high degree of fi t-and-finish -** The listings for featured applications include all the requisite promotional assets, while the applications themselves have few bugs and a highquality user interface.
- ✓ **Broad device and platform support**- Featured applications typically support a broad rangeof device types and platform versions, including both handsets and tablets.
- ✓ **Use of newly released features**- Applications that leverage hardware features and softwareAPIs from new Android platform releases offer an opportunity for the Google Play team tohighlight those new features to reviewers and early adopters.
- ✓ **Consistency with the platform user experience**- Featured applications provide a compellinguser experience that is consistent with the UI and interaction models offered by the Androidplatform.

Analytics and Referral Tracking

Mobile application analytics packages, such as Google Analytics and Flurry, can be effective tools for you to better understand who is using your application and how they are using it. Understanding this information can help you make objective decisions on where to focus your development resources. While the statistics provided by the Android Developer Console offer valuable insight into your users' language, country, and handsets, using detailed analytics can provide a much richer source of information, from which you can discover bugs, prioritize your feature list, and decide where best to allocate your development resources.

Broadly speaking, you can track three types of data within your application:

- ✓ **User analytics**-Understand the geographic locations (and language settings) of your users,as well as the speed of their Internet connections, their screen sizes and resolutions, and theorientation of their displays. Use this information to prioritize your translation efforts andoptimize your layout and assets for different screen sizes and resolutions.
- ✓ **Application usage patterns**- The first step in integrating analytics is to record each Activityas you would a web site. This will help you understand the way your application is beingused, and will help you to optimize your workflows in the same way you would a web site. Taken one step further, you can record any action — which options were changed, which menu items or Action Bar actions were selected, which popup menus were displayed, if a Widget was added, and which buttons were pressed. Using this information, you can determine exactly how your application is being used, allowing you to better understand how well the assumptions you made during design match actual usage.When building games, you can use the same process to gain insight into players' progress though the game. You can track how far people progress before quitting, identify levels that are more difficult (or easier) than you expected, and then modify your game accordingly.

- ✓ **Exception tracking-** In addition to printing an error into the log output, post each unique exception thrown using your analytics. Not only does this alert you to which exceptions are being thrown, you will also gain insight into their context. Specifically, you'll be able to see if there are particular devices, locations, or usage patterns that lead to particular exceptions.

While it's important to track as much analytic information as possible, care must be taken when transmitting that data back to the analytics server. Every time a new data connection is created, the wireless radio may be activated and it will continue to draw power for up to 20 seconds on a typical 3G wireless radio.

As a result, it's important to bundle the analytics information you collect and queue it for transfer the next time your application transfers data, rather than transmitting it as it's collected.