Plan your MSBTE Journey with MSBTE NATION app

GET IT ON Google Play

Unit 1- Android and its Tools

Course Outcome:

Interpret features of Android operating system.

Unit Outcomes:

1a. Explain the given basic terms related to Android system.
1b. Explain with sketches Android architecture for the given application.
1c. Identify tools and software required for developing the given Android application with justification.

Contents:

1.1 Introduction to android**,** Open Handset Alliance, Android Ecosystem
1.2 Need of Android, Features of Android
1.3 Tools and software required for developing an Android Application
1.4 Android Architecture

1.1 Introduction to android, Open Handset Alliance, Android Ecosystem

Introduction to android

- **Android** is an open source operating system based on Linux with a Java programming interface for mobile devices such as Smartphone (Touch Screen Devices who supports Android OS) as well for Tablets too.
- The operating system has developed a lot in last 15 years starting from black and white phones to recent smart phones or mini computers. One of the most widely used mobile OS these days is android.
- The android is a powerful operating system and it supports large number of applications in Smartphones.

What is Android?

Android is a stack of software for mobile devices that are an Operating System, Middleware and Key Applications.

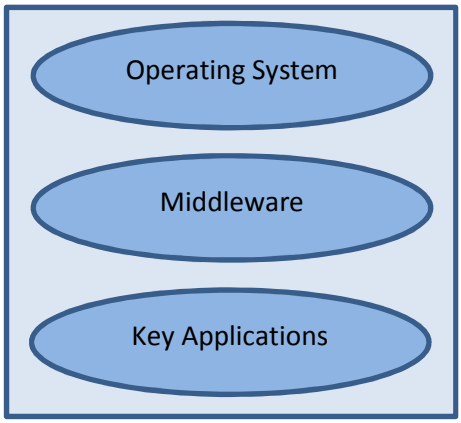


Figure 1.1: Android Operating System

- Android is a Linux-based operating system which is designed for touchscreen mobile devices like smartphones and tablet computers.
- It is an open source technology that allows the software to be freely modified and distributed by device manufacturers, wireless carriers and developers.
- Android was unveiled during 2007 along with the founding of the Open Handset Alliance.

Open Handset Alliance

The Open Handset Alliance (OHA) is a association whose goal is to develop open standards for mobile devices, promote innovation in mobile phones and provide a better experience for consumers at a lower cost.
The OHA is the group that is in charge of the Android smartphones operating system. It was created by Google.

The **Open Handset Alliance** (OHA) is consortium of multiple companies like Samsung, Sony, Intel and many more to provide services and deploy handsets using android platform.

Android Ecosystem

Ecosystem in Market terminology refers to the inter-dependence between demand and supply.

In the Android ecosystem this translates to inter-dependence between users, developers, and equipment makers. One cannot exist without the other:

- Users- buy devices and applications
- Equipment makers- sell devices, sometimes bundled with applications
- Developers- buy devices, then make and sell applications

1.2 Need of Android, Features of Android

Need of Android

There are so many reasons you should choose Android platform for mobile application development.

1. Zero/negligible development cost

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

2. Open Source

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

3. Multi-Platform Support

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

4. Multi-Carrier Support

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

5. Open Distribution Model

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

Features of Android

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

Feature	Description
Connectivity	Android supports multiple connectivity technologies including GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX

Storage	SQLite, a lightweight relational database, is used for data storage purposes
Media support	Android supports various type of audio/video/still media formats like: H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, BMP and WebP
Web browser	The web browser available in Android is based on the open-source Blink (previously WebKit) layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3
Messaging	SMS and MMS are available forms of messaging, it also include threaded text messaging and Android Cloud To Device Messaging (C2DM) and now support the enhanced version of C2DM, Android Google Cloud Messaging (GCM) is also a part of Android Push Messaging services
Multi-tasking	Multitasking of applications, with unique handling of memory allocation, is available, using this user can jump from one task to another and at the same time various application can run simultaneously
Resizable widgets	Widgets are re-sizable, so users can expand them to show more content or shrink them to save space
Multi-touch	Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero
Wi-Fi	A technology that lets apps discover and pair directly, over a high-bandwidth peer-to-peer connection.
Screen capture	Android supports capturing a screenshot by pressing the power and home-screen buttons at the same time. This features supports after Android 4.0
Android Beam	A popular NFC-based technology that lets users instantly share, just by touching two NFC-enabled phones together
Multi-Language	Android supports multiple languages, also supports single direction and bi-directional text

1.3 Tools and software required for developing an Android Application

The android developer tools let you create interactive and powerful application for android platform. The tools can be generally categorized into two types.

- SDK tools
- Platform tools

SDK tools

SDK tools are generally platform independent and are required no matter which android platform you are working on. When you install the Android SDK into your

system, these tools get automatically installed. The list of SDK tools has been given below –

Sr. No	Tool & description
1	**android** This tool lets you manage AVDs, projects, and the installed components of the SDK
2	**ddms** This tool lets you debug Android applications
3	**Draw 9-Patch** This tool allows you to easily create a NinePatch graphic using a WYSIWYG editor
4	**emulator** This tools let you test your applications without using a physical device
5	**mksdcard** Helps you create a disk image (external sdcard storage) that you can use with the emulator
6	**proguard** Shrinks, optimizes, and obfuscates your code by removing unused code
7	**sqlite3** Lets you access the SQLite data files created and used by Android applications
8	**traceview** Provides a graphical viewer for execution logs saved by your application
9	**Adb** Android Debug Bridge (adb) is a versatile command line tool that lets you communicate with an emulator instance or connected Android-powered device.

Three important tools are android, ddms and sqlite3.

1. **Android**

Android is a development tool that lets you perform these tasks:

- Manage Android Virtual Devices (AVD)

- Create and update Android projects
- Update your sdk with new platform add-ons and documentation

2. DDMS

DDMS stands for Dalvik debug monitor server that provides many services on the device. The service could include message formation, call spoofing, capturing screenshot, exploring internal threads and file systems etc.

Running DDMS

From Android studio click on Tools>Android>Android device Monitor.

How it works

- In android, each application runs in its own process and each process run in the virtual machine. Each VM exposes a unique port, that a debugger can attach to.
- When DDMS starts, it connects to adb. When a device is connected, a VM monitoring service is created between adb and DDMS, which notifies DDMS when a VM on the device is started or terminated.

3. Sqlite3

- Sqlite3 is a command line program which is used to manage the SQLite databases created by Android applications. The tool also allows us to execute the SQL statements on the fly.
- There are two ways through which you can use SQLite, either from remote shell or you can use locally.

Platform tools

- The platform tools are customized to support the features of the latest android platform.
- The platform tools are typically updated every time you install a new SDK platform. Each update of the platform tools is backward compatible with older platforms.
- Some of the platform tools are listd below –
  - Android Debug bridge (ADB)
  - Android Interface definition language (AIDL)
  - aapt, dexdump and dex etc.

1.4 Android Architecture

Android architecture or Android software stack is categorized into five parts:

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

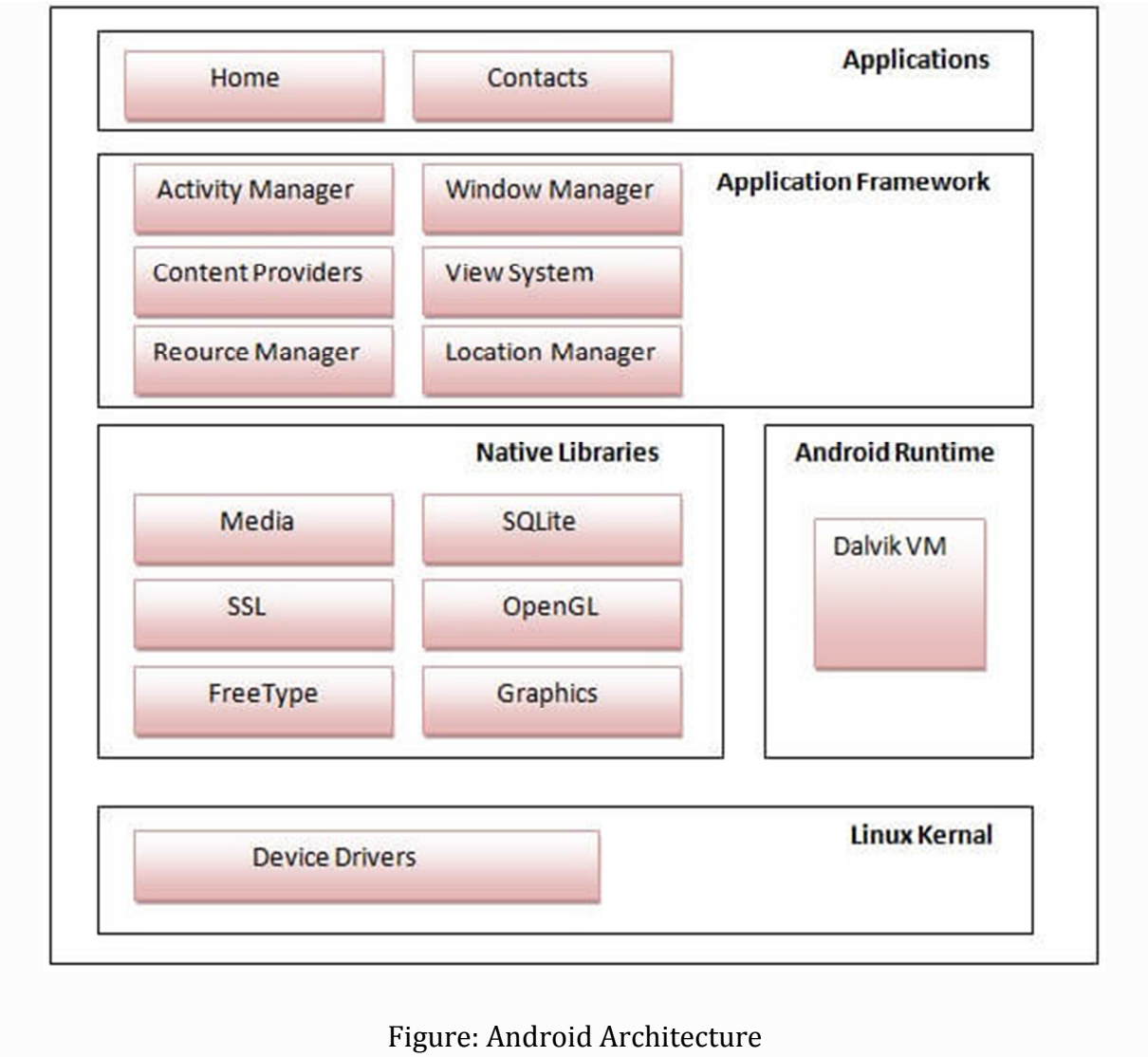


Figure: Android Architecture

1) 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.

2) Native Libraries

Running on the top of the kernel, the Android framework was developed with various features. It consists of various C/C++ core libraries with numerous of open source tools. Some of these are:

1. **The Android runtime:**
   The Android runtime consist of core libraries of Java and ART(the Android RunTime). Older versions of Android (4.x and earlier) had Dalvik runtime.

2. **Open GL(graphics library):**
   This cross-language, cross-platform application program interface (API) is used to produce 2D and 3D computer graphics.

3. **WebKit:**
   This open source web browser engine provides all the functionality to display web content and to simplify page loading.

4. **Media frameworks:**
   These libraries allow you to play and record audio and video.

5. **Secure Socket Layer (SSL):**
   These libraries are there for Internet security.

3) Android Runtime

In android runtime, there are core libraries and DVM (Dalvik Virtual Machine) 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.

4) Android Framework

On the top of Native libraries and android runtime, there is android framework. 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.

1. **Activity Manager:**
   It manages the activity lifecycle and the activity stack.

2. **Telephony Manager:**
   It provides access to telephony services as related subscriber information, such as phone numbers.

3. **View System:**
   It builds the user interface by handling the views and layouts.

4. **Location manager:**
   It finds the device's geographic location.

5) Applications

On the top of android framework, there are applications. All applications such as home, contact, settings, games, browsers are using android framework that uses android runtime and libraries. Android runtime and native libraries are using linux kernal.

Terminologies Related to Android

1. **XML**
   In Android, XML is used for designing the application's UI like creating layouts, views, buttons, text fields etc. and also used in parsing data feeds from the internet.

2. **View**
   A view is an UI which occupies rectangular area on the screen to draw and handle user events.

3. **Layout**
   Layout is the parent of view. It arranges all the views in a proper manner on the screen.

4. **Activity**
   An activity can be referred as your device's screen which you see. User can place UI elements in any order in the created window of user's choice.

5. **Emulator**
   An emulator is an Android virtual device through which you can select the target Android version or platform to run and test your developed application.

6. **Manifest file**
   Manifest file acts as a metadata for every application. This file contains all the essential information about the application like app icon, app name, launcher activity, and required permissions etc.

7. **Service**
   Service is an application component that can be used for long-running background processes. It is not bounded with any activity as there is no UI. Any other application component can start a service and this service will continue to run even when the user switches from one application to another.

8. **Broadcast Receiver**

Broadcast Receiver is another building block of Android application development which allows you to register for system and application events. It works in such a way that, when the event triggers for the first time all the registered receivers through this broadcast receiver will get notified for all the events by Android Runtime. To know more about the broadcast receivers, kindly refer Android Basic Building Blocks.

9. Content Providers

Content Providers are used to share data between two applications. This can be implemented in two ways:

1. When you want to implement the existing content provider in another application.
2. When you want to create a new content provider that can share its data with other applications

10. Intent

Intent is a messaging object which can be used to communicate between two or more components like activities, services, broadcast receiver etc. Intent can also be used to start an activity or service or to deliver a broadcast message.

Unit 2- Installation and configuration of Android

Course Outcome:

Configure Android environment and development tools.

Unit Outcomes:

2a. Describe function of the given component to operate the specified IDE.
2b. Explain the given term related to virtual machine.
2c. Explain the given basic term related to Android development tools.
2d. Describe the features of given android emulator.
2e. Describe the steps to configure the given android development environment

---

Contents:

2.1 Operating System, Java JDK, Android SDK
2.2 Android Development Tools (ADT)
2.3 Android Virtual Devices (AVDs)
2.4 Emulators
2.5 Dalvik Virtual Machine, Difference between JVM and DVM
2.6 Steps to install and configure Android Studio and SDK

2.1 Operating System, Java JDK, Android SDK

Operating System

- A mobile OS is an operating system for smartphones, tablets, PDAs, or other mobile devices.
- Mobile OSs combine features of a personal computer OS with other features useful for mobile or handheld use; usually including, and most of the following considered essential in modern mobile systems;

Touchscreen, cellular, Bluetooth, Wi-Fi, GPS mobile navigation, camera, video camera, speech recognition, voice recorder, music player, etc

Java JDK

The Java Development Kit (JDK) is a software development environment used for developing Java applications and applets. It includes the Java Runtime Environment (JRE), an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development.

JVM

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent). JVM is a part of Java Run Environment (JRE).

The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JRE

The Java Runtime Environment (JRE) is a set of software tools for development of Java applications. It combines the Java Virtual Machine (JVM), platform core classes and supporting libraries.

Android SDK

Android development starts with the Android SDK (Software Development Kit). It is a software development kit that enables developers to create applications for the Android platform.

The Android SDK (software development kit) is a set of development tools used to develop applications for Android platform. The Android SDK includes the following:

- Required libraries
- Debugger
- An emulator
- Relevant documentation for the Android application program interfaces (APIs)
- Sample source code
- Tutorials for the Android OS

2.2 Android Development Tools (ADT)

1. Android Studio

Developed by Google, Android Studio is an all-rounder integrated development environment that allows the Android developers to get what they desire without an Integrated Development Environment or IDE.

Android has Gradle-base support that has features like visual layout editor, intelligent code editor, real-time profilers and APK analyzer. It acts just like any other Java IDE in terms of error investigating and file hierarchy.

2. Visual Studio – Xamarin

Xamarin was launched in 2011 which is the best free IDE for delivering an enterprise-quality, cross-platform approach. **Xamarin** supplies add-ins to Microsoft **Visual Studio** that allows developers to build **Android**, iOS, and Windows apps within the IDE

3. IntelliJ IDEA

The framework based assistance, productivity boosters, unobtrusive intelligence, duplicates, and inspections are provided with the IDE. Using this IDE, you can do in-depth coding, quick navigation, and error analysis. It supports mobile app development with the help of Java, Scala, Kotlin, Groovy.

4. Eclipse IDE

It is one of the most popular IDES of Android apps. The open-source software is free to use. Released under the Eclipse Public License, it holds a large community having plenty of plugins and configurations. Highly customizable offers full support for Java programming language and XML.

Android Development IDEs	Languages	Target OS	Runs On	Audience	License	Price
Android Studio	Java C C++ Kotlin	Android	Windows MacOS Linux	Experienced	Freeware	Free
Eclipse	Java C C++ C# JavaScript Python more	Android iOS Linux MacOS Windows	Any OS supporting Java	Professional Developers	Eclipse Public License	Free
Visual Studio (with Xamarin)	C++ C C# Visual Basic PHP JavaScript more	Cross-Platform Windows Android iOS more	Windows MacOS Linux	Experienced	Proprietary, Visual Studio Code is Open Source MIT	Free to $2,999+
IntelligJ	Java	Any OS	Windows	Professional	Proprieta	Free to

IDEA	Scala Groovy Kotlin JavaScript TypeScript SQL	supporting Java	MacOS Linux	Java Developers	ry, Community Edition is Apache 2.0 License	$499/year
NetBeans	Java C C++ HTML PHP JavaScript others	Cross-platform	Windows MacOS Linux Solaris	Professional Developers	CDDL 1.0 and GPL2	Free
Komodo	Java JavaScript Python PHP HTML Ruby others	Cross-platform	Windows MacOS Linux	Professional Web and mobile developers	Proprietary, Komodo Edit is Mozilla Public License	Free to $394+
Cordova	HTML CSS JavaScript	Cross-platform Android Windows iOS MacOS Ubuntu	Windows MacOS Linux	Experienced Web developers	Apache 2.0 License	Free
PhoneGap	HTML CSS JavaScript	Cross-platform Android iOS	Windows MacOS Linux Android Windows Phone	Web developers	Apache 2.0 License	Free
App Inventor	Kawa	Android	Windows MacOS Linux	Students and amateurs	MIT License	Free
AIDE	Java C C++ XML HTML CSS JavaScript	Android Web	Android	Amateurs or mobile professionals	Proprietary	Free with in-app purchases

2.3 Android Virtual Devices (AVDs)

An Android Virtual Device (AVD) is a configuration that defines the characteristics of an Android phone, tablet, Wear OS, Android TV, or Automotive OS device that you want to

simulate in the Android Emulator. The AVD Manager is an interface you can launch from Android Studio that helps you create and manage AVDs.

To open the AVD Manager, do one of the following:

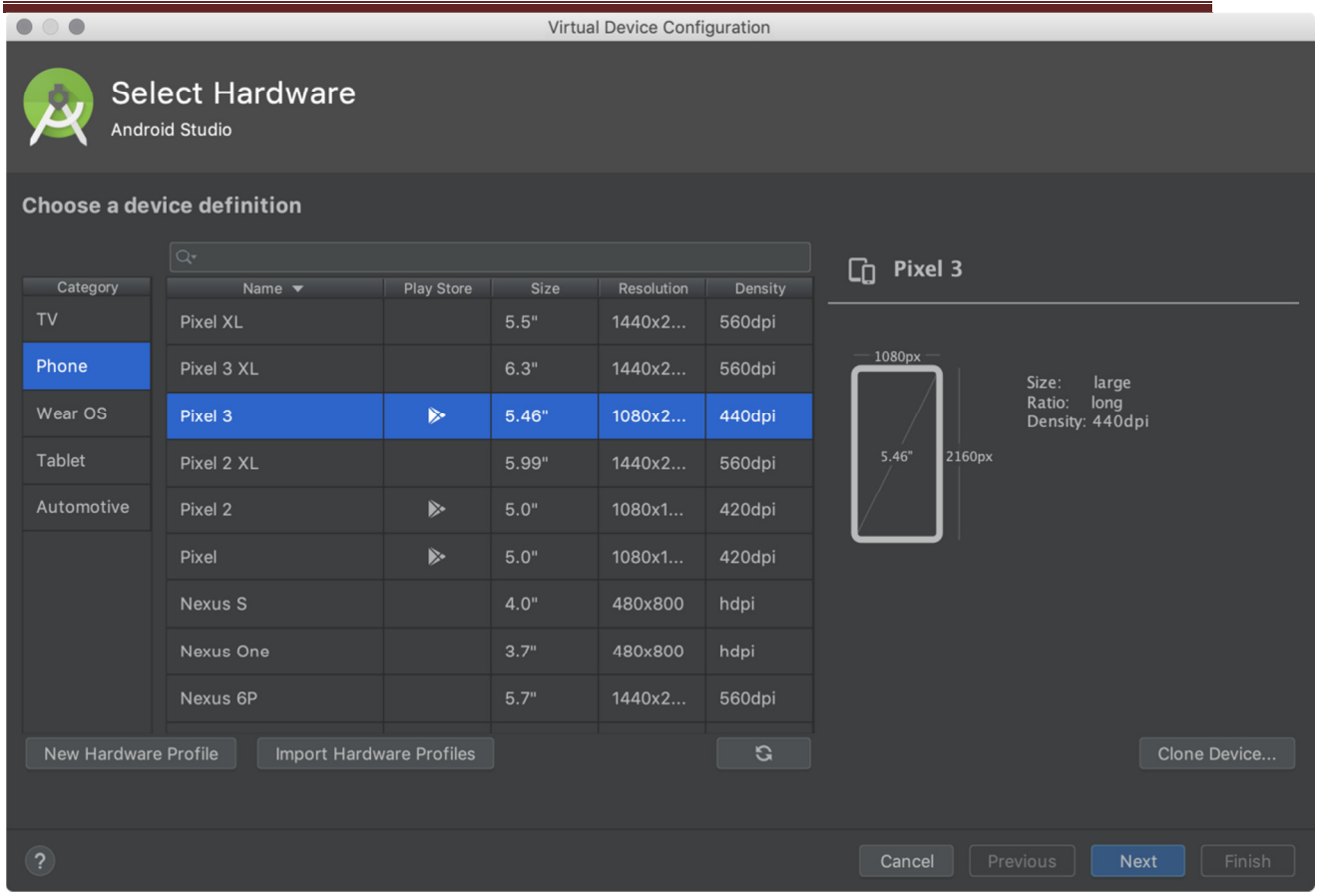
- Select **Tools > AVD Manager**.
- Click **AVD Manager**  in the toolbar.

To create a new AVD:

1. Open the AVD Manager by clicking **Tools > AVD Manager**.

2. Click **Create Virtual Device**, at the bottom of the AVD Manager dialog.

   The **Select Hardware** page appears.

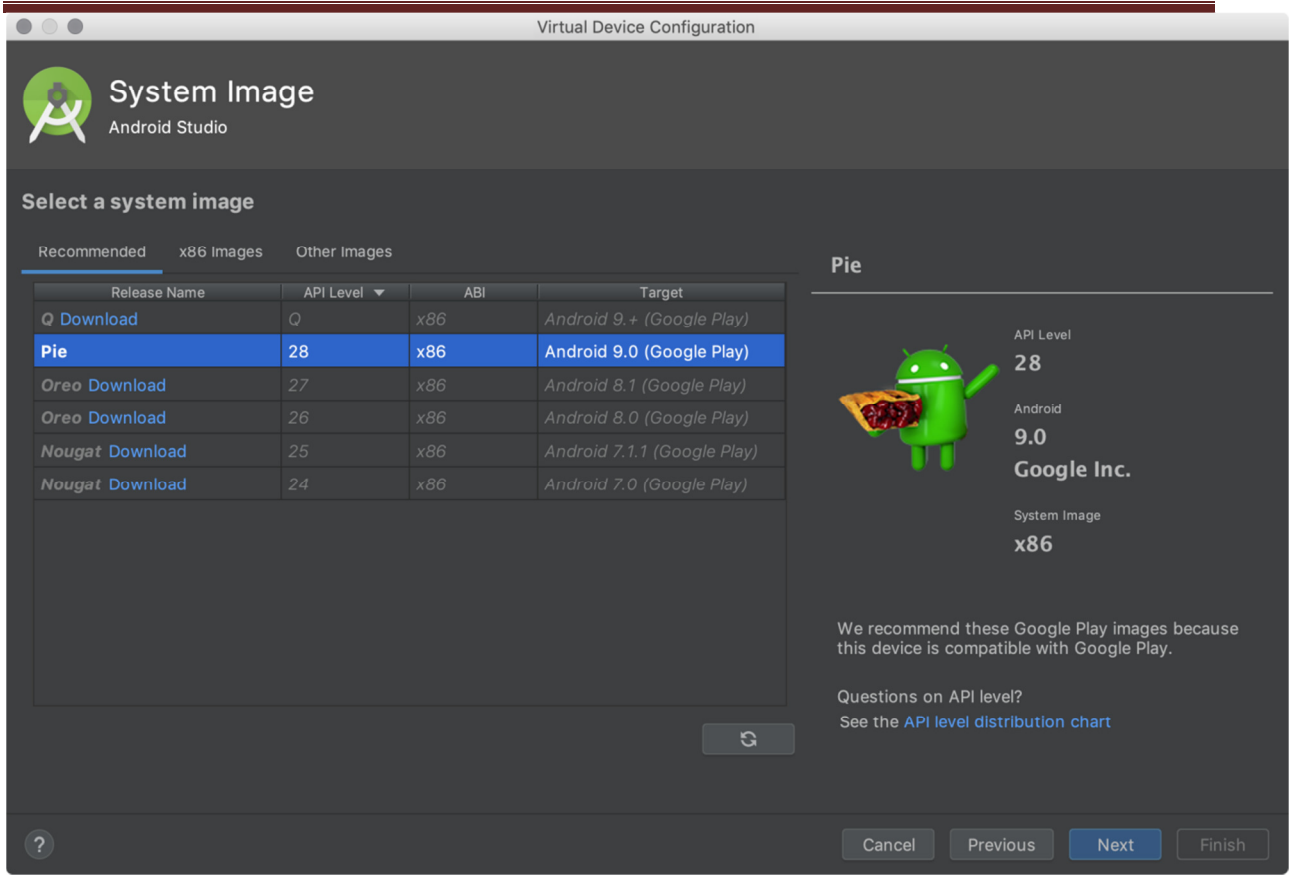


Notice that only some hardware profiles are indicated to include **Play Store**. This indicates that these profiles are fully CTS compliant and may use system images that include the Play Store app.

3. Select a hardware profile, and then click **Next**.

   If you don't see the hardware profile you want, you can create or import a hardware profile.

   The **System Image** page appears.



4. Select the system image for a particular API level, and then click **Next**.

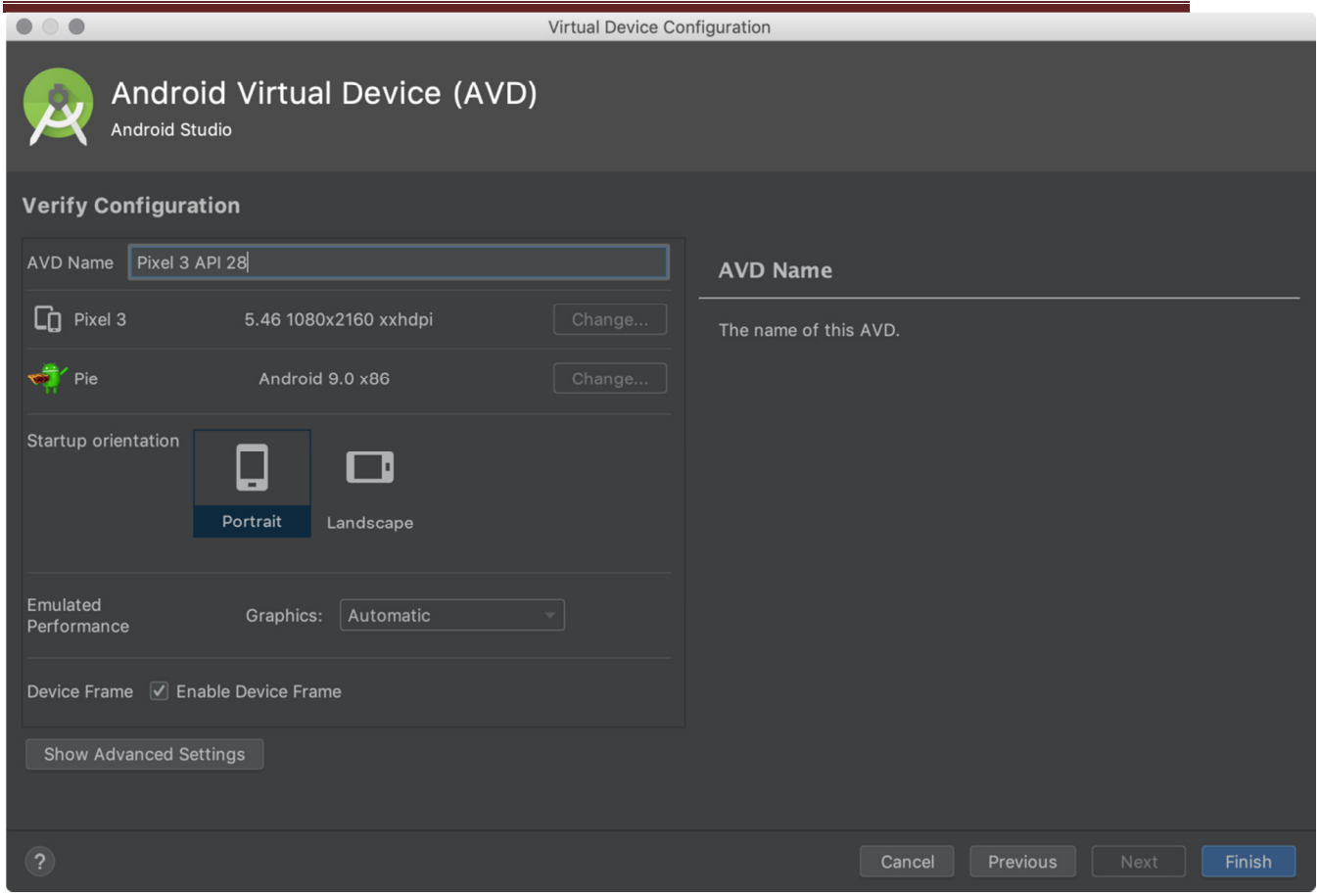
   The **Recommended** tab lists recommended system images. The other tabs include a more complete list. The right pane describes the selected system image. x86 images run the fastest in the emulator.

   If you see **Download** next to the system image, you need to click it to download the system image. You must be connected to the internet to download it.

   The API level of the target device is important, because your app won't be able to run on a system image with an API level that's less than that required by your app, as specified in the `minSdkVersion` attribute of the app manifest file. For more information about the relationship between system API level and `minSdkVersion`, see Versioning Your Apps.

   If your app declares a `<uses-library>` element in the manifest file, the app requires a system image in which that external library is present. If you want to run your app on an emulator, create an AVD that includes the required library. To do so, you might need to use an add-on component for the AVD platform; for example, the Google APIs add-on contains the Google Maps library.

   The **Verify Configuration** page appears.



Change AVD properties as needed, and then click **Finish**.

Now you get a new AVD ready for launching your apps on it.

2.4 Emulators

The Android Emulator simulates Android devices on your computer so that you can test your application on a variety of devices and Android API levels without needing to have each physical device.

The emulator provides almost all of the capabilities of a real Android device. You can simulate incoming phone calls and text messages, specify the location of the device, simulate different network speeds, simulate rotation and other hardware sensors, access the Google Play Store, and much more.

Testing your app on the emulator is in some ways faster and easier than doing so on a physical device. For example, you can transfer data faster to the emulator than to a device connected over USB.

The emulator comes with predefined configurations for various Android phone, tablet, Wear OS, and Android TV devices.

In short, An **Android emulator** is an **Android** Virtual Device (AVD) that represents a specific **Android** device. You can use an **Android emulator** as a target platform to run and test your **Android** applications on your PC. Using **Android emulators** is optional.

Launch the Android Emulator without first running an app

To start the emulator:

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

The Android Emulator loads.

While the emulator is running, you can run Android Studio projects and choose the emulator as the target device. You can also drag one or more APKs onto the emulator to install them, and then run them.

2.5 Dalvik Virtual Machine (DVM)

As we know the modern JVM is high performance and provides excellent memory management. But it needs to be optimized for low-powered handheld devices as well.

The **Dalvik Virtual Machine (DVM)** is an android virtual machine optimized for mobile devices. It optimizes the virtual machine for *memory*, *battery life* and *performance*.

Dalvik is a name of a town in Iceland. The Dalvik VM was written by Dan Bornstein.

The Dex compiler converts the class files into the .dex file that run on the Dalvik VM. Multiple class files are converted into one dex file.

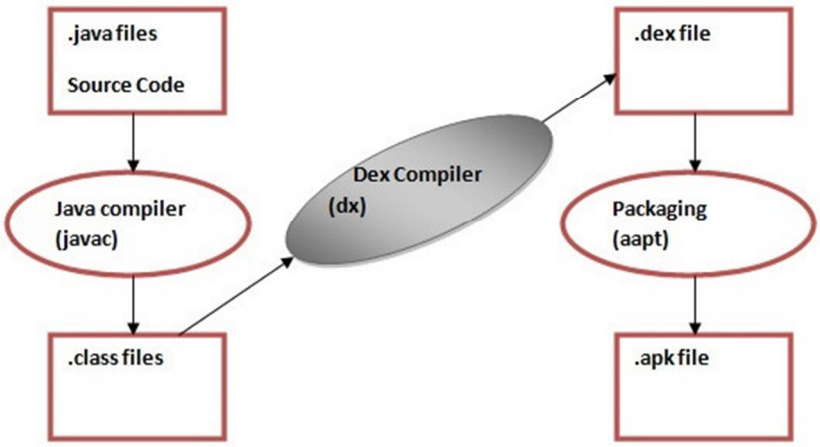


Fig.: The compiling and packaging process from the source file

The **javac tool** compiles the java source file into the class file.

The **dx tool** takes all the class files of your application and generates a single .dex file. It is a platform-specific tool.

The **Android Assets Packaging Tool (aapt)** handles the packaging process.

Difference between JVM and DVM

DVM (Dalvik Virtual Machine)	JVM (Java Virtual Machine)
It is Register based which is designed to run on low memory.	It is Stack based.
DVM uses its own byte code and runs ".Dex" file. From Android 2.2 SDK Dalvik has got a Just in Time compiler	JVM uses java byte code and runs ".class" file having JIT (Just In Time).
DVM has been designed so that a device can run multiple instances of the VM efficiently. Applications are given their own instance.	Single instance of JVM is shared with multiple applications.
DVM supports Android operating system only.	JVM supports multiple operating systems.
For DVM very few Re-tools are available.	For JVM many Re-tools are available.
There is constant pool for every application.	It has constant pool for every class.
Here the executable is APK.	Here the executable is JAR.

2.6 Steps to install and configure Android Studio and SDK

Installation
Follow steps below for complete installation and configuration of Android Studio.

Step 1) Download Android Studio
You can download Android Studio from this link or go to developer.android.com homepage and search for downloads. Choose appropriate platform either for windows, mac or linux. Following are the pre requirements for windows operating system.

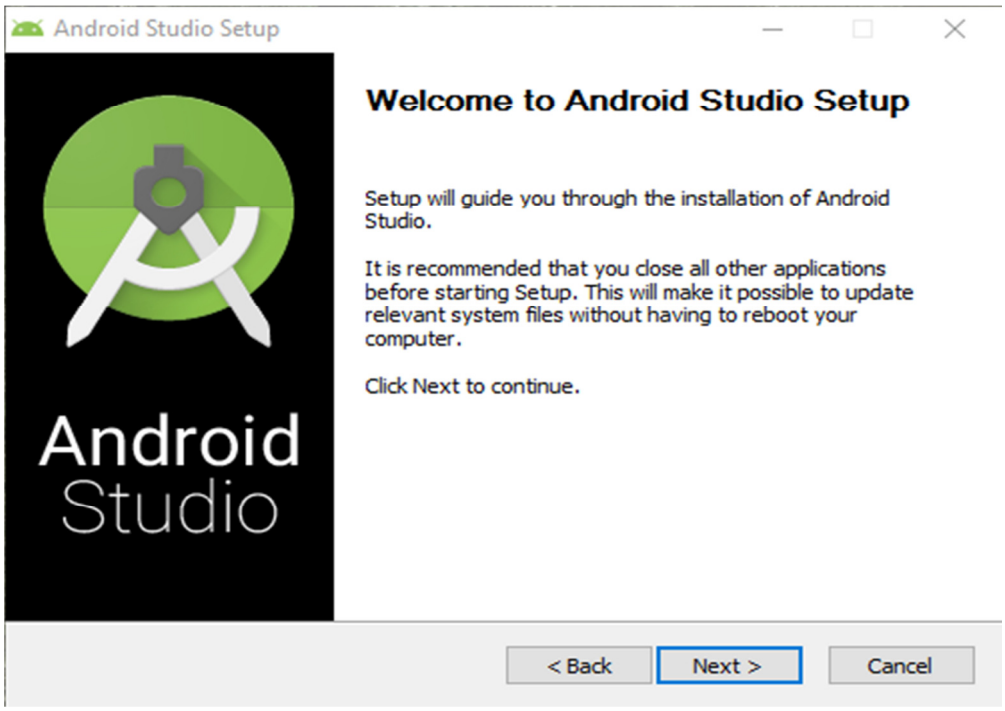
Pre-requirements

- Microsoft windows 7/8/10 (32 or 64 bits)
- Minimum 3GB RAM (recommended 8GB)
- 2GB disk space
- 1280 x 800 minimum screen resolution size
- Intel processor for accelerated emulator
- Android SDK

Note: If you don't have Android SDK, you can download with Android studio. Go to the end of download's page and find android-studio-bundle-162.4069837-windows.exe it includes SDK also.

Step 2) Run .exe file

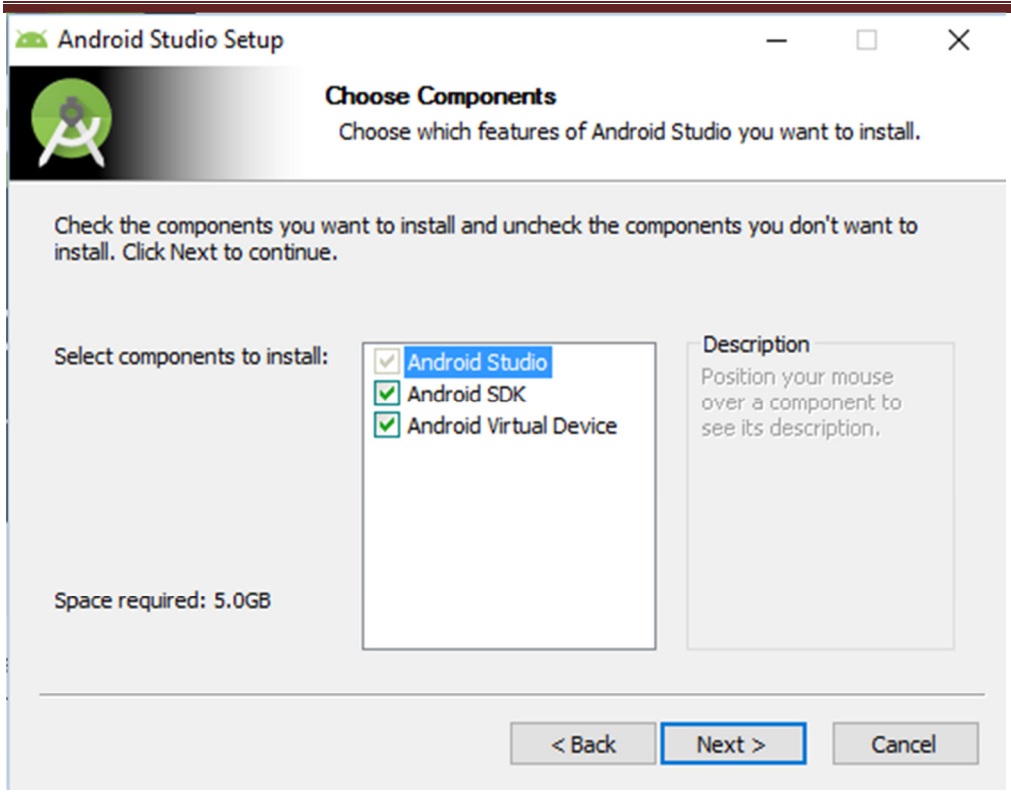
Now the next step is to launch .exe file you just download. Following screen will appear



Step 1: Run .exe file

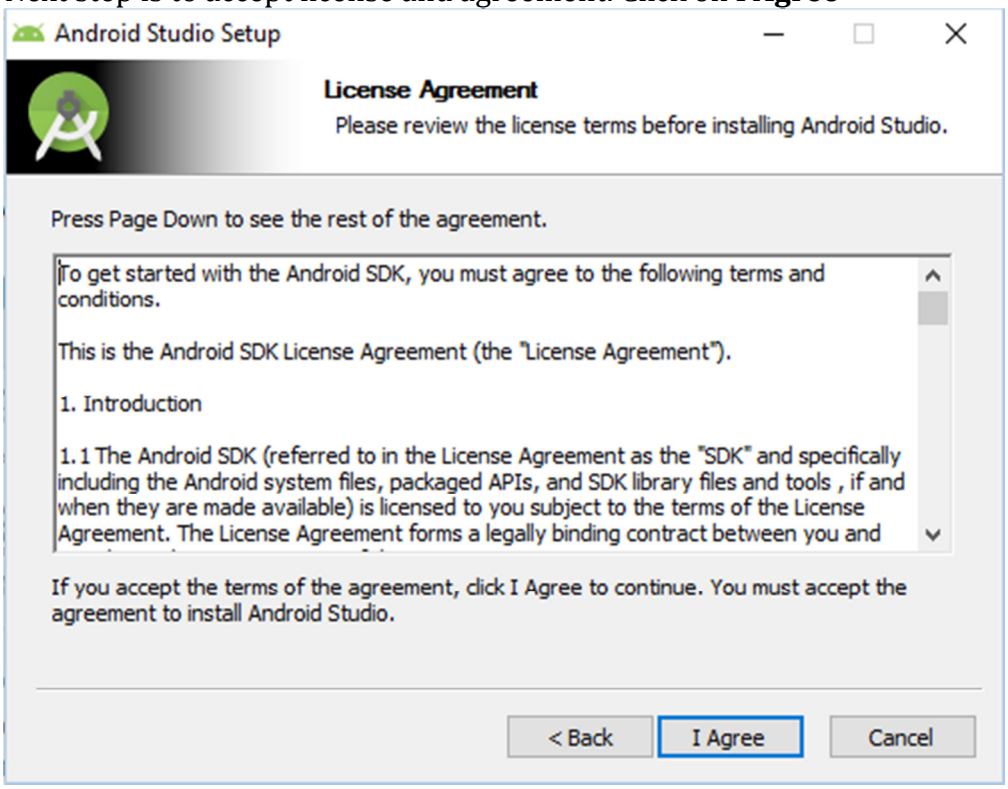
Click next and select Android SDK checked if you don't have it already. Better is to leave the default settings.

Make sure Android virtual device is also checked.



Step 2: Choose components

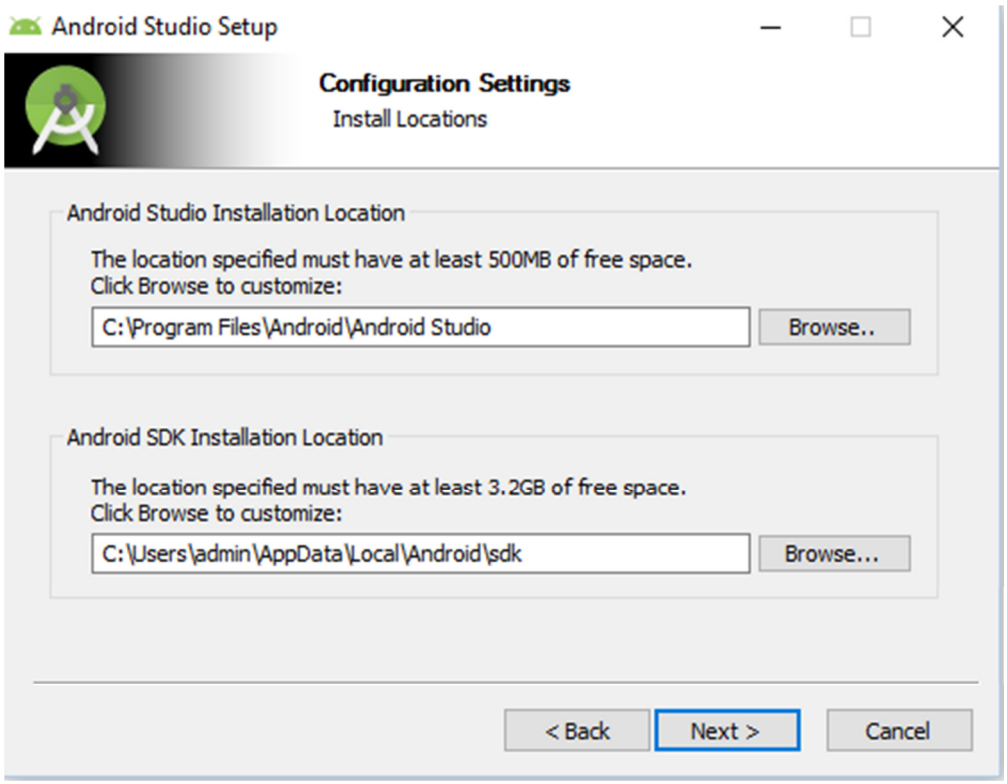
Next step is to accept license and agreement. Click on **I Agree**



Step 3: Accept license

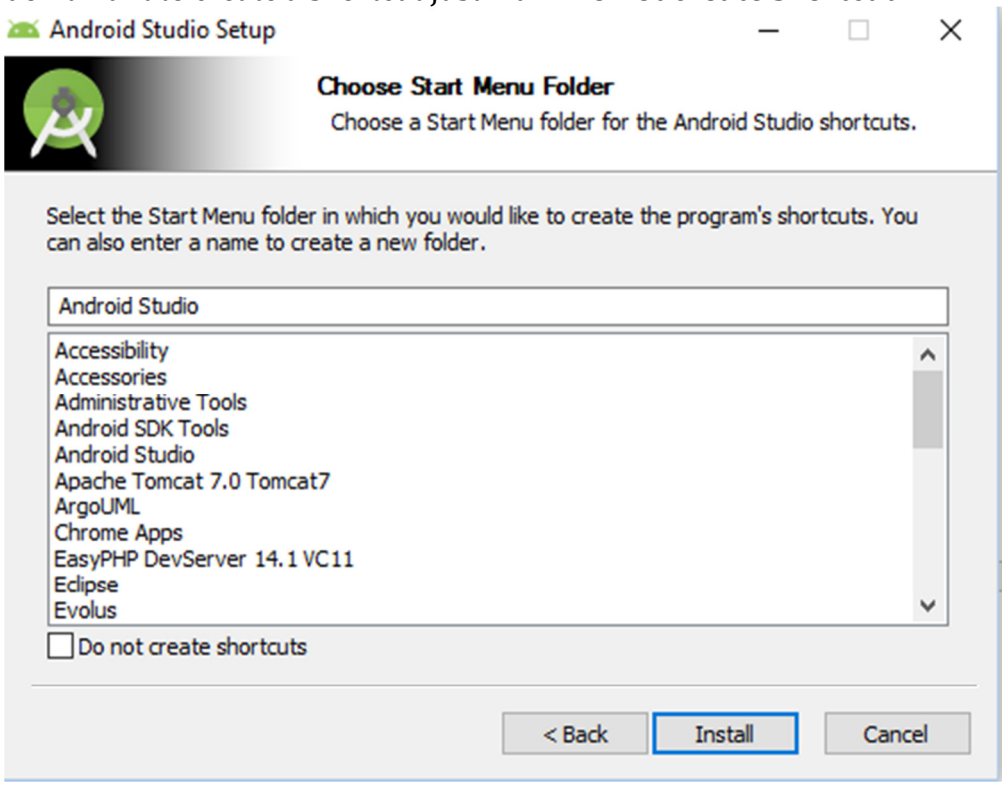
Next step is to set location of installation. Please make sure your disk has minimum required space before clicking on Next. For Android Studio installation location must

have at least 500MB free space. For Android SDK installation, selected location must have at least 3.25GB free space.



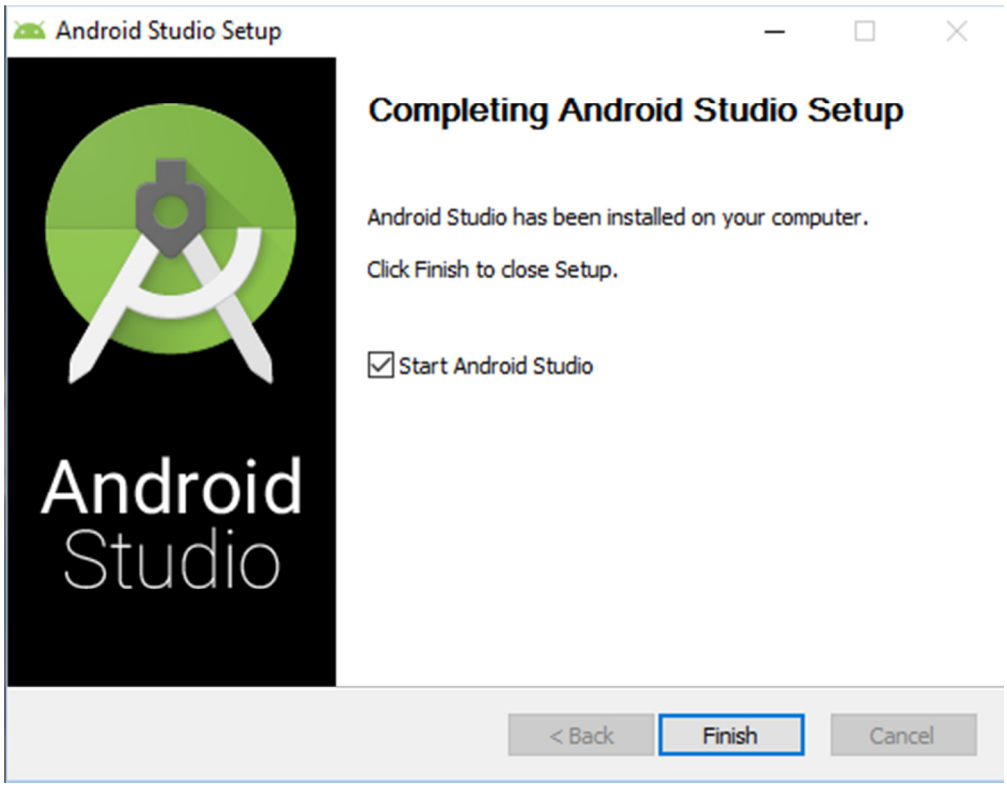
Step 4: Install location

Next step is to choose the start menu folder, where you want to create shortcut. If you don't want to create a shortcut just mark **Do not create shortcut**.



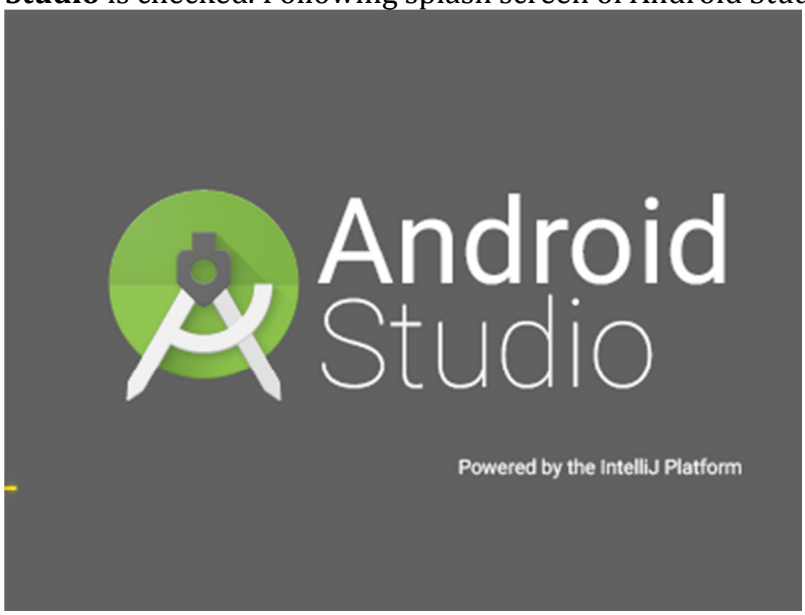
Step 5: Choose start menu folder

And hit **Install** button.
It will start installation. Once it's done following window will appear.



Step 6: Finish

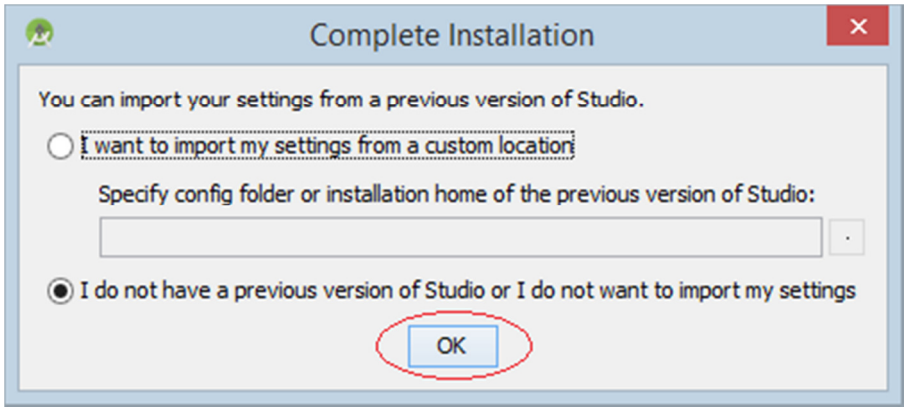
This informs you installation has completed. Click **Finish**. Make sure **Start Android Studio** is checked. Following splash screen of Android Studio will appear.



Step 7: Android Studio Splash Screen

Step 3) Configure Android Studio

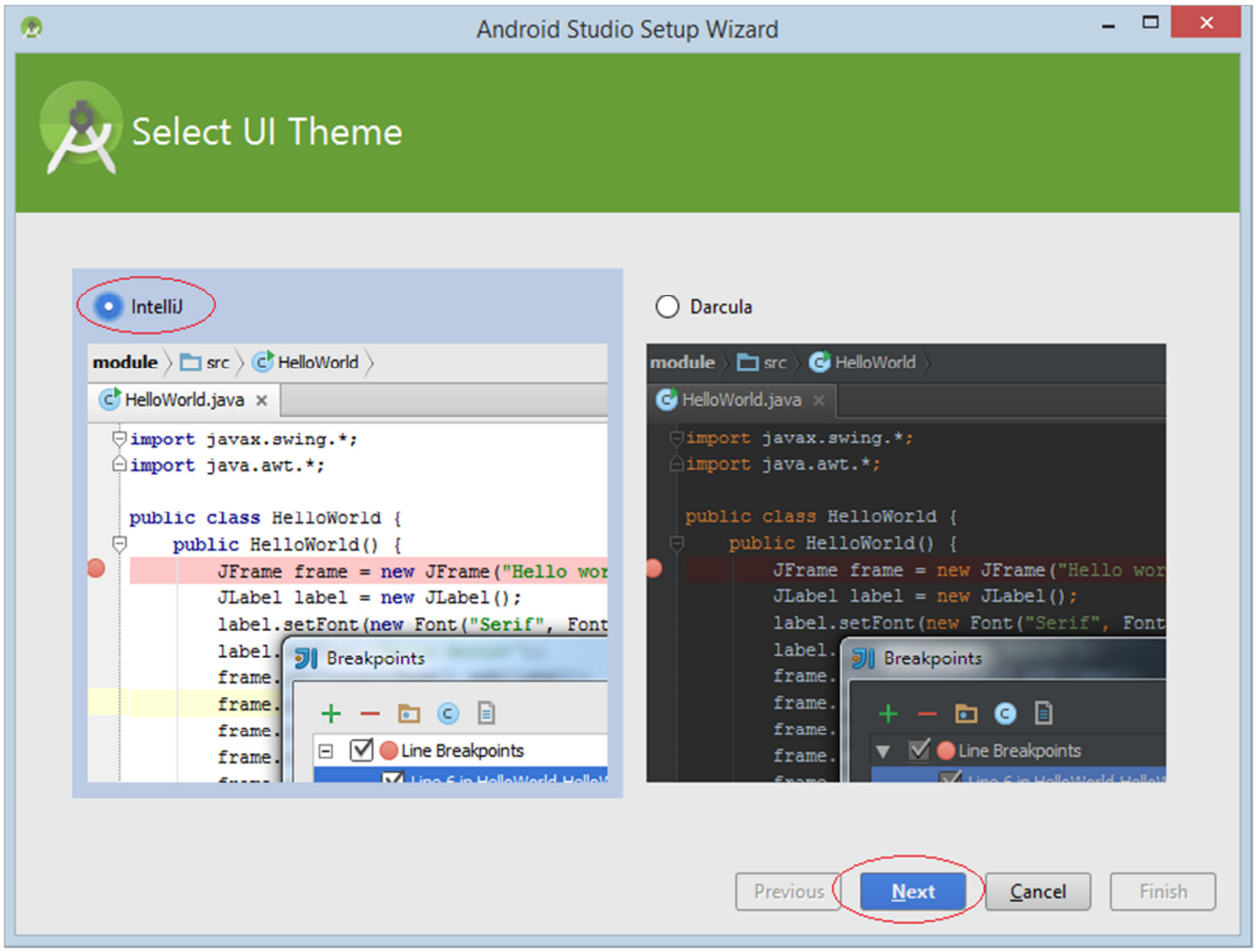
When you run it for the first time it will ask for Android Studio settings.



Step 8: Import settings

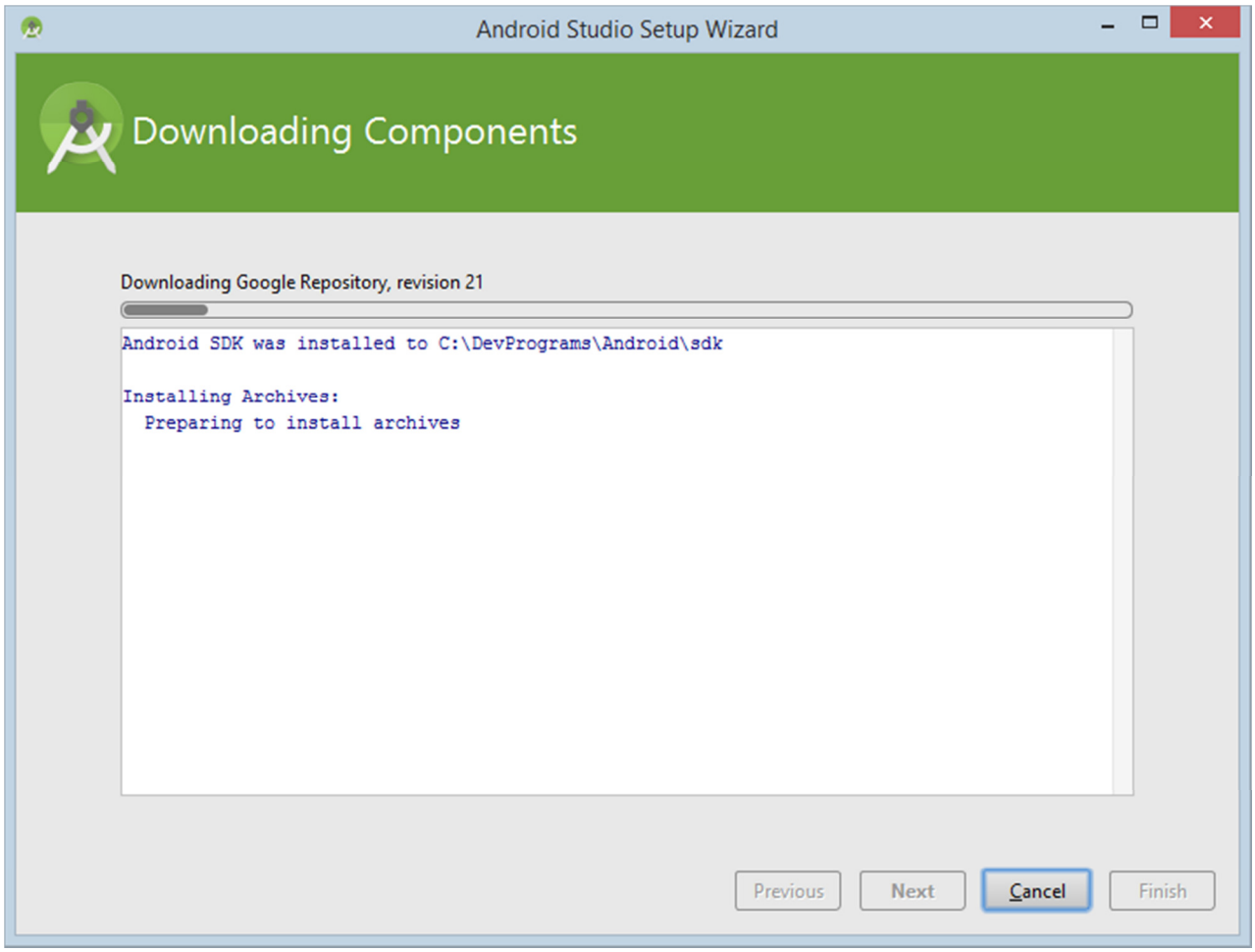
If you don't have any previous settings click on the second option (I don't have a previous version of Studio or I don't want to import my settings).

Select a theme and click next.



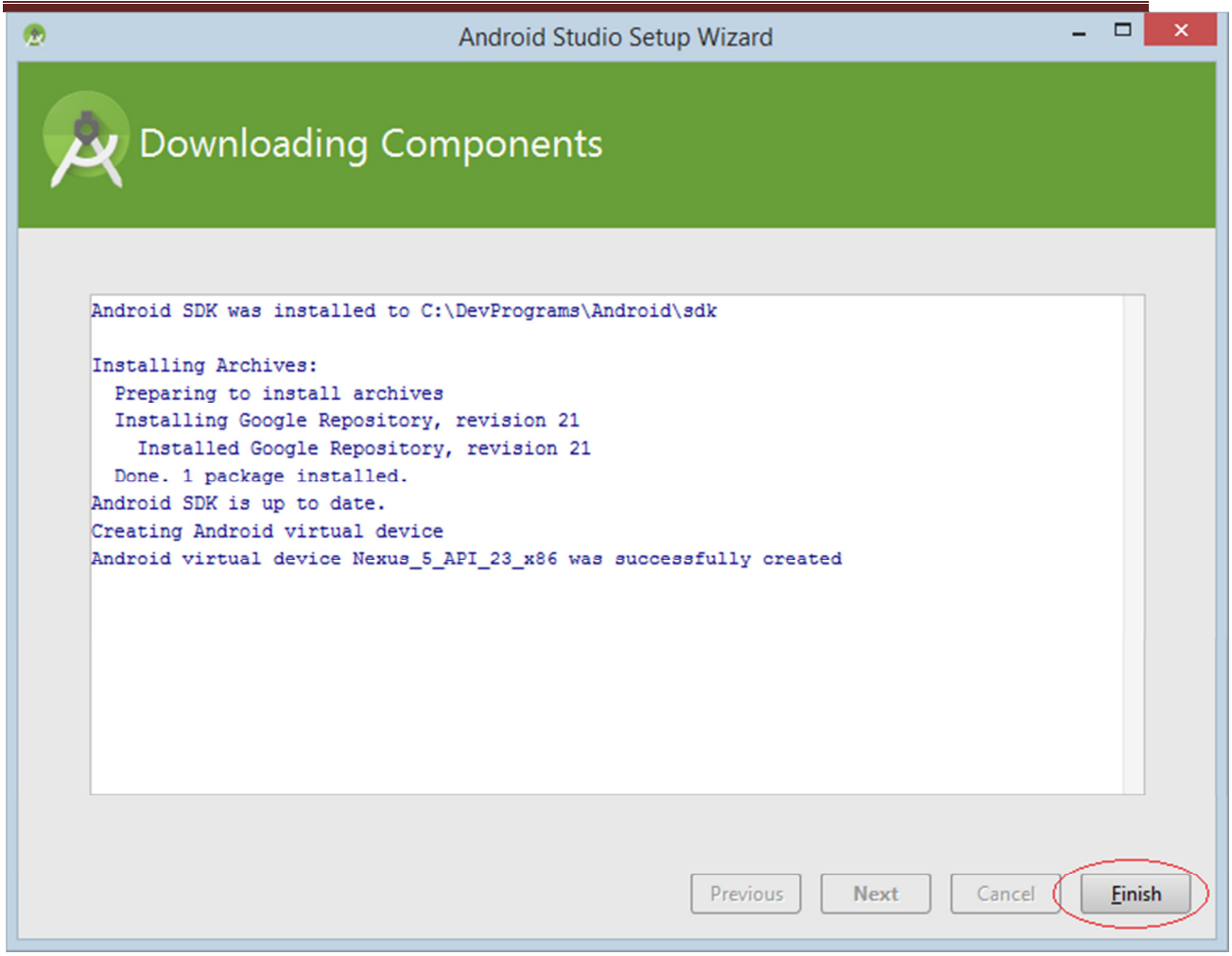
Step 9: Select theme

At the very first run it needs to download some necessary components, wait till it completes.



Step 10: Download components

And it's all done.

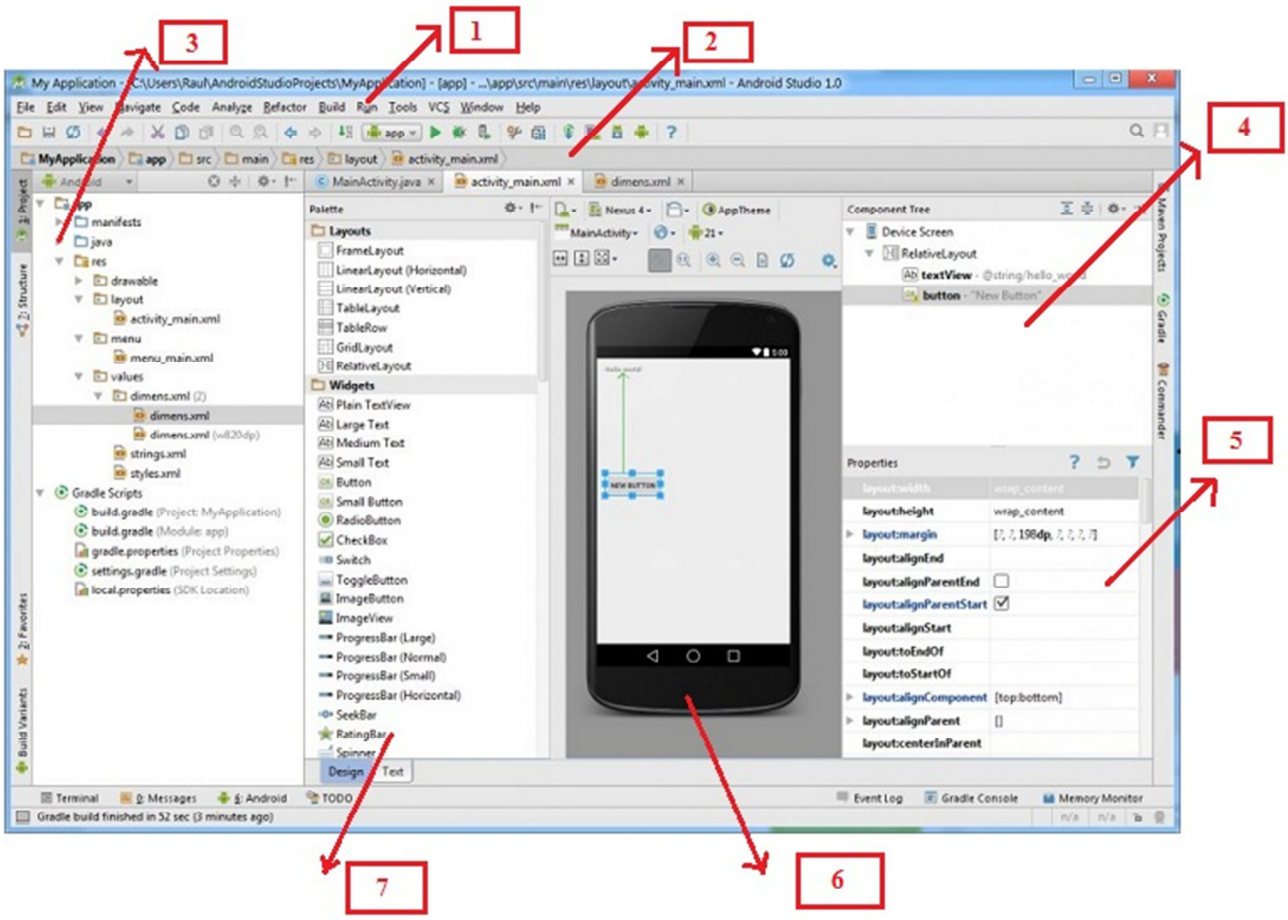


Step 11: Done

Click on **Finish** and start building your Android apps.

Introduction to Android Studio User Interface

Android Studio is an Integrated Development Environment (IDE). You have seen download and installation in this tutorial. Let's learn some basics of Android Studio. Here is the a screenshot of a running Android studio.



Android Studio screen

Red mark shows

1: Tool bar- It is collection of many tools like cut, copy, paste, run debug and others.
2: Navigation bar- It helps you to navigate through the recent open files of your project.
3: Project hierarchy- It is the hierarchy of your project's folders.
4: Component Tree- It shows component used in an activity in the form of a tree structure.
5: Properties window- It shows properties of selected item on the screen.
6: Layout editor- It shows graphical layout, how your app will look like.
7: Palette window- Palette window shows component, layouts, and widgets available in Android Studio.

Create an Android project

This lesson shows you how to create a new Android project with Android Studio, and it describes some of the files in the project.

To create your new Android project, follow these steps:

1. Install the latest version of Android Studio.

2. In the **Welcome to Android Studio** window, click **Start a new Android Studio project**.

Figure 1. Android Studio welcome screen

If you have a project already opened, select **File > New > New Project**.

3. In the **Choose your project** window, select **Empty Activity** and click **Next**.
4. In the **Configure your project** window, complete the following:

- Enter "My First App" in the **Name** field.
- Enter "com.example.myfirstapp" in the **Package name** field.
- If you'd like to place the project in a different folder, change its **Save** location.
- Select either **Java** or **Kotlin** from the **Language** drop-down menu.
- Select the checkbox next to **Use androidx.* artifacts**.
- Leave the other options as they are.

5. Click **Finish**.

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

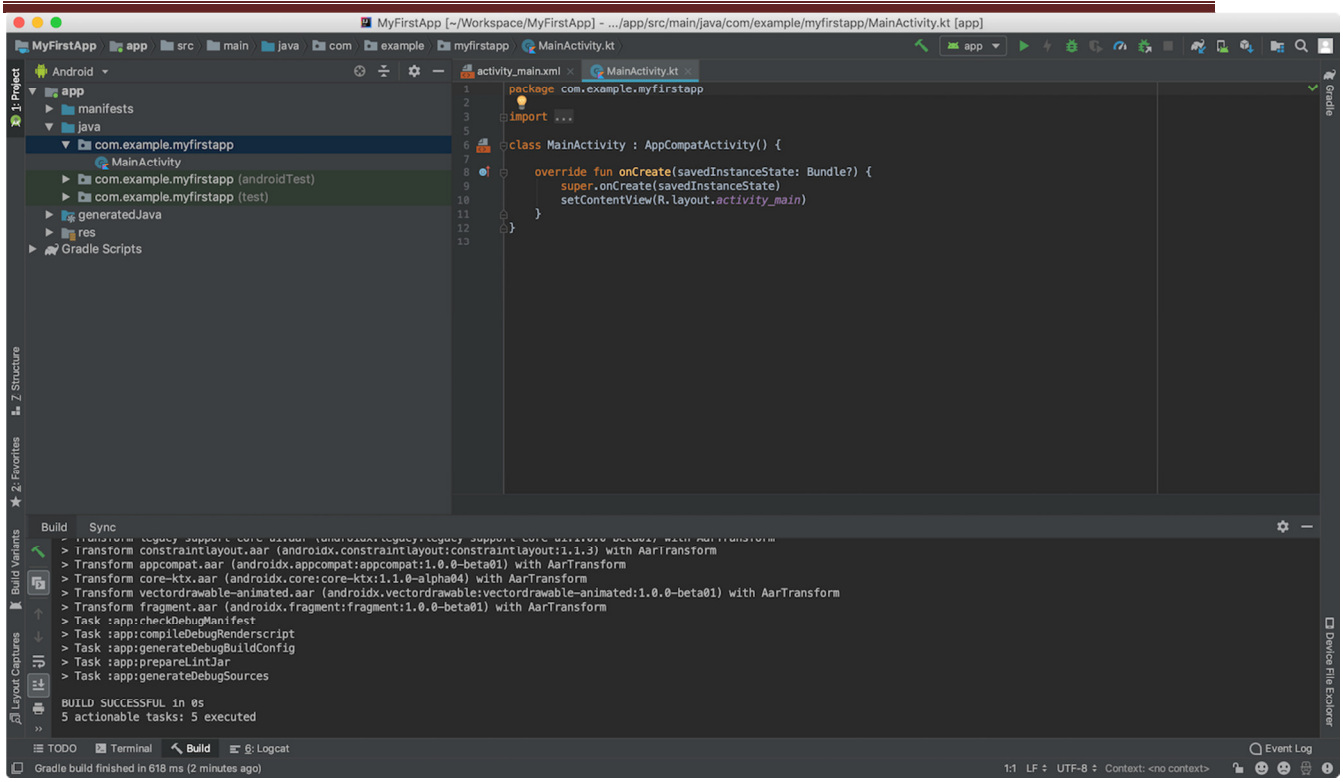


Figure 2. Android Studio main window

Now take a moment to review the most important files.

First, be sure the **Project** window is open (select **View > Tool Windows > Project**) and the Android view is selected from the drop-down list at the top of that window. You can then see the following files:

app > java > com.example.myfirstapp > MainActivity

This is the main activity. It's the entry point for your app. When you build and run your app, the system launches an instance of this Activity and loads its layout.

app > res > layout > activity_main.xml

This XML file defines the layout for the activity's user interface (UI). It contains a TextView element with the text "Hello, World!"

app > manifests > AndroidManifest.xml

The manifest file describes the fundamental characteristics of the app and defines each of its components.

Gradle Scripts > build.gradle

There are two files with this name: one for the project, "Project: My First App," and one for the app module, "Module: app." Each module has its own build.gradle file, but this project currently has just one module. Use each

module's build.file to control how the Gradle plugin builds your app. For more information about this file, see Configure your build.

Run your app

In the previous section, you created an Android app that displays "Hello, World!" You can now run the app on a real device or an emulator.

Run on a real device

Set up your device as follows:

1. Connect your device to your development machine with a USB cable. If you developed on Windows, you might need to install the appropriate USB driver for your device.
2. Perform the following steps to enable **USB debugging** in the **Developer options** window:

a. Open the **Settings** app.

b. If your device uses Android v8.0 or higher, select **System**. Otherwise, proceed to the next step.

c. Scroll to the bottom and select **About phone**.

d. Scroll to the bottom and tap **Build number** seven times.

e. Return to the previous screen, scroll to the bottom, and tap **Developer options**.

f. In the **Developer options** window, scroll down to find and enable **USB debugging**.

Run the app on your device as follows:

1. In Android Studio, select your app from the run/debug configurations drop-down menu in the toolbar.
2. In the toolbar, select the device that you want to run your app on from the target device drop-down menu.

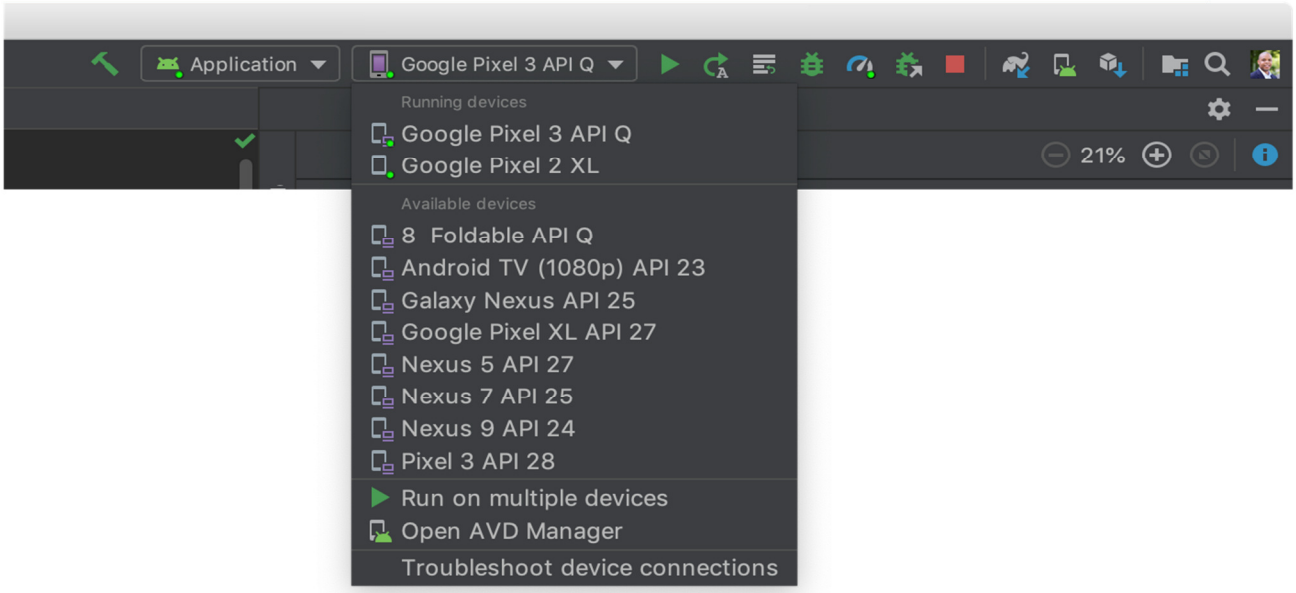


Figure 1. Target device drop-down menu

1. Click **Run**  .

   Android Studio installs your app on your connected device and starts it. You now see "Hello, World!" displayed in the app on your device.

   To begin to develop your app, continue to the next lesson.

Run on an emulator

Run the app on an emulator as follows:

1. In Android Studio, create an Android Virtual Device (AVD) that the emulator can use to install and run your app.
2. In the toolbar, select your app from the run/debug configurations drop-down menu.
3. From the target device drop-down menu, select the AVD that you want to run your app on.

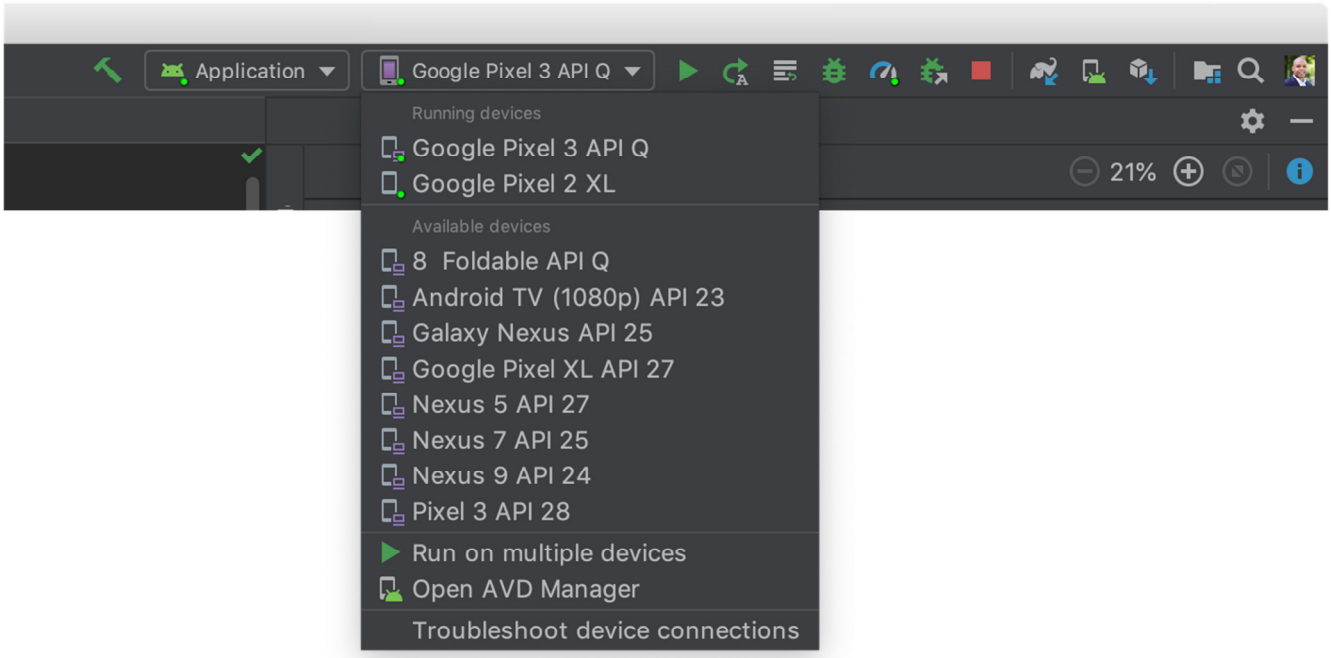


Figure 2. Target device drop-down menu

4. Click **Run**  .

   Android Studio installs the app on the AVD and starts the emulator. You now see "Hello, World!" displayed in the app.

Unit-IV Designing User Interface

Course Outcome:

Use User Interface components for android application development..

Unit Outcomes:

4a. Develop rich user Interfaces for the given Android application.
4b. Develop Android application using the given view.
4c. Explain the significance of the given display Alert.
4d. Develop the given application using time and date picker.

Contents:

4.1 Text View, Edit Text; Button, Image Button; Toggle Button, Radio Button And Radio Group; Checkbox; Progress Bar
4.2 List View; Grid View; Image View; Scroll View; Custom Toast Alert
4.3 Time And Date Picker

Fundamentals of UI Design

With View

The "views" are the building blocks of a U.I design and composes of almost every basic U.I element like TextViews, EditTexts, ImageViews etc. This '***view***' however comes along with a few properties bundled to them. Some of the important and are often used to build up a complete meaningful screen design.

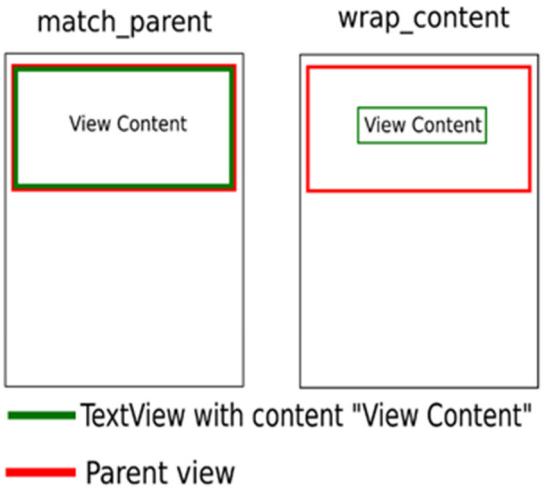
1. ***"id"***
2. ***"width"***
3. ***"height"***
4. ***"margin"***
5. ***"padding"***

"id"

This is basically the name of the particular view and will be used to refer that particular view through out the project. It has to be unique(*multiple views referencing to same id will confuse the compiler*). Common ethic to name an id of a view is **"descriptionOfView_viewType" (**For ex. if there is a textview that denotes an email, its id will be **""email_textview")**

"width" and "height"

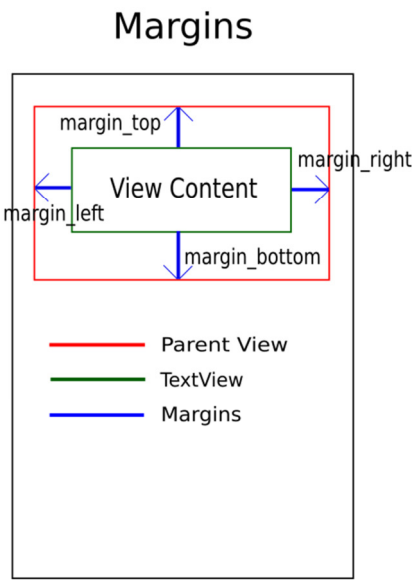
As the name of these properties suggest, these are the dimensions of that particular view. Now these dimensions can be set using hard-coded values and it will adopt to them in most layouts, but its not a very good design as the content inside them might get cropped or will have unwanted spaces. Android provides two pre-defined options to set these dimensions — ***"match_parent"*** and ***"wrap_content".***



Setting the dimensions to "**match_parent**" will make them equal to those of its parent's dimensions. If there is no parent to that particular view, it merely sets to the screen's dimensions (parent of a view is the U.I element in which it is housed in). And setting it to "**wrap_content**" will force the view to adopt its dimensions according to its content.

"margin"

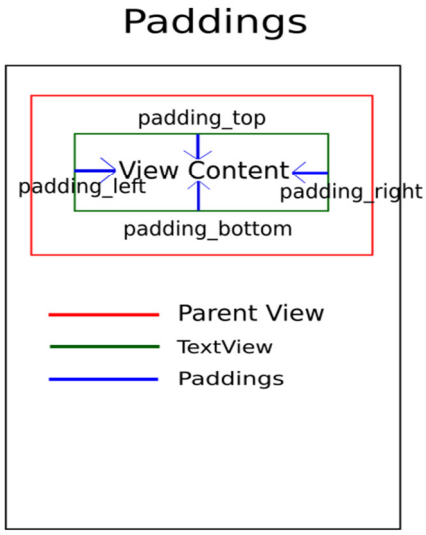
This is the minimum distance that a view has to maintain from its neighbouring views. That's it. Since there are four sides to any view, the four margins corresponding to them are **"margin_left", "margin_right", "margin_top"** and **"margin_bottom"**. If the same margin is needed to be set on all sides, it can be set directly through **"margin"** property.



Margins

"padding"

The distance between the view's outline and its content. Again similar to the "margin" property, "padding" too has **"padding_left", "padding_right", "padding_top", "padding_bottom"** and the common padding to all sides can be set through **"padding"** property.



Padding

TextView In Android Studio

In Android, TextView displays text to the user and optionally allows them to edit it programmatically. TextView is a complete text editor, however basic class is configured to not allow editing but we can edit it.

View is the parent class of TextView. Being a subclass of view the text view component can be used in your app's GUI inside a ViewGroup, or as the content view of an activity. We can create a TextView instance by declaring it inside a layout(XML file) or by instantiating it programmatically(Java Class).

TextView code in XML:

```
<TextView android:id="@+id/simpleTextView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="AbhiAndroid" />
```

TextView code in JAVA:

```
TextView textView = (TextView) findViewById(R.id.textView);
textView.setText("AbhiAndroid"); //set text for text view
```

Attributes of TextView:
Now let's we discuss about the attributes that helps us to configure a TextView in your xml file.

1**. id:** id is an attribute used to uniquely identify a text view. Below is the example code in which we set the id of a text view.

```
<TextView
android:id="@+id/simpleTextView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
```

2. **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.

Below is the example code with explanation included in which we set the center_horizontal gravity for text of a TextView.

```
<TextView
android:id="@+id/simpleTextView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="AbhiAndroid"
android:textSize="20sp"
android:gravity="center_horizontal"/> <!--center horizontal gravity-->
```

3. **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.
Below is the example code with explanation included in which we set the text "AbhiAndroid" in a text view.

```
<TextView
android:id="@+id/simpleTextView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:textSize="25sp"
android:text="AbhiAndroid"/><!--Display Text as AbhiAndroid-->
```

In Java class:
Below is the example code in which we set the text in a textview programmatically means in java class.

```
TextView textView = (TextView)findViewById(R.id.textView);
textView.setText("AbhiAndroid"); //set text for text view
```

4. **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".

Below is the example code with explanation included in which we set the red color for the displayed text.

```
<TextView
android:id="@+id/simpleTextView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="AbhiAndroid"
android:layout_centerInParent="true"
android:textSize="25sp"
android:textColor="#f00"/><!--red color for text view-->
```

In Java class:
Below is the example code in which we set the text color of a text view programmatically means in java class.

```
TextView textView = (TextView)findViewById(R.id.textView);
textView.setTextColor(Color.RED); //set red color for text view
```

5. **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).

Below is the example code in which we set the 20sp size for the text of a text view.

```
<TextView
   android:id="@+id/simpleTextView"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="AbhiAndroid"
   android:layout_centerInParent="true"
   android:textSize="40sp" /><!--Set size-->
```

In Java class:
Below is the example code in which we set the text size of a text view programmatically means in java class.

```
TextView textView = (TextView)findViewById(R.id.textView);
textView.setTextSize(20); //set 20sp size of text
```

6. **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.

Below is the example code with explanation included in which we set the bold and italic text styles for text.

```
<TextView
   android:id="@+id/simpleTextView"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="AbhiAndroid"
   android:layout_centerInParent="true"
   android:textSize="40sp"
   android:textStyle="bold|italic"/><!--bold and italic text style of text-->
```

7. **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.

8. **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.

Below is the example code with explanation included in which we set the black color for the background, white color for the displayed text and set 10dp padding from all the side's for text view.

```
<TextView
android:id="@+id/simpleTextView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="AbhiAndroid"
android:layout_centerInParent="true"
android:textSize="40sp"
android:padding="10dp"
android:textColor="#fff"
android:background="#000"/> <!--red color for background of text view-->
```

In Java class:
Below is the example code in which we set the background color of a text view programmatically means in java class.

```
TextView textView = (TextView)findViewById(R.id.textView);
textView.setBackgroundColor(Color.BLACK);//set background color
```

Example of TextView:
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:

Step 1: Create a new project and name it textViewExample.
Select File -> New -> New Project. Fill the forms and click "Finish" button.

Step 2: Open res -> layout -> xml (or) activity_main.xml and add following code. Here we will create a button and a textview in Relative Layout.

```
<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: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" />
</RelativeLayout>
```

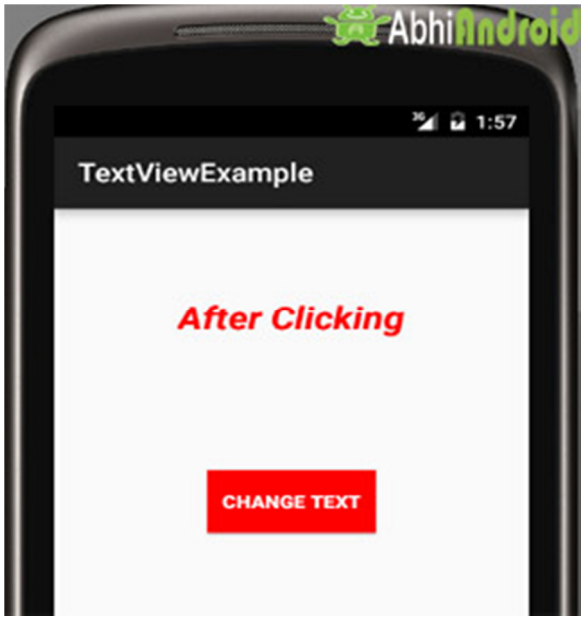
Step 3: Open app -> java -> package and open MainActivity.java and add the following code. Here we will change the text of TextView after the user click on Button.

```
package example.abhiandriod.textviewexample;

import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
```

```
public class MainActivity extends AppCompatActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main); //set the layout
      final TextView simpleTextView = (TextView) findViewById(R.id.simpleTextView);
//get the id for TextView
      Button changeText = (Button) findViewById(R.id.btnChangeText); //get the id for
button
      changeText.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View view) {
            simpleTextView.setText("After Clicking"); //set the text after clicking button
         }
      });
   }


}
```

Output:

Now run the app in Emulator and click on the button. You will see text will change "After Clicking".



TextView Example Output

EditText In Android Studio: Input Field

In Android, EditText is a standard entry widget in android apps. It is an overlay over TextView that configures itself to be editable. EditText is a subclass of TextView with text editing operations. We often use EditText in our applications in order to provide an input or text field, especially in forms. The most simple example of EditText is Login or Sign-in form.

EditText Input Field Example in Android Text Fields in Android Studio are basically EditText:
Important Note: An EditText is simply a thin extension of a TextView. An EditText inherits all the properties of a TextView.

EditText Code:
We can create a EditText instance by declaring it inside a layout(XML file) or by instantiating it programmatically (i.e. in Java Class).

EditText code in XML:

```
<EditText
android:id="@+id/simpleEditText"
android:layout_height="wrap_content"
android:layout_width="match_parent"/>
```

Retrieving / Getting the Value From EditText In Java Class:

Below is the example code of EditText in which we retrieve the value from a 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.simpleEditText);
String editTextValue = simpleEditText.getText().toString();
```

Attributes of EditText:
Now let's we discuss few attributes that helps us to configure a EditText in your xml.

1. id: id is an attribute used to uniquely identify a text EditText. Below is the example code in which we set the id of a edit text.

```
<EditText
android:id="@+id/simpleEditText"
android:layout_height="wrap_content"
android:layout_width="match_parent"/>
```

2. 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.

Below is the example code with explanation included in which we set the right gravity for text of a EditText.

```
<EditText
   android:id="@+id/simpleEditText"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Enter Email"
   android:gravity="right"/><!--gravity of a edit text-->
```

3. text: text attribute is used to set the text in a EditText. We can set the text in xml as well as in the java class.
Below is the example code in which we set the text "Username" in a edit text.

```
<EditText
   android:id="@+id/simpleEditText"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:layout_centerInParent="true"
   android:text="Username"/><!--set text in edit text-->
```

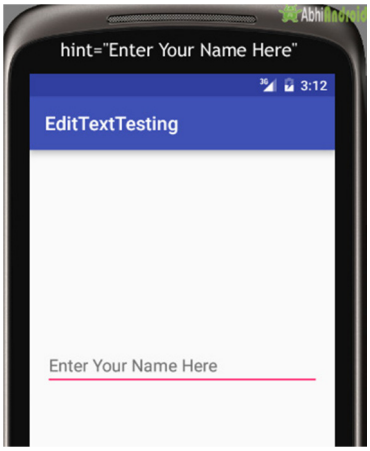
Below is the example code in which we set the text in a text view programmatically means in java class.

```
EditText editText = (EditText)findViewById(R.id.simpleEditText);
editText.setText("Username");//set the text in edit text
```

4. 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.

Below is the example code with explanation in which we set the hint of a edit text.

```
<EditText
   android:id="@+id/simpleEditText"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:layout_centerInParent="true"
   android:hint="Enter Your Name Here" /><!--display the hint-->
```



In java class

Below is the example code in which we set the text in a text view programmatically means in java class.

```
EditText editText = (EditText)findViewById(R.id.simpleEditText);
editText.setHint("Enter Your Name Here");//display the hint
```

5. textColor: textColor attribute is used to set the text color of a text edit text. Color value is in the form of "#argb", "#rgb", "#rrggbb", or "#aarrggbb".

Below is the example code with explanation included in which we set the red color for the displayed text of an edit text.

```
<EditText
   android:id="@+id/simpleEditText"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:layout_centerInParent="true"
   android:text="Password"
   android:textColor="#f00"/><!--set the red text color-->
```

In Java class:
Below is the example code in which we set the text color of a edit text programmatically means in java class.

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);
simpleEditText.setTextColor(Color.RED);//set the red text color
```

6. textColorHint: textColorHint is an attribute used to set the color of displayed hint.

Below is the example code with explanation included in which we set the green color for displayed hint of a edit text.

```
<EditText
   android:id="@+id/simpleEditText"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:layout_centerInParent="true"
   android:hint="Enter Your Name Here"
   android:textColorHint="#0f0"/><!--set the hint color green-->
```

textColorHint in EditText AndroidSetting textColorHint in EditText In Java class:
Below is the example code in which we set the hint color of a edit text programmatically means in java class.

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);
simpleEditText.setHintTextColor(Color.green(0));//set the green hint color
```

7. textSize: textSize attribute is used to set the size of text of a edit text. We can set the text size in sp(scale independent pixel) or dp(density pixel).

Below is the example code in which we set the 25sp size for the text of a edit text.

```
<EditText
   android:id="@+id/simpleEditText"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:layout_centerInParent="true"
   android:text="AbhiAndroid"
   android:textSize="25sp" /><!--set 25sp text size-->
```

Setting textSize in EditText AndroidSetting textSize in EditText in Java class:
Below is the example code in which we set the text size of a edit text programmatically means in java class.

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);
simpleEditText.setTextSize(25);//set size of text
```

8. textStyle: textStyle attribute is used to set the text style of a edit text. The possible text styles are bold, italic and normal. If we need to use two or more styles for a edit text then "|" operator is used for that.

Below is the example code with explanation included, in which we set the bold and italic text styles for text.

```
<EditText
   android:id="@+id/simpleEditText"
```

```
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:text="Email"
android:textSize="25sp"
android:textStyle="bold|italic"/><!--set bold and italic text style-->
```

9. background: background attribute is used to set the background of a edit text. We can set a color or a drawable in the background of a edit text.
Below is the example code with explanation included in which we set the black color for the background, white color for the displayed hint and set 10dp padding from all the side's for edit text.

```
<EditText
  android:id="@+id/simpleEditText"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:layout_centerInParent="true"
  android:hint="Enter Your Name Here"
  android:padding="15dp"
  android:textColorHint="#fff"
  android:textStyle="bold|italic"
  android:background="#000"/><!--set background color black-->
```

Background in EditText AndroidSetting Background in EditText In Java class:
Below is the example code in which we set the background color of a edit text programmatically means in java class.

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);
simpleEditText.setBackgroundColor(Color.BLACK);//set black background color
```

10. 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 edit text.

Example – EditText in Android Studio
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.

EditText Example In Android Studio
Step 1: Create a new project in Android Studio and name it EditTextExample.
Step 2: Now Open res -> layout -> xml (or) activity_main.xml and add following code. In this code we have added multiple edittext and a button with onclick functionality.

```
<?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:id="@+id/activity_main"
   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="com.example.edittextexample.MainActivity">

   <EditText
      android:id="@+id/editText1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      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"
    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>
```

Step 3: Now open app -> java -> package -> MainActivity.java and add the below code. In this we just fetch the text from the edittext, further with the button click event a toast will show the text fetched before.

```
package com.example.edittextexample;

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 {

   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();
            }
        }
    });
  }
}
```

Output:

Now start the AVD in Emulator and run the App. You will see screen asking you to fill the data in required fields like name, password(numeric), email, date, contact number. Enter data and click on button. You will see the data entered will be displayed as Toast on screen.

Button In Android Studio

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 Example

AndroidButton 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.

Button code in XML:

The below code will create Button and write “Abhi Android” text on it.

```
<Button
android:id="@+id/simpleButton"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Abhi Android"/>
```

Attributes of Button in Android:
Now let’s we discuss some important attributes that helps us to configure a Button in your xml file (layout).

1. id: id is an attribute used to uniquely identify a text Button. Below is the example code in which we set the id of a Button.

```
<Button
android:id="@+id/simpleButton"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Abhi Android"/>
```

2. 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.

Below is the example code with explanation included in which we set the right and center vertical gravity for text of a Button.

```
<Button
android:id="@+id/simpleButton"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Abhi Android"
android:layout_centerInParent="true"
android:gravity="right|center_vertical"/><!--set the gravity of button-->
```
Button Gravity in Android

3. text: text attribute is used to set the text in a Button. We can set the text in xml as well as in the java class.

Below is the example code with explanation included in which we set the text “Learning Android @ AbhiAndroid” in a Button.

```
<Button
  android:id="@+id/simpleButton"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:layout_centerInParent="true"
  android:text="Learn Android @ AbhiAndroid"/><!--display text on button-->
```

Setting Text on Button in AndroidSetting Text Using Java class:
Below is the example code in which we set the text on Button programmatically means in java class. The output will be same as the above.

```
Button button = (Button) findViewById(R.id.simpleButton);
button.setText("Learn Android @ AbhiAndroid");//set the text on button
```

4.textColor: textColor attribute is used to set the text color of a Button. Color value is in the form of “#argb”, “#rgb”, “#rrggbb”, or “#aarrggbb”.

Below is the example code with explanation included in which we set the red color for the displayed text of a Button.

```
<Button
android:id="@+id/simpleButton"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:text="AbhiAndroid"
android:textColor="#f00"/><!--red color for the text-->
```

Setting Text Color on Button in AndroidSetting Text Color On Button Inside Java class:
Below is the example code in which we set the text color of a Button programmatically means in java class.

```
Button simpleButton=(Button) findViewById(R.id.simpleButton);
simpleButton.setTextColor(Color.RED);//set the red color for the text
```

5. textSize: textSize attribute is used to set the size of the text on Button. We can set the text size in sp(scale independent pixel) or dp(density pixel).

Below is the example code in which we set the 25sp size for the text of a Button.

```
<Button
android:id="@+id/simpleButton"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:text="AbhiAndroid"
android:textSize="25sp" /><!--25sp text size-->
```

Setting TextSize on Button in AndroidSetting textSize In Java class:
Below is the example code in which we set the text size of a Button programmatically means in java class.

```
Button simpleButton=(Button)findViewById(R.id.simpleButton);
simpleButton.setTextSize(25);//set the text size of button
```

6. textStyle: textStyle attribute is used to set the text style of a Button. The possible text styles are bold, italic and normal. If we need to use two or more styles for a Button then "|" operator is used for that.

Below is the example code with explanation included, in which we set the bold and italic text styles for text of a button.

```
<Button
   android:id="@+id/simpleButton"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:layout_centerInParent="true"
   android:text="AbhiAndroid"
   android:textSize="20sp"
   android:textStyle="bold|italic"/><!--bold and italic text style-->
```

7. background: background attribute is used to set the background of a Button. We can set a color or a drawable in the background of a Button.
Below is the example code in which we set the gren color for the background, Black color for the displayed text and set 15dp padding from all the side's for Button.

```
<Button
android:id="@+id/simpleButton"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:text="Download"
android:textSize="20sp"
android:padding="15dp"
```

android:textStyle="bold|italic"
android:background="#147D03" /><!--Background green color-->
setting background in Button AndroidSetting background in Button In Java class:
Below is the example code in which we set the background color of a Button programmatically means in java class.

Button simpleButton=(Button)findViewById(R.id.simpleButton);
simpleButton.setBackgroundColor(Color.BLACK);//set the black color of button background

8. 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 button.

9. drawableBottom: drawableBottom is the drawable to be drawn to the below of the text.

Below is the example code in which we set the icon to the below of the text.

Make sure you have image saved in your drawable folder name ic_launcher.

```
<Button
   android:id="@+id/simpleButton"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:layout_centerInParent="true"
   android:background="#147D03"
   android:text="Download Code"
   android:textSize="20sp"
   android:padding="15dp"
   android:textStyle="bold|italic"
   android:drawableBottom="@drawable/ic_launcher"/><!--image drawable on button-
->
```

drawableBottom in Button in Android10. drawableTop, drawableRight And drawableLeft: Just like the above attribute we can draw drawable to the left, right or top of text.

In the Below example we set the icon to the right of the text. In the same way you can do for other two attribute by your own:

```
<Button
   android:id="@+id/simpleButton"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
```

```
  android:layout_centerInParent="true"
  android:background="#147D03"
  android:text="Download Code"
  android:textSize="20sp"
  android:padding="15dp"
  android:textStyle="bold|italic"
  android:drawableRight="@drawable/ic_launcher"/>drawableRight of Text on Button
in Android
```

Button Example In Android Studio:

Below is the example of button in which we display two buttons with different background and whenever a user click on the button the text of the button will be displayed in a toast.

Step 1: Create a new project in Android Studio and name it ButtonExample.
Select File -> New -> New Project and Fill the forms and click "Finish" button.
Step 2: Now open res -> layout -> xml (or) activity_main.xml and add following code. Here we are designing the UI of two button in Relative Layout.

```
<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">

  <Button
    android:id="@+id/simpleButton1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="100dp"
    android:background="#00f"
    android:drawableRight="@drawable/ic_launcher"
    android:hint="AbhiAndroid Button1"
    android:padding="5dp"
    android:textColorHint="#fff"
    android:textSize="20sp"
    android:textStyle="bold|italic" />

  <Button
```

```
    android:id="@+id/simpleButton2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:background="#f00"
    android:drawableLeft="@drawable/ic_launcher"
    android:hint="AbhiAndroid Button2"
    android:padding="5dp"
    android:textColorHint="#fff"
    android:textSize="20sp"
    android:textStyle="bold|italic" />
</RelativeLayout>
```

Step 3: Now Open app -> package -> MainActivity.java and the following code. Here using setOnClickListener() method on button and using Toast we will display which button is clicked by user.

```
package example.abhiandriod.buttonexample;

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

public class MainActivity extends AppCompatActivity {

  Button simpleButton1, simpleButton2;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    simpleButton1 = (Button) findViewById(R.id.simpleButton1);//get id of button 1
    simpleButton2 = (Button) findViewById(R.id.simpleButton2);//get id of button 2

    simpleButton1.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        Toast.makeText(getApplicationContext(), "Simple Button 1",
Toast.LENGTH_LONG).show();//display the text of button1
      }
```

```
    });
    simpleButton2.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        Toast.makeText(getApplicationContext(), "Simple Button 2",
Toast.LENGTH_LONG).show();//display the text of button2
      }
    });
  }
}
```

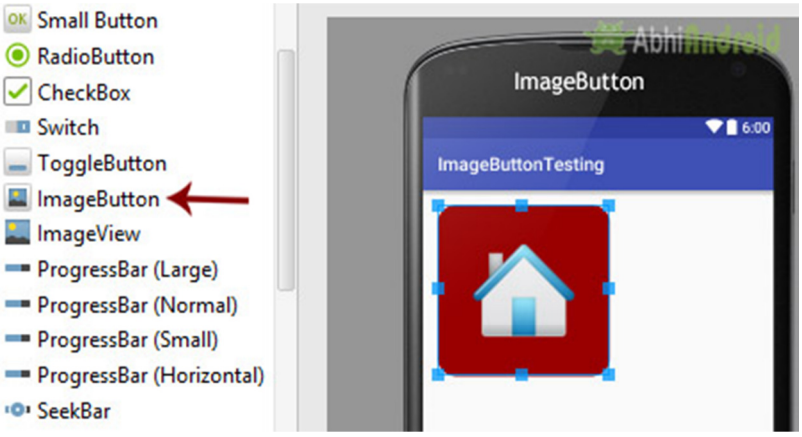
Output:

Now start the AVD in Emulator and run the App. You will see two button. Click on any button and you will see the message on screen which button is clicked.

button example output android

ImageButton In Android Studio

In Android, ImageButton is used to display a normal button with a custom image in a button. In simple words we can say, ImageButton is a button with an image that can be pressed or clicked by the users. By default it looks like a normal button with the standard button background that changes the color during different button states.



An image on the surface of a button is defined within a xml (i.e. layout) by using src attribute or within java class by using setImageResource() method. We can also set an image or custom drawable in the background of the image button.
Important Note: Standard button background image is displayed in the background of button whenever you create an image button. To remove that image, you can define your own background image in xml by using background attribute or in java class by using setBackground() method.

Below is the code and image which shows how custom imagebutton looks in Android:

Important Note: ImageButton has all the properties of a normal button so you can easily perform any event like click or any other event which you can perform on a normal button.

ImageButton code in XML:

```
<!--Make Sure To Add Image Name home in Drawable Folder-->

<ImageButton

android:id="@+id/simpleImageButton"

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:src="@drawable/home" />
```

Attributes of ImageButton:

Now let's we discuss some important attributes that helps us to configure a image button in your xml file (layout).

1. id: id is an attribute used to uniquely identify a image button. Below is the example code in which we set the id of a image button.

```
<ImageButton

android:id="@+id/simpleImageButton"

android:layout_width="wrap_content"

android:layout_height="wrap_content"/>
```

2. src: src is an attribute used to set a source file of image or you can say image in your image button to make your layout look attractive.

Below is the example code in which we set the source of an image button. Make sure you have saved an image in drawable folder name home before using below code.

```
<ImageButton

  android:id="@+id/simpleImageButton"

  android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
android:src="@drawable/home"/> <!--src(source)file from drawable folder which display an imagebutton-->
```

Setting Image Source In ImageButton Using Java class:

We can also set the source image at run time programmatically in java class. For that we use setImageResource() function as shown in below example code.

```
/*Add in Oncreate() funtion after setContentView()*/
ImageButton simpleImageButton = (ImageButton)findViewById(R.id.simpleImageButton);
simpleImageButton.setImageResource(R.drawable.home); //set the image programmatically
```

3. background: background attribute is used to set the background of an image button. We can set a color or a drawable in the background of a Button.

Below is the example code in which we set the black color for the background and an home image as the source of the image button.

```
<ImageButton
android:id="@+id/simpleImageButton"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/home"
android:background="#000"/><!-- black background color for image button-->
```

Setting Background In ImageButton Using Java class:
Below is the example code in which we set the black background color of a image button programmatically means in java class.

```
/*Add in Oncreate() funtion after setContentView()*/
ImageButton simpleImageButton =findViewById(R.id.simpleImageButton);
simpleImageButton.setBackgroundColor(Color.BLACK);
```

4. padding: padding attribute is used to set the padding from left, right, top or bottom of the ImageButton.

- **paddingRight :** set the padding from the right side of the image button**.**
- **paddingLeft :** set the padding from the left side of the image button**.**
- **paddingTop :** set the padding from the top side of the image button**.**
- **paddingBottom :** set the padding from the bottom side of the image button**.**
- **padding :** set the padding from the all side's of the image button**.**

Below is the example code of padding attribute in which we set the 20dp padding from all the side's of a image button.

```
<ImageButton
   android:id="@+id/simpleImageButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:background="#000"
   android:src="@drawable/home"
   android:padding="30dp"/>
```

Example of ImageButton In Android Studio:

In the below example of ImageButton we display two custom image buttons with source and background. One is simple image button with simple background and other one is a round corner image button and whenever you click on an button, the name of the button will be displayed in a toast. Below is the code and final output:

Step 1: Create a new project and name it ImageButtonExample
In this step we create a new project in android studio by filling all the necessary details of the app like app name, package name, api versions etc.

```
Select File -> New -> New Project and Fill the forms and click "Finish" button.
```

Step 3: Right click on **drawable** -> New -> Drawable resource file and create new xml file name **custom_image-buttton.xml** and add following code

In this Step we create drawable xml in which we used solid and corner properties, solid is used to set the background color for the image button and corner is used to set the radius for button corners.

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
   <solid android:color="#900" /><!-- background color for imagebutton-->
   <corners android:radius="20dp" /><!-- round corners for imagebutton-->
</shape>
```

Step 3: Open app -> layout -> **activity_main.xml (or) main.xml** and add following code:

In this step, we open an xml file and add the code which display two custom image buttons by using src, background, gravity and other attributes in Relative Layout.

```
<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">

<!--Make sure to save two images home and youtube in drawable folder-->

    <ImageButton
        android:id="@+id/simpleImageButtonHome"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:background="@drawable/custom_image_button"
        android:padding="20dp"
        android:src="@drawable/home" />

    <ImageButton
        android:id="@+id/simpleImageButtonYouTube"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/simpleImageButtonHome"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:background="#005"
        android:padding="20dp"
        android:src="@drawable/youtube" />

</RelativeLayout>
```

Step 4: Open app -> package -> **MainActivity.java**

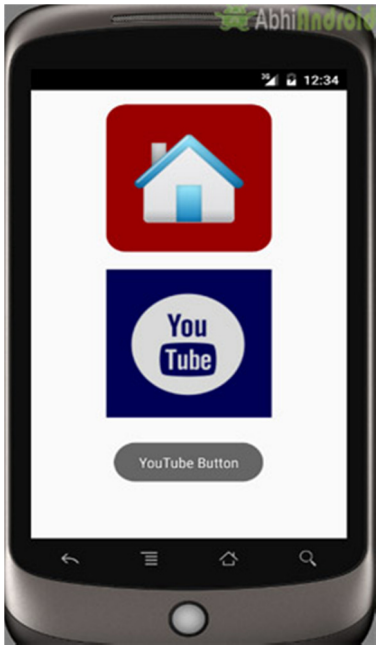
In this step, we add the code to initiate the image button's and then perform click event on them and display the text for selected item using a toast.

```
package example.abhiandriod.imagebuttonexample;


import android.app.Activity;
import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ImageButton;
import android.widget.Toast;



public class MainActivity extends Activity {


  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
// initiate view's
    ImageButton simpleImageButtonHome = (ImageButton)findViewById(R.id.simpleI
mageButtonHome);
    ImageButton simpleImageButtonYouTube = (ImageButton)findViewById(R.id.simpl
eImageButtonYouTube);


// perform click event on button's
    simpleImageButtonHome.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        Toast.makeText(getApplicationContext(),"Home  Button",Toast.LENGTH_LONG)
.show();// display the toast on home button click
      }
```

```
    });
    simpleImageButtonYouTube.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        Toast.makeText(getApplicationContext(),"YouTube  Button",Toast.LENGTH_LONG).show();// display the toast on you tube button click
      }
    });
  }
}
```

Output:

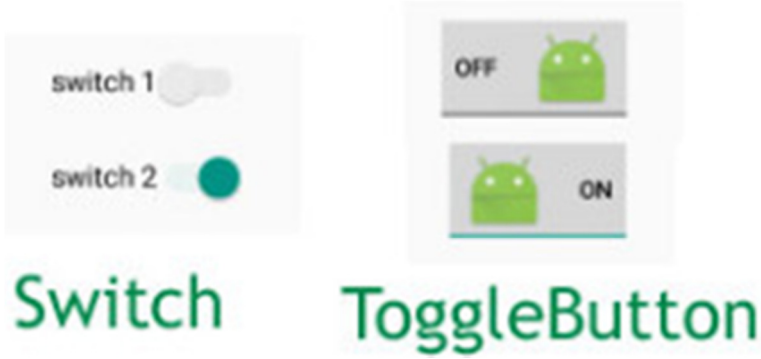
Now start the AVD in Emulator and run the App. You will see two ImageButton out of which top one is round corner. Click on any image and its name will be displayed on screen.



ToggleButton (On/Off) In Android

In Android, ToggleButton is used to display checked and unchecked state of a button. ToggleButton basically an off/on button with a light indicator which indicate the current state of toggle button. The most simple example of ToggleButton is doing on/off in sound, Bluetooth, wifi, hotspot etc. It is a subclass of compoundButton.

ToggleButton Vs Switch In Android:
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.



Important Note: 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.simpleToggleBu
tton); // initiate a toggle button
Boolean ToggleButtonState = simpleToggleButton.isChecked();
```

Attributes of ToggleButton:

Now let's we discuss important attributes that helps us to configure a Toggle Button in XML file (layout).

1. id: id is an attribute used to uniquely identify a toggle button.

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

2. 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. The default value of checked attribute is false. We can also set the current state programmatically.

Below we set true value for checked attribute sets the current state to checked.

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

Setting Checked Of ToogleButton Current State In Java Class:
Below we set the current state of toggle button to checked:

```
/*Add in Oncreate() funtion after setContentView()*/
ToggleButton simpleToggleButton = (ToggleButton) findViewById(R.id.simpleToggleButton); // initiate a toggle button
simpleToggleButton.setChecked(true); // set the current state of a toggle button
```

3. gravity: The gravity attribute is an optional attribute which is used to control the alignment of the text in ToogleButton like left, right, center, top, bottom, center_vertical, center_horizontal etc.

```
<ToggleButton
   android:id="@+id/simpleToggleButton"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:layout_centerHorizontal="true"
   android:checked="true"
   android:gravity="right|center_vertical"/>
```

4. textOn And textOff: textOn attribute is used to set the text when toggle button is in checked/on state. We can set the textOn in XML as well as in the java class.
Below is the example code with explanation included in which we set the textOn "Enabble Attribute Of Toggle button" for a toggle button.

```
<ToggleButton
   android:id="@+id/simpleToggleButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="true"
   android:layout_centerHorizontal="true"
   android:textOff="Disable"
   android:textOn="Enable"/>
```

Setting textOn and textOff Of ToggleButton In Java class:

```
/*Add in Oncreate() funtion after setContentView()*/

// initiate toggle button

ToggleButton simpleToggleButton = (ToggleButton) findViewById(R.id.simpleToggleBu
tton);

// displayed text of the toggle button whenever it is in checked or on state

simpleToggleButton.setTextOn("TextOn Attribute Of Toggle b3`utton");

// displayed text of the toggle button whenever it is in unchecked or off state

simpleToggleButton.setTextOff("TextOff Attribute Of Toggle b3`utton");
```

5. textColor: textColor attribute is used to set the text color of a toggle button. Color value is in the form of "#argb", "#rgb", "#rrggbb", or "#aarrggbb".

Below we set the red color for the displayed text of a Toggle button.

```
<ToggleButton
   android:id="@+id/simpleToggleButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="false"
   android:textOff="On State"
   android:textOn="Off State"
   android:layout_centerHorizontal="true"
   android:textColor="#f00" />
```

Setting textColor Of ToggleButton In Java class:

```
/*Add in Oncreate() funtion after setContentView()*/

ToggleButton simpleToggleButton = (ToggleButton) findViewById(R.id.simpleToggleButton);// initiate toggle button

simpleToggleButton.setTextColor(Color.RED); //red color for displayed text of toggle button
```

6. textSize: textSize attribute set the size of the text of a toggle button. We can set the text size in sp(scale independent pixel) or dp(density pixel).

Below we set the 25sp size for the text of a toggle button.

```
<ToggleButton
   android:id="@+id/simpleToggleButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="false"
   android:textOff="On State"
   android:textOn="Off State"
   android:layout_centerHorizontal="true"
   android:textColor="#f00"
   android:textSize="25sp"/>
```

Setting textSize Of ToggleButton Text In Java class:

```
/*Add in Oncreate() funtion after setContentView()*/

ToggleButton simpleToggleButton = (ToggleButton) findViewById(R.id.simpleToggleButton); // initiate toggle button

simpleToggleButton.setTextSize(25); // set 25sp displayed text size of toggle button
```

7. textStyle: textStyle attribute is used to set the text style of the text of a Toggle button. You can set bold, italic and normal. If you need to use two or more styles for a text view then "|" operator is used for that.

Below we set the bold and italic text styles for text of a toggle button.

```
<ToggleButton
   android:id="@+id/simpleToggleButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="true"
   android:textOff="Off State"
   android:textOn="On State"
   android:textSize="25sp"
   android:layout_centerHorizontal="true"
   android:textColor="#f00"
   android:textStyle="bold|italic"/>
```

8. background: background attribute is used to set the background of a toggle button. We can set a color or a drawable in the background of a toggle button.
Below we set the black color for the background and red color for the displayed text of a ToggleButton.

```
<ToggleButton
   android:id="@+id/simpleToggleButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="true"
   android:textOff="Off State"
   android:textOn="On State"
```

```
android:textSize="25sp"
android:layout_centerHorizontal="true"
android:textStyle="bold|italic"
android:textColor="#f00"
android:background="#000"/>
```

Setting Background Of ToggleButton Text In Java class:

```
/*Add in Oncreate() funtion after setContentView()*/
ToggleButton simpleToggleButton = (ToggleButton) findViewById(R.id.simpleToggleButton);
simpleToggleButton.setBackgroundColor(Color.BLACK);
```

9. padding: padding attribute is used to set the padding from left, right, top or bottom.

- **paddingRight :**set the padding from the right side of the toggle button**.**
- **paddingLeft :**set the padding from the left side of the toggle button**.**
- **paddingTop :**set the padding from the top side of the toggle button**.**
- **paddingBottom :**set the padding from the bottom side of the toggle button**.**
- **Padding :**set the padding from the all side's of the toggle button**.**

Below we set the 40dp padding from all the side's of the toggle button.

```
<ToggleButton
android:id="@+id/simpleToggleButton"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:checked="true"
android:textOff="Off State"
android:textOn="On State"
android:textSize="25sp"
android:layout_centerHorizontal="true"
```

```
android:textColor="#f00"
android:padding="40dp"/>
```

10. drawableBottom, drawableTop, drawableRight And drawableLeft: These attribute draw the drawable below, top, right and left of the text of ToggleButton. Below we set the icon to the Top of the text of a ToggleButton. In the similar way you can try for other three attribute yourself.

```
<!--Make sure to add ic_launcher image in drawable folder-->
<ToggleButton
   android:id="@+id/simpleToggleButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="true"
   android:textOff="Off State"
   android:textOn="On State"
   android:layout_centerHorizontal="true"
   android:textColor="#000"
   android:drawableTop="@drawable/ic_launcher" />
```

ToggleButton Example In Android Studio:

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. Below is the final output, download code and step by step explanation:

Step 1: Create a new project and name it ToggleButtonExample
In this step we create a new project for ToggleButton in Android Studio by filling all the necessary details of the app like app name, package name, api versions etc.

Select File -> New -> New Project -> Fill the forms and click "Finish" button.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we open an xml file and add the code for displaying two toggle button and one normal Button.

```
<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="vertical"
   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">

   <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_launcher"
        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>
```

Step 3: Open app -> java -> package -> **MainActivity.java**

In this step we open MainActivity where we add the code to initiate the Toggle Buttons and normal Button. After initiating we perform click event on button and display the text of current state of ToggleButton using a Toast.

```
package example.abhiandriod.togglebuttonexample;


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


public class MainActivity extends AppCompatActivity {


  ToggleButton simpleToggleButton1, simpleToggleButton2;
  Button submit;


  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // initiate toggle button's
    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" + "To
ggleButton2 : " + simpleToggleButton2.getText();
        Toast.makeText(getApplicationContext(), status, Toast.LENGTH_SHORT).show(
); // display the current state of toggle button's
```

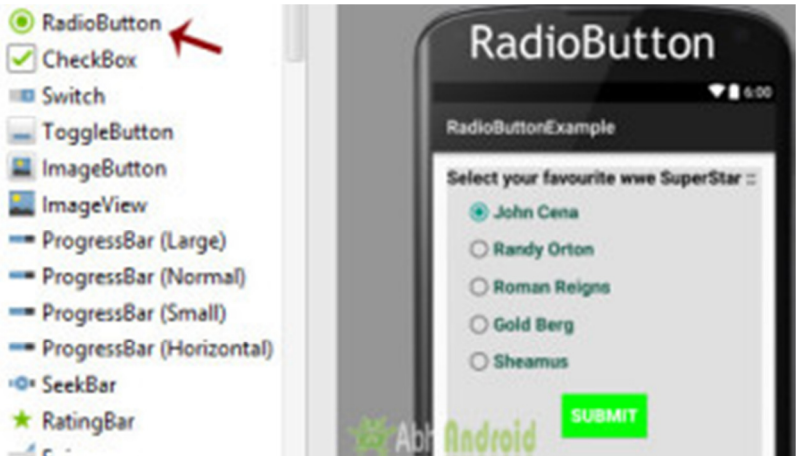
```
        }
    });
  }


}
```

Output:

Now start the AVD in Emulator and run the App. You will see two ToggleButton and submit Button. Click on the submit button which will display the state of ToggleButton.

RadioButton & RadioGroup In Android Studio

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.

Important Note: 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();
```

Attributes of RadioButton In Android:

Now let's we discuss important attributes that helps us to create a beautiful radio button in xml file (layout).

1. id: id is an attribute used to uniquely identify a radio button.

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

2. checked: checked attribute in radio button is used to set the current state of a radio button. We can set it either true or false where true shows the checked state and false shows unchecked state of a radio button. As usual default value of checked attribute is false. We can also set the current state in JAVA.

Below we set true value for checked attribute which sets the current state to checked of a Button

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

Setting checked State Of RadioButton In Java Class:
Below code set the current state of RadioButton to checked programmatically.

```
/*Add in Oncreate() funtion after setContentView()*/
// initiate a radio button
```

```
RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton);


// set the current state of a radio button
simpleRadioButton.setChecked(true);
```

3. text: text attribute is used to set the text in a radio button. We can set the text both ways either in XML or in JAVA class.

Below is the example code with explanation included in which we set the text "I am a radiobutton" of a radio button.

```
<RadioButton
  android:id="@+id/simpleRadioButton"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:checked="true"
  android:layout_centerHorizontal="true"
  android:text="I am a radiobutton" /> <!-- displayed text of radio button-->
```

Setting text of RadioButton In Java class:

Below we set the text of a radio button programmatically:

```
/*Add in Oncreate() funtion after setContentView()*/
RadioButton simpleRadioButton=(RadioButton) findViewById(R.id.simpleRadioButton);
simpleRadioButton.setText("I am a radiobutton");
```

4. gravity: The gravity attribute is an optional attribute which is used to control the alignment of text like left, right, center, top, bottom, center_vertical, center_horizontal etc.
Below is the example code with explanation included in which we set the center gravity for the text of a radio button.

```
<RadioButton
  android:id="@+id/simpleRadioButton"
```

```
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:checked="true"
android:text="I am a Button"
android:gravity="center"/>
```

5. textColor: textColor attribute is used to set the text color of a radio button. Color value is in the form of "#argb", "#rgb", "#rrggbb", or "#aarrggbb".
Below we set the red color for the displayed text of a radio button.

```
<RadioButton
   android:id="@+id/simpleRadioButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="true"
   android:layout_centerHorizontal="true"
   android:text="Male"
   android:textColor="#f00" />
```

Setting textColor of RadioButton text In Java class:
Below we set the text color of a radio button programmatically.

```
/*Add in Oncreate() funtion after setContentView()*/
```

```
RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton);// initiate radio button


simpleRadioButton.setTextColor(Color.RED); //red color for displayed text of radio button
```

6. textSize: textSize attribute is used to set the size of the text of a radio button. We can set the text size in sp(scale independent pixel) or dp(density pixel).

Below we set the 25sp size for the text of a radio button.

```
<RadioButton
   android:id="@+id/simpleRadioButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="true"
   android:layout_centerHorizontal="true"
   android:text="AbhiAndroid"
   android:textColor="#f00"
   android:textSize="25sp"/>
```

Setting textSize Of RadioButton Text In Java class:
Below we set the text size of a radio button programmatically:

```
/*Add in Oncreate() funtion after setContentView()*/
RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton); // initiate radio button


simpleRadioButton.setTextSize(25); // set 25sp displayed text size of radio button
```

7. textStyle: textStyle attribute is used to set the text style of the text of a radio button. 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.

Below is the example code with explanation included in which we set the bold and italic text styles for text of a radio button.

```
<RadioButton
   android:id="@+id/simpleRadioButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="true"
   android:textSize="25sp"
   android:layout_centerHorizontal="true"
   android:text="Male"
   android:textColor="#f00"
   android:textStyle="bold|italic"/>
```

8. background: background attribute is used to set the background of a radio button. We can set a color or a drawable in the background of a radio button.
Below we set the black color for the background and red color for the displayed text of a radio button.

```
<RadioButton
   android:id="@+id/simpleRadioButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="true"
   android:textSize="25sp"
   android:textStyle="bold|italic"
   android:padding="20dp"
```

```
android:layout_centerHorizontal="true"
android:text="Male"
android:textColor="#f00"
android:background="#000"/>
```

Setting Background Of RadioButton In Java class:
Below we set the background color of a radio button programmatically.

```
/*Add in Oncreate() funtion after setContentView()*/
RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton);

simpleRadioButton.setBackgroundColor(Color.BLACK);
```

9. padding: padding attribute is used to set the padding from left, right, top or bottom.

- **paddingRight:** set the padding from the right side of the radio button.
- **paddingLeft :** set the padding from the left side of the radio button.
- **paddingTop :** set the padding from the top side of the radio button.
- **paddingBottom:** set the padding from the bottom side of the radio button.
- **Padding:** set the padding from the all side's of the radio button.

Below we set padding attribute of 20dp padding from all the side's of the radio button.

```
<RadioButton
  android:id="@+id/simpleRadioButton"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:checked="true"
  android:textSize="25sp"
  android:textStyle="bold|italic"
```

```
android:layout_centerHorizontal="true"
android:text="AbhiAndroid"
android:textColor="#f00"
android:padding="40dp"/>
```

10. drawableBottom, drawableTop, drawableLeft And drawableRight: These attribute draw the drawable to the below of the text of RadioButton.
Below we set the icon to the right of the text of a RadioButton.

```
<RadioButton
android:id="@+id/simpleRadioButton"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:checked="true"
android:textSize="25sp"
android:padding="20dp"
android:layout_centerHorizontal="true"
android:text="AbhiAndroid"
android:textColor="#f00"
android:drawableRight="@drawable/ic_launcher" />
```

Example Of RadioButton And RadioGroup in Android Studio:

Below is the example of Radiobutton in Android where we display five radio buttons with background and other attributes. The radio buttons are used to select your favorite WWE superstar with one "submit" button. Below is the final output, download code and step by step explanation of tutorial:

Step 1: Create a new project and name it RadioButtonExample
Select File -> New -> New Project and Fill the forms and click "Finish" button.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we open an xml file and add the code for displaying 5 RadioButton and one normal button.

```
<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="vertical"
   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">

   <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="Select your favourite wwe SuperStar :: "
   android:textColor="#000"
   android:textSize="20sp"
   android:textStyle="bold" />

<RadioGroup
   android:layout_width="wrap_content"
   android:layout_height="wrap_content">

   <RadioButton
      android:id="@+id/johnCena"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginLeft="20dp"
      android:layout_marginTop="10dp"
      android:checked="true"
      android:text="@string/johnCena"
      android:textColor="#154"
      android:textSize="20sp"
      android:textStyle="bold" />

   <RadioButton
      android:id="@+id/randyOrton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginLeft="20dp"
      android:layout_marginTop="10dp"
      android:checked="false"
      android:text="@string/randyOrton"
      android:textColor="#154"
```

```
    android:textSize="20sp"
    android:textStyle="bold" />

<RadioButton
    android:id="@+id/romanReigns"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="20dp"
    android:layout_marginTop="10dp"
    android:checked="false"
    android:text="@string/romanReigns"
    android:textColor="#154"
    android:textSize="20sp"
    android:textStyle="bold" />

<RadioButton
    android:id="@+id/goldBerg"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="20dp"
    android:layout_marginTop="10dp"
    android:checked="false"
    android:text="@string/goldBerg"
    android:textColor="#154"
    android:textSize="20sp"
    android:textStyle="bold" />


<RadioButton
    android:id="@+id/sheamus"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="20dp"
```

```
        android:layout_marginTop="10dp"
        android:checked="false"
        android:text="@string/sheamus"
        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"
      android:text="Submit"
      android:textColor="#fff"
      android:textSize="20sp"
      android:textStyle="bold" />
  </LinearLayout>



</LinearLayout>
```

Step 3: Open src -> package -> **MainActivity.java**

In this step we open MainActivity and add the code to initiate the RadioButton and normal button. We also perform click event on button and display the selected superstar's name by using a Toast.

```
package example.gb.radiobuttonexample;


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

```
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

   RadioButton johnCena, randyOrton, goldBerg, romanReigns, sheamus;
   String selectedSuperStar;
   Button submit;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      johnCena = (RadioButton) findViewById(R.id.johnCena);
      randyOrton = (RadioButton) findViewById(R.id.randyOrton);
      goldBerg = (RadioButton) findViewById(R.id.goldBerg);
      romanReigns = (RadioButton) findViewById(R.id.romanReigns);
      sheamus = (RadioButton) findViewById(R.id.sheamus);
      submit = (Button) findViewById(R.id.submitButton);
      submit.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            if (randyOrton.isChecked()) {
               selectedSuperStar = randyOrton.getText().toString();
            } else if (sheamus.isChecked()) {
               selectedSuperStar = sheamus.getText().toString();
            } else if (johnCena.isChecked()) {
               selectedSuperStar = johnCena.getText().toString();
            } else if (romanReigns.isChecked()) {
               selectedSuperStar = romanReigns.getText().toString();
            } else if (goldBerg.isChecked()) {
```

```
            selectedSuperStar = goldBerg.getText().toString();
        }
        Toast.makeText(getApplicationContext(), selectedSuperStar, Toast.LENGTH_LO
NG).show(); // print the value of selected super star
      }
    });
  }
}
```

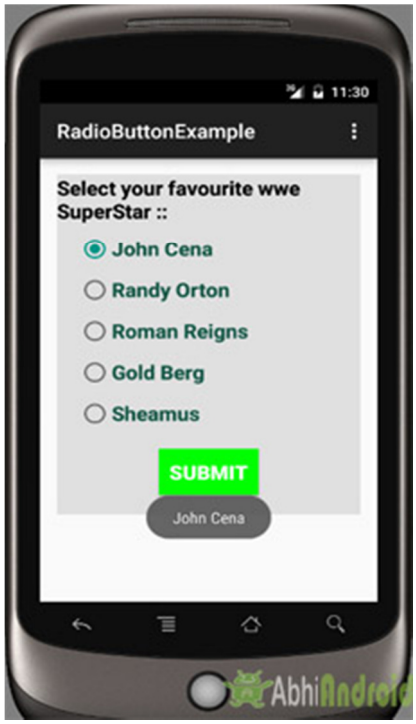
Step 4: Open res -> values -> **strings.xml**

In this step we open String file which is used to store string data of the app.

```
<resources>
  <string name="app_name">RadioButtonExample</string>
  <string name="hello_world">Hello world!</string>
  <string name="action_settings">Settings</string>
  <string name="randyOrton">Randy Orton</string>
  <string name="johnCena">John Cena</string>
  <string name="romanReigns">Roman Reigns</string>
  <string name="goldBerg">Gold Berg</string>
  <string name="sheamus">Sheamus</string>
</resources>
```

Run The App:

Now run the App in Emulator and you will see 5 RadioButton in RadioGroup listing WWE superstar name. Now choose your favorite one and click on Submit Button. The name will be displayed on Screen.



ProgressBar In Android Studio

In Android, ProgressBar is used to display the status of work being done like analyzing status of work or downloading a file etc. In Android, by default a progress bar will be displayed as a spinning wheel but If we want it to be displayed as a horizontal bar then we need to use style attribute as horizontal. It mainly use the **"android.widget.ProgressBar"** class.

Important Note: A progress bar can also be made indeterminate. In this mode a progress bar shows a cyclic animation without an indication of progress. This mode is used in application when we don't know the amount of work to be done.
To add a progress bar to a layout (xml) file, you can use the <ProgressBar> element. By default, a progress bar is a spinning wheel (an indeterminate indicator). To change to a horizontal progress bar, apply the progress bar's horizontal style.

ProgressBar code:

```
<ProgressBar
android:id="@+id/simpleProgressBar"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

Horizontal ProgressBar code:

```
<ProgressBar
android:id="@+id/simpleProgressBar"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
```

```
style="@style/Widget.AppCompat.ProgressBar.Horizontal"/>
```

Important Methods Used In ProgressBar:

1. getMax() – returns the maximum value of progress bar

We can get the maximum value of the progress bar in java class. This method returns a integer value. Below is the code to get the maximum value from a Progress bar.

```
ProgressBar simpleProgressBar=(ProgressBar) findViewById(R.id.simpleProgressBar); // initiate the progress bar

int maxValue=simpleProgressBar.getMax(); // get maximum value of the progress bar
```

2. getProgress() – returns current progress value

We can get the current progress value from a progress bar in java class. This method also returns a integer value. Below is the code to get current progress value from a Progress bar.

```
ProgressBar simpleProgressBar=(ProgressBar)findViewById(R.id.simpleProgressBar); // initiate the progress bar

int progressValue=simpleProgressBar.getProgress(); // get progress value from the progress bar
```

Attributes of ProgressBar In Android:

Now let's discuss important attributes that helps us to configure a Progress bar in xml file (layout).

1. id: id is an attribute used to uniquely identify a Progress bar.

```
<ProgressBar

android:id="@+id/simpleProgressBar"

android:layout_width="fill_parent"

android:layout_height="wrap_content"

style="@style/Widget.AppCompat.ProgressBar.Horizontal"/>
```

2. max: max is an attribute used in android to define maximum value of the progress can take. It must be an integer value like 100, 200 etc.

Below we set 100 maximum value for a progress bar.

```
<ProgressBar

  android:id="@+id/simpleProgressBar"

  android:layout_width="fill_parent"
```

```
android:layout_height="wrap_content"
style="@style/Widget.AppCompat.ProgressBar.Horizontal"
android:max="100" /><!--set 100 maximum value for the progress bar-->
```

Set Max Value of ProgressBar In Java Class :

```
ProgressBar simpleProgressBar=(ProgressBar) findViewById(R.id.simpleProgressBar);
// initiate the progress bar
simpleProgressBar.setMax(100);
```

3. progress: progress is an attribute used in android to define the default progress value between 0 and max. It must be an integer value.
Below we set the 100 max value and then set 50 default progress.

```
<ProgressBar
android:id="@+id/simpleProgressBar"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:max="100"
style="@style/Widget.AppCompat.ProgressBar.Horizontal"
android:progress="50"/>
```

Setting Progress Value of ProgressBar In Java Class :

```
ProgressBar simpleProgressBar=(ProgressBar)findViewById(R.id.simpleProgressBar);
// initiate the progress bar
simpleProgressBar.setMax(100); // 100 maximum value for the progress value
```

```
simpleProgressBar.setProgress(50); // 50 default progress value for the progress bar
```

4. progressDrawable: progress drawable is an attribute used in Android to set the custom drawable for the progress mode.

Below we set a custom gradient drawable for the progress mode of a progress bar. Before you try below code make sure to download a progress icon from the web and add in your drawable folder.

Step 1: Add this code in activity_main.xml or main.xml inside layout.

```
<ProgressBar
android:id="@+id/simpleProgressBar"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:max="100"
android:progress="60"
android:layout_marginTop="100dp"
style="@style/Widget.AppCompat.ProgressBar.Horizontal"
android:progressDrawable="@drawable/custom_progress"/><!--custom progress dra
wable for progress mode-->
```

Step 2: Create a new drawable resource xml in drawable folder and name it custom_progress. Here add the below code which creates gradient effect in progressbar.

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >
  <item>
    <shape>
      <gradient
        android:endColor="#fff"
        android:startColor="#1f1"
        android:useLevel="true" />
    </shape>
  </item>
</layer-list>
```

5. background: background attribute is used to set the background of a Progress bar.
We can set a color or a drawable in the background of a Progress bar.
Below we set the black color for the background of a Progress bar.

```
<ProgressBar
   android:id="@+id/simpleProgressBar"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:max="100"
   android:progress="50"
   style="@style/Widget.AppCompat.ProgressBar.Horizontal"
   android:background="#000"/>
```

6. indeterminate: indeterminate attribute is used in Android to enable the indeterminate mode. In this mode a progress bar shows a cyclic animation without an indication of progress. This mode is used in application when we don't know the amount of work to be done. In this mode the actual working will not be shown.
In below code we set the indeterminate to true.

```
<ProgressBar
   android:id="@+id/simpleProgressBar"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:max="100"
   android:progress="50"
   android:background="#000"
   android:padding="20dp" style="@style/Widget.AppCompat.ProgressBar.Horizontal"
   android:indeterminate="true"/>
```

Setting indeterminate of ProgressBar In Java class:

```
ProgressBar simpleProgressBar=(ProgressBar)findViewById(R.id.simpleProgressBar);
// initiate the progress bar

simpleProgressBar.setBackgroundColor(Color.BLACK); // black background color for th
e progress bar
```

7. padding: padding attribute is used to set the padding from left, right, top or bottom of ProgressBar.

- **paddingRight:** set the padding from the right side of the Progress bar**.**
- **paddingLeft:** set the padding from the left side of the Progress bar**.**
- **paddingTop:** set the padding from the top side of the Progress bar**.**
- **paddingBottom:** set the padding from the bottom side of the Progress bar**.**
- **Padding:** set the padding from the all side's of the Progress bar**.**

Below we set the 20dp padding from all the side's of the Progress bar.

```
<ProgressBar
   android:id="@+id/simpleProgressBar"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:max="100"
   android:progress="50"
   android:background="#000"
   style="@style/Widget.AppCompat.ProgressBar.Horizontal"
   android:padding="20dp"/>
```

ProgressBar Example In Android Studio:

In the first example of ProgressBar we displayed a default spinning wheel progress bar and a start button whenever a user click on the button the progress bar is displayed. Below is the final output, download code and step by step explanation:

Step 1: Create a new project and name it ProgressBarExample.

```
Select File -> New -> New Project... then Fill the forms and click "Finish" button.
```

Step 2: Open res -> layout -> **activity_main.xml (or) main.xml** and add following code:

In this step we open an xml file and add the code for displaying a progress bar and set its visibility to invisible and one start button.

```
<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">

   <ProgressBar
      android:id="@+id/simpleProgressBar"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:visibility="invisible"
      android:layout_centerHorizontal="true"/>

   <Button
      android:id="@+id/startButton"
      android:layout_width="200dp"
      android:layout_height="wrap_content"
      android:text="Start"
      android:textSize="20sp"
      android:textStyle="bold"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="100dp"
      android:padding="10dp"
      android:background="#0f0"
      android:textColor="#fff"/>
</RelativeLayout>
```

Step 3: Now Open src -> package -> **MainActivity.java**

In this step we open MainActivity where we add the code to initiate the progress bar & button and then perform click event on button which display the progress bar.

```
package example.gb.progressbarexample;
import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ProgressBar;
import android.widget.Button;


public class MainActivity extends AppCompatActivity {


  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // initiate progress bar and start button
    final ProgressBar simpleProgressBar = (ProgressBar) findViewById(R.id.simplePro
gressBar);
    Button startButton = (Button) findViewById(R.id.startButton);
    // perform click event on button
    startButton.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        // visible the progress bar
        simpleProgressBar.setVisibility(View.VISIBLE);
      }
    });
  }
}
```

Output:

Now start the AVD in Emulator and run the App. Click on the start button and Progress Bar will be displayed on screen.

ImageView In Android

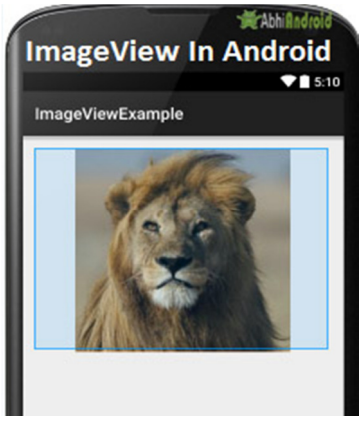
In Android, ImageView class is used to display an image file in application. Image file is easy to use but hard to master in Android, because of the various screen sizes in Android devices. An android is enriched with some of the best UI design widgets that allows us to build good looking and attractive UI based application.

Important Note: ImageView comes with different configuration options to support different scale types. Scale type options are used for scaling the bounds of an image to the bounds of the imageview. Some of them scaleTypes configuration properties are center, center_crop, fit_xy, fitStart etc. You can read our ScaleType tutorial to learn all details on it.

Below is an ImageView code in XML:

Make sure to save lion image in drawable folder

```
<ImageView
android:id="@+id/simpleImageView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:src="@drawable/lion" />
```



Attributes of ImageView:

Now let's we discuss some important attributes that helps us to configure a ImageView in your xml file.

1. id: id is an attribute used to uniquely identify a image view in android. Below is the example code in which we set the id of a image view.

```
<ImageView
android:id="@+id/simpleImageView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
/>
```

2. src: src is an attribute used to set a source file or you can say image in your imageview to make your layout attractive.

Below is the example code in which we set the source of a imageview lion which is saved in drawable folder.

```
<ImageView
android:id="@+id/simpleImageView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:src="@drawable/lion" /><!--set the source of an image view-->
```

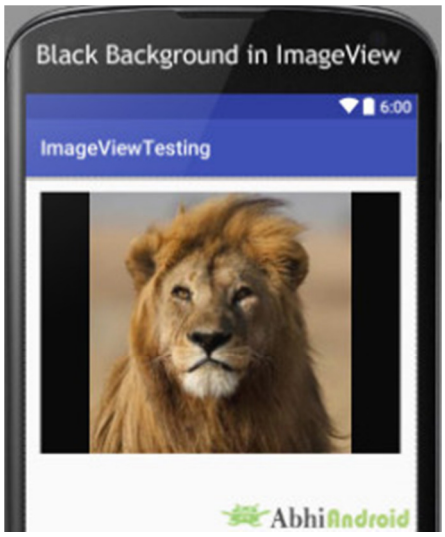
In Java:

We can also set the source image at run time programmatically in java class. For that we use setImageResource() method as shown in below example code.

```
/*Add in Oncreate() funtion after setContentView()*/
ImageView simpleImageView=(ImageView) findViewById(R.id.simpleImageView);
simpleImageView.setImageResource(R.drawable.lion);
```

3. background: background attribute is used to set the background of a ImageView. We can set a color or a drawable in the background of a ImageView. Below is the example code in which we set the black color in the background and an image in the src attribute of image view.

```
<ImageView
  android:id="@+id/simpleImageView"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:src="@drawable/lion"
  android:background="#000"/>
```



In Java:
We can also set the background at run time programmatically in java class. In below example code we set the black color in the background of a image view.

```
/*Add in Oncreate() funtion after setContentView()*/
ImageView simpleImageView=(ImageView) findViewById(R.id.simpleImageView);
simpleImageView.setBackgroundColor(Color.BLACK);//set black color in background of a image view in j
ava class
```

4. padding: padding attribute is used to set the padding from left, right, top or bottom of the Imageview.

- **paddingRight:** set the padding from the right side of the image view.
- **paddingLeft:** set the padding from the left side of the image view.
- **paddingTop:** set the padding from the top side of the image view.
- **paddingBottom:** set the padding from the bottom side of the image view.
- **padding:** set the padding from the all side's of the image view.

Below is the example code of padding attribute in which we set the 30dp padding from all the side's of a image view.

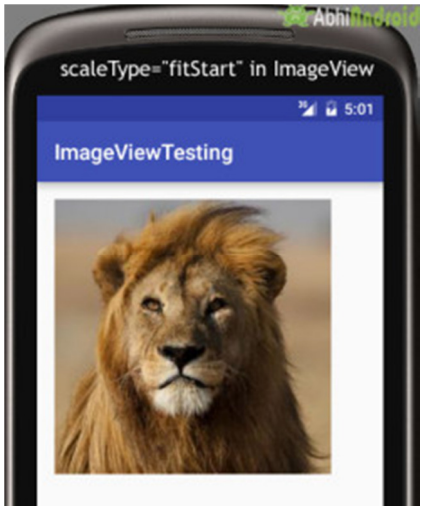
```
<ImageView
   android:id="@+id/simpleImageView"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:background="#000"
   android:src="@drawable/lion"
   android:padding="30dp"/>
```

5. scaleType: scaleType is an attribute used to control how the image should be re-sized or moved to match the size of this image view. **The value for scale type attribute can be fit_xy, center_crop, fitStart etc.**
Below is the example code of scale type in which we set the scale type of image view to fit_xy.

```
<ImageView
android:id="@+id/simpleImageView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:src="@drawable/lion"
android:scaleType="fitXY"/>
```

Let's we take an another example of scale type to understand the actual working of scale type in a image view.
In below example code we set the value for scale type "fitStart" which is used to fit the image in the start of the image view as shown below:

```
<ImageView
   android:id="@+id/simpleImageView"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:src="@drawable/lion"
   android:scaleType="fitStart"/>
```



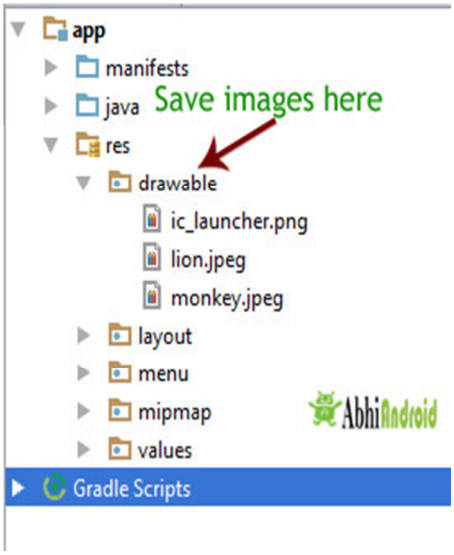
Example of ImageView:

Below is the example of imageview in which we display two animal images of Lion and Monkey. And whenever user click on an image Animal name is displayed as toast on screen. Below is the final output and code:

Step 1: Create a new project and name it ImageViewExample.
In this step we create a new project in android studio by filling all the necessary details of the app like app name, package name, api versions etc.

Select File -> New -> New Project and Fill the forms and click "Finish" button.

Step 2: Download two images lion and monkey from the web. Now save those images in the drawable folder of your project.



Step 3: Now open res -> layout -> activity_main.xml (or) main.xml and add following code:
In this step we add the code for displaying an image view on the screen in a relative layout. **Here make sure you have already saved two images name lion and monkey in your drawable folder.**

```
<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">



   <ImageView
      android:id="@+id/simpleImageViewLion"
      android:layout_width="fill_parent"
      android:layout_height="200dp"
      android:scaleType="fitXY"
      android:src="@drawable/lion" />

   <ImageView
   android:id="@+id/simpleImageViewMonkey"
   android:layout_width="fill_parent"
   android:layout_height="200dp"
   android:layout_below="@+id/simpleImageViewLion"
   android:layout_marginTop="10dp"
   android:scaleType="fitXY"
   android:src="@drawable/monkey" />


</RelativeLayout>
```

Step 4: Now open app -> java -> package -> MainActivity.java and add the following code:

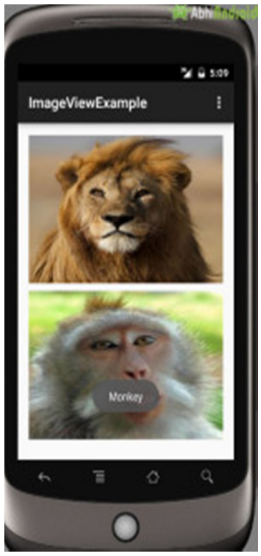
In this step we add the code to initiate the image view's and then perform click event on them.

```
package example.abhiandriod.imageviewexample;


import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ImageView;
import android.widget.Toast;


public class MainActivity extends AppCompatActivity {


   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      ImageView simpleImageViewLion = (ImageView) findViewById(R.id.simpleImageViewLion);//get th
e id of first image view
      ImageView simpleImageViewMonkey = (ImageView) findViewById(R.id.simpleImageViewMonkey);/
/get the id of second image view
      simpleImageViewLion.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View view) {
            Toast.makeText(getApplicationContext(), "Lion", Toast.LENGTH_LONG).show();//display the text
on image click event
         }
      });
      simpleImageViewMonkey.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View view) {
            Toast.makeText(getApplicationContext(), "Monkey", Toast.LENGTH_LONG).show();//display the
text on image click event
         }
      });
   }
}
```

Output:

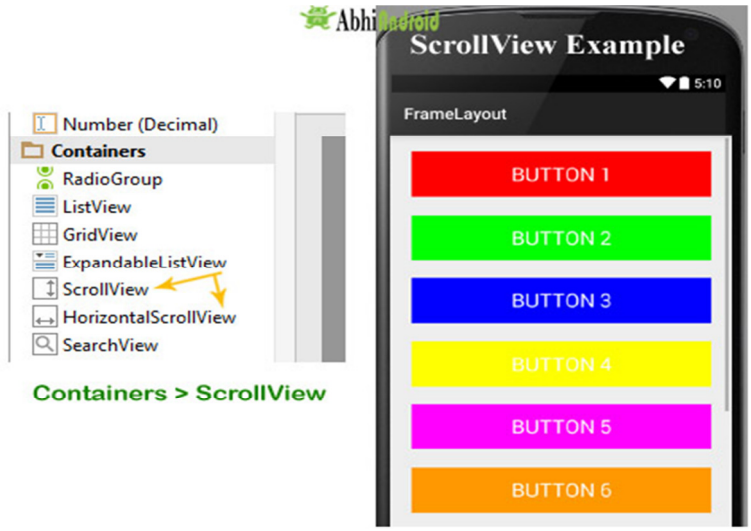
Now start AVD in Emulator and run the App. You will see the images of Lion and Monkey displayed on screen. Click on any Animal image and his name will appear on Screen. We clicked on Lion.



ScrollView And Horizontal ScrollView In Android

In android ScrollView can hold only one direct child. This means that, if you have complex layout with more views(Buttons, TextViews or any other view) then you must enclose them inside another standard layout like Table Layout, Relative Layout or Linear Layout. You can specify layout_width and layout_height to adjust width and height of screen. You can specify height and width in dp(density pixel) or px(pixel). Then after enclosing them in a standard layout, enclose the whole layout in ScrollView to make all the element or views scrollable.

ScrollView in Android Studio Design: It is present inside Containers >> ScrollView or HorizontalScrollView



Important Note 1: We never use a Scroll View with a ListView because List View is default scrollable(i.e. vertical scrollable). More importantly, doing this affects all of the important optimizations in a List View for dealing with large

lists(list items). Just because it effectively forces the List View to display its entire list of items to fill up the infinite container supplied by a ScrollView so we don't use it with List View.

Important Note 2: In android default ScrollView is used to scroll the items in vertical direction and if you want to scroll the items horizontally then you need to implement horizontal ScrollView.

ScrollView Syntax:

```
<ScrollView
android:id="@+id/scrollView"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<!-- add child view's here -->
</ScrollView>
```

Horizontal ScrollView:

In android, You can scroll the elements or views in both vertical and horizontal directions. To scroll in Vertical we simply use ScrollView as we shown in the previous code of this article and to scroll in horizontal direction we need to use HorizontalScrollview.

Below is HorizontalScrollView syntax in Android is:

```
<HorizontalScrollView
android:id="@+id/horizontalscrollView"
android:layout_width="fill_parent"
android:layout_height="fill_parent">

<-- add child view's here -->

</HorizontalScrollView >
```

Attributes Of Scroll View:

ScrollView and HorizontalScrollView has same attributes, the only difference is scrollView scroll the child items in vertical direction while horizontal scroll view scroll the child items in horizontal direction.

Now let's we discuss about the attributes that helps us to configure a ScrollView and its child controls. Some of the most important attributes you will use with ScrollView include:

1. id: In android, id attribute is used to uniquely identify a ScrollView.

Below is id attribute's example code with explanation included.

```
<ScrollView
android:id="@+id/scrollView"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
/>
```

2. scrollbars: In android, scrollbars attribute is used to show the scrollbars in horizontal or vertical direction. The possible Value of scrollbars is vertical, horizontal or none. By default scrollbars is shown in vertical direction in scrollView and in horizontal direction in HorizontalScrollView.

Below is scrollbars attribute's example code in which we set the scrollbars in vertical direction.

```
< HorizontalScrollView
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:scrollbars="vertical"/><!--scrollbars in vertical direction-->
```

Example of ScrollView In Android Studio:

Example 1: In this example we will use 10 button and scroll them using ScrollView in vertical direction. Below is the code and final Output we will create:

Step 1: Create a new project in Android Studio and name it scrollviewExample.

Select File -> New -> New Project -> Android Application Project (or) Android Project. Fill the forms and click "Finish" button.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add below code. Here we are creating a Relative Layout having 10 buttons which are nested in Linear Layout and then in ScrollView.

Important Note: Remember ScrollView can hold only one direct child. So we have to jointly put 10 buttons inside Linear Layout to make it one child. And then we put it inside ScrollView.

```
<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">
```

```
<ScrollView
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:scrollbars="vertical">

<LinearLayout
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:layout_margin="20dp"
android:orientation="vertical">

<Button
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:background="#f00"
android:text="Button 1"
android:textColor="#fff"
android:textSize="20sp" />

<Button
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="20dp"
android:background="#0f0"
android:text="Button 2"
android:textColor="#fff"
android:textSize="20sp" />

<Button
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="20dp"
android:background="#00f"
```

```
android:text="Button 3"
android:textColor="#fff"
android:textSize="20sp" />

<Button
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="20dp"
android:background="#ff0"
android:text="Button 4"
android:textColor="#fff"
android:textSize="20sp" />

<Button
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="20dp"
android:background="#f0f"
android:text="Button 5"
android:textColor="#fff"
android:textSize="20sp" />

<Button
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="20dp"
android:background="#f90"
android:text="Button 6"
android:textColor="#fff"
android:textSize="20sp" />

<Button
android:layout_width="fill_parent"
android:layout_height="wrap_content"
```

```
android:layout_gravity="center"
android:layout_marginTop="20dp"
android:background="#f00"
android:text="Button 7"
android:textColor="#ff9"
android:textSize="20sp" />

<Button
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="20dp"
android:background="#444"
android:text="Button 8"
android:textColor="#fff"
android:textSize="20sp" />

<Button
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="20dp"
android:background="#ff002211"
android:text="Button 9"
android:textColor="#fff"
android:textSize="20sp" />

<Button
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="20dp"
android:background="#0f0"
android:text="Button 10"
android:textColor="#fff"
android:textSize="20sp" />
```

```
</LinearLayout>

</ScrollView>

</RelativeLayout>
```

Step 3: Now Open src -> package -> MainActivity.java and paste the below code

```
package com.example.gourav.scrollviewExample;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Step 4: Now open manifest.xml and paste the below code

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

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

```
</intent-filter>
</activity>
</application>


</manifest>
```

Step 5: Lastly open res ->values -> strings.xml and paste the below code

```
<resources>
<string name="app_name">ScrollViewExample</string>
<string name="hello_world">Hello world!</string>
<string name="action_settings">Settings</string>
</resources>
```

Output:

Now run the App in Emulator / AVD or in real device. You will see the 10 buttons which can be scrollable in vertical direction.

Horizontal ScrollView

Example 2: In this example we will scroll the buttons in horizontal direction. Below is the complete code and final output:

Step 1: Create a new project and name it horizontalscrollviewExample.

Select File -> New -> New Project and Fill the forms and click "Finish" button.

Step 2: Now open res -> layout -> activity_mail.xml (or) main.xml and add below code. Here we are creating same buttons in HorizontalScrollView.

```
<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">

<HorizontalScrollView
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:scrollbars="horizontal">

   <LinearLayout
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="horizontal">

      <Button
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_gravity="center"
         android:layout_margin="20dp"
         android:background="#f00"
         android:padding="10dp"
         android:text="Button 1"
         android:textColor="#fff"
         android:textSize="20sp" />

      <Button
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_gravity="center"
         android:layout_margin="20dp"
         android:background="#0f0"
         android:padding="10dp"
         android:text="Button 2"
         android:textColor="#fff"
         android:textSize="20sp" />

      <Button
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_margin="20dp"
    android:background="#00f"
    android:padding="10dp"
    android:text="Button 3"
    android:textColor="#fff"
    android:textSize="20sp" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_margin="20dp"
    android:background="#ff0"
    android:padding="10dp"
    android:text="Button 4"
    android:textColor="#fff"
    android:textSize="20sp" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_margin="20dp"
    android:background="#f0f"
    android:padding="10dp"
    android:text="Button 5"
    android:textColor="#fff"
    android:textSize="20sp" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_margin="20dp"
```

```
    android:background="#f90"
    android:padding="10dp"
    android:text="Button 6"
    android:textColor="#fff"
    android:textSize="20sp" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_margin="20dp"
    android:background="#f00"
    android:padding="10dp"
    android:text="Button 7"
    android:textColor="#ff9"
    android:textSize="20sp" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_margin="20dp"
    android:background="#444"
    android:padding="10dp"
    android:text="Button 8"
    android:textColor="#fff"
    android:textSize="20sp" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_margin="20dp"
    android:background="#ff002211"
    android:padding="10dp"
    android:text="Button 9"
    android:textColor="#fff"
```

```
            android:textSize="20sp" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_margin="20dp"
            android:background="#0f0"
            android:padding="10dp"
            android:text="Button 10"
            android:textColor="#fff"
            android:textSize="20sp" />

    </LinearLayout>

  </HorizontalScrollView>

</RelativeLayout>
```

Step 3: Now open app -> java -> MainActivity.java in package and paste the below code:

```
package com.example.gourav.horizontalscrollviewExample;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Step 5: Now open AndroidManifest.xml inside manifests and paste the below code:

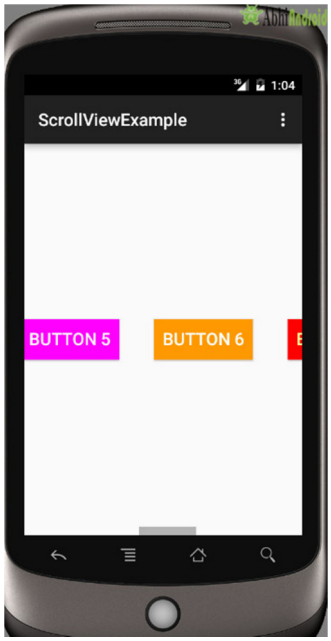
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.gourav.horizontalscrollviewExample" >

   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name="com.example.gourav.horizontalscrollviewExample.MainActivity"
         android:label="@string/app_name" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Step 6: Open res ->values -> strings.xml and paste the below code:

```
<resources>
   <string name="app_name">ScrollViewExample</string>
   <string name="hello_world">Hello world!</string>
   <string name="action_settings">Settings</string>
</resources>
```

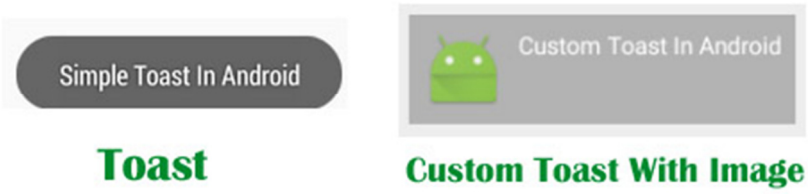
Output:

Now run the App in Emulator /AVD or real device and you will see the 10 buttons can now be scrollable in Horizontal direction.



Toast & Custom Toast In Android Studio

In Android, Toast is used to display information for a period of time. It contains a message to be displayed quickly and disappears after specified period of time. It does not block the user interaction. Toast is a subclass of Object class. In this we use two constants for setting the duration for the Toast. Toast notification in android always appears near the bottom of the screen. We can also create our custom toast by using custom layout(xml file).



Special Note: In Android, Toast is used when we required to notify user about an operation without expecting any user input. It displays a small popup for message and automatically fades out after timeout.

Important Methods Of Toast:

Let's we discuss some important methods of Toast that may be called in order to manage the Toast.

1. makeText(Context context, CharSequence text, int duration): This method is used to initiate the Toast. This method take three parameters First is for the application Context, Second is text message and last one is duration for the Toast.

Constants of Toast: Below is the constants of Toast that are used for setting the duration for the Toast.

1. LENGTH_LONG: It is used to display the Toast for a long period of time. When we set this duration the Toast will be displayed for a long duration.

2. LENGTH_SHORT: It is used to display the Toast for short period of time. When we set this duration the Toast will be displayed for short duration.

Below we show the use of makeText() method of Toast in which we set application context, a text message and duration for the Toast.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast", Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for the Toast
```

2. show(): This method is used to display the Toast on the screen. This method is display the text which we create using makeText() method of Toast.

Below we Firstly initiate the Toast and then display it using show() method.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android", Toast.LENGTH_LONG);
// initiate the Toast with context, message and duration for the Toast

toast.show(); // display the Toast
```

3. setGravity(int,int,int): This method is used to set the gravity for the Toast. This method accepts three parameters: a Gravity constant, an x-position offset, and a y-position offset.

Below we Firstly initiate the Toast, set top and left gravity and then display it using show() method.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android", Toast.LENGTH_LONG);
// initiate the Toast with context, message and duration for the Toast

toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0);    // set gravity for the Toast.

toast.show(); // display the Toast
```

4. setText(CharSequence s): This method is used to set the text for the Toast. If we use makeText() method and then we want to change the text value for the Toast then we use this method.

Below we firstly create a new Toast using makeText() method and then set the text for the Toast.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android", Toast.LENGTH_LONG);
// initiate the Toast with context, message and duration for the Toast

toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0);    // set gravity for the Toast.

toast.setText("Changed Toast Text"); // set the text for the Toast

toast.show(); // display the Toast
```

5. setDuration(int duration): This method is used to set the duration for the Toast. If we use makeText() method and then we want to change the duration for the Toast then we use this method.

Below we firstly create a new Toast using makeText() method and then set the duration for the Toast.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android", Toast.LENGTH_LONG);
// initiate the Toast with context, message and duration for the Toast

toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0);     // set gravity for the Toast.

toast.setDuration(Toast.LENGTH_SHORT); // set the duration for the Toast.

toast.show(); // display the Toast
```

6. inflate(int, ViewGroup): This method is used to inflate the layout from the xml. In this method first parameter is the layout resource ID and the second is the root View.

Below we retrieve the Layout Inflater and then inflate the layout from the xml file.

```
// Retrieve the Layout Inflater and inflate the layout from xml

LayoutInflater inflater = getLayoutInflater();

View layout = inflater.inflate(R.layout.custom_toast_layout,

(ViewGroup) findViewById(R.id.toast_layout_root));
```

7. setView(View): This method is used to set the view for the Toast. In this method we pass the inflated layout which we inflate using inflate() method.

Below we firstly retrieve the layout inflater and then inflate the layout and finally create a new Toast and pass the inflated layout in the setView() method.

```
// Retrieve the Layout Inflater and inflate the layout from xml

LayoutInflater inflater = getLayoutInflater();

View layout = inflater.inflate(R.layout.custom_toast_layout,

(ViewGroup) findViewById(R.id.toast_layout_root));


// create a new Toast using context

Toast toast = new Toast(getApplicationContext());

toast.setDuration(Toast.LENGTH_LONG); // set the duration for the Toast

toast.setView(layout); // set the inflated layout

toast.show(); // display the custom Toast
```

Custom Toast in Android:

In Android, Sometimes simple Toast may not be satisfactory, and then we can go for customizing a Toast. For creating a custom layout, define a View layout, in XML and pass the root View object to the setView(View) method.

Steps for Implementation of Custom Toast In Android:

Step1: Firstly Retrieve the Layout Inflater with getLayoutInflater () (or getSystemService ()) and then inflate the layout from XML using inflate (int, ViewGroup). In inflate method first parameter is the layout resource ID and the second is the root View.

Step 2: Create a new Toast with Toast(Context) and set some properties of the Toast, such as the duration and gravity.

Step 3: Call setView(View) and pass the inflated layout in this method.

Step 4: Display the Toast on the screen using show() method of Toast.

In the below example we have shown the functioning of Toast and custom Toast both.

Toast And Custom Toast Example In Android Studio:

Below is the example of Toast and Custom Toast in Android. In this example we display two Button's one for Simple Toast and other for Custom Toast and perform click event on them. Whenever a user click on simple Toast Button a Toast with message "Simple Toast In Android" displayed on the screen and when a user clicks on custom toast Button a message "Custom Toast In Android" with a image displayed on the screen. For Creating a custom toast we firstly retrieve the layout inflater and then inflate the custom toast layout from the xml file. After that we get the reference of TextView and ImageView from the inflated layout and set the text and image in the TextView and ImageView. Finally we create a new Toast and pass the inflated layout in the setView() method and then display the Toast by using show() method of Toast.

Step 1: Create a new project and name it ToastExample

Step 2: Open res -> layout ->activity_main.xml (or) main.xml and add following code:

```
<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">
<!-- Button's for simple and custom Toast -->
<Button
android:id="@+id/simpleToast"
android:layout_width="200dp"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:layout_marginTop="150dp"
android:background="#f00"
android:text="Simple Toast"
android:textColor="#fff"
android:textSize="20sp" />

<Button
android:id="@+id/customToast"
android:layout_width="200dp"
android:layout_height="wrap_content"
android:layout_below="@+id/simpleToast"
android:layout_centerHorizontal="true"
android:layout_margin="50dp"
android:background="#0f0"
android:text="Custom Toast"
android:textColor="#fff"
android:textSize="20sp" />

</RelativeLayout>
```

Step 3: Now create a xml layouts by right clicking on res/layout -> New -> Layout Resource File and name it custom_toast_layout.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/toast_layout_root"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#DAAA"
android:orientation="horizontal"
android:padding="8dp">
```

```
<!-- ImageVView and TextView for custom Toast -->
<ImageView
android:id="@+id/toastImageView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginRight="8dp" />

<TextView
android:id="@+id/toastTextView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textColor="#FFF" />
</LinearLayout>
```

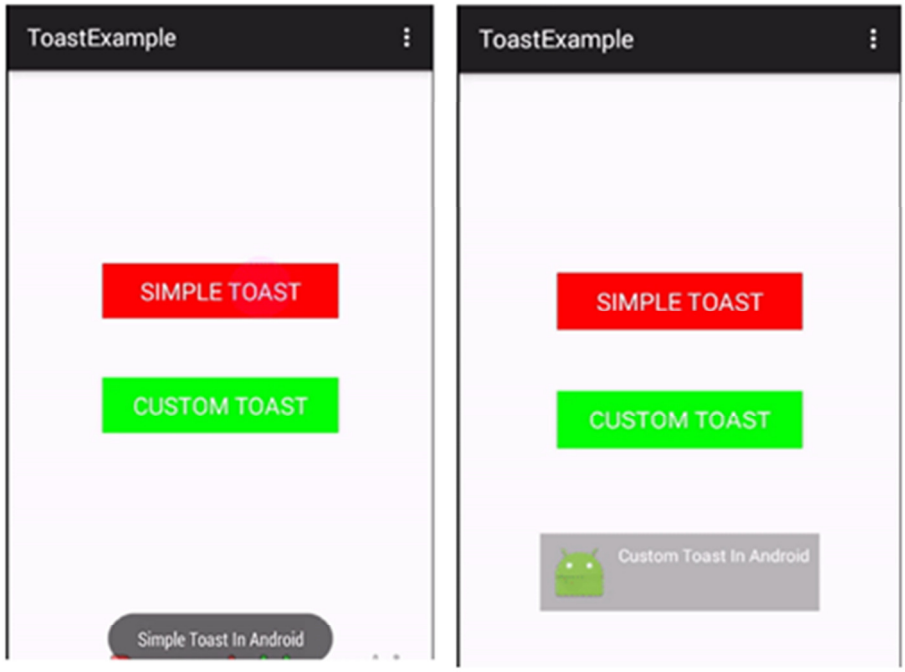
Step 4: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code for initiate the Button's and perform click event on Button's. Whenever a user click on simple Toast Button a Toast with message "Simple Toast In Android" displayed on the screen and when a user clicks on custom toast Button a message "Custom Toast In Android" with a image displayed on the screen. For Creating a custom toast we firstly retrieve the layout inflater and then inflate the custom toast layout from the xml file. After that we get the reference of TextView and ImageView from the inflated layout and set the text and image in the TextView and ImageView. Finally we create a new Toast and pass the inflated layout in the setView() method and then display the Toast by using show() method of Toast.

```
package com.abhiandroid.toastexample;


import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.Button;
import android.view.ViewGroup;


public class MainActivity extends AppCompatActivity {
```

```
Button simpleToast, customToast;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // get the reference of Button's
    simpleToast = (Button) findViewById(R.id.simpleToast);
    customToast = (Button) findViewById(R.id.customToast);
    // perform setOnClickListener event on simple Toast Button
    simpleToast.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // initiate a Toast with message and duration
            Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android", Toast.LENGTH
_LONG); // initiate the Toast with context, message and duration for the Toast
            toast.setGravity(Gravity.BOTTOM | Gravity.CENTER_HORIZONTAL, 0, 0);     // set gravity for the
Toast.
            toast.show(); // display the Toast

        }
    });
    // perform setOnClickListener event on custom Toast Button
    customToast.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // Retrieve the Layout Inflater and inflate the layout from xml
            LayoutInflater inflater = getLayoutInflater();
            View layout = inflater.inflate(R.layout.custom_toast_layout,
                  (ViewGroup) findViewById(R.id.toast_layout_root));
            // get the reference of TextView and ImageVIew from inflated layout
            TextView toastTextView = (TextView) layout.findViewById(R.id.toastTextView);
            ImageView toastImageView = (ImageView) layout.findViewById(R.id.toastImageView);
            // set the text in the TextView
            toastTextView.setText("Custom Toast In Android");
            // set the Image in the ImageView
            toastImageView.setImageResource(R.drawable.ic_launcher);
```

```
            // create a new Toast using context
            Toast toast = new Toast(getApplicationContext());
            toast.setDuration(Toast.LENGTH_LONG); // set the duration for the Toast
            toast.setView(layout); // set the inflated layout
            toast.show(); // display the custom Toast
        }
    });
  }
}
```



Simple Toast Alert Custom Toast Alert

TimePicker Tutorial In Android Studio

In Android, TimePicker is a widget used for selecting the time of the day in either AM/PM mode or 24 hours mode. The displayed time consist of hours, minutes and clock format. If we need to show this view as a Dialog then we have to use a TimePickerDialog class.

TimePicker code:

```
<TimePicker
android:id="@+id/simpleTimePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:timePickerMode="spinner"/>
```

Methods of TimePicker:

Let's discuss some common methods of a time picker, which are used to configure a time picker in our application.

1. setCurrentHour(Integer currentHour):

This method is used to set the current hours in a time picker.

setHour(Integer hour): *setCurrentHour() method was deprecated in API level 23. From api level 23 we have to use* ***setHour(Integer hour)***. In this method there is only one parameter of integer type which is used to set the value for hours.

Below we set the 5 value for the current hours.

```
TimePicker simpleTimePicker=(TimePicker)findViewById(R.id.simpleTimePicker); // initiate a time pick
er
// set the value for current hours
simpleTimePicker.setCurrentHour(5); // before api level 23
simpleTimePicker.setHour(5);
```

2. setCurrentMinute(Integer currentMinute):

This method is used to set the current minutes in a time picker.

setMinute(Integer minute): *setCurrentMinute() method was deprecated in API level 23. From api level 23 we have to use* ***setMinute(Integer minute)***. In this method there is only one parameter of integer type which set the value for minutes.

Below we set the 35 value for the current minutes.

```
TimePicker simpleTimePicker=(TimePicker)findViewById(R.id.simpleTimePicker); // initiate a time pick
er

// set the value for current hours

simpleTimePicker.setCurrentMinute(35); // before api level 23

simpleTimePicker.setMinute(35); // from api level 23
```

3. getCurrentHour():

This method is used to get the current hours from a time picker.

getCurrentHour(): *getCurrentHour() method was deprecated in API level 23. From api level 23 you have to use* ***getHour().*** This method returns an integer value.

Below we get the value of hours from a timepicker set by user.

```
TimePicker simpleTimePicker = (TimePicker)findViewById(R.id.simpleTimePicker); // initiate a time

pickerint hours =simpleTimePicker.getCurrentHour(); // before api level 23

int hours =simpleTimePicker.getHour(); // after api level 23
```

4. getCurrentMinute():

This method is used to get the current minutes from a time picker.

getMinute(): *getCurrentMinute() method was deprecated in API level 23. From api level 23 we have to use* ***getMinute().*** This method returns an integer value.

Below we get the value of minutes from a time picker.

```
TimePicker simpleTimePicker = (TimePicker)findViewById(R.id.simpleTimePicker); // initiate a time pic
ker

int minutes = simpleTimePicker.getCurrentMinute(); // before api level 23

int minutes = simpleTimePicker.getMinute(); // after api level 23
```

5. setIs24HourView(Boolean is24HourView):

This method is used to set the mode of the Time picker either 24 hour mode or AM/PM mode. In this method we set a Boolean value either true or false. True value indicate 24 hour mode and false value indicate AM/PM mode.

Below we set the current mode of the time picker.

```
TimePicker simpleTimePicker = (TimePicker)findViewById(R.id.simpleTimePicker); // initiate a time pic
ker

simpleTimePicker.setIs24HourView(true);
```

6. is24HourView():

This method is used to check the current mode of the time picker. This method returns true if its 24 hour mode or false if AM/PM mode is set.

Below we get the current mode of the time picker:

```
TimePicker simpleTimePicker = (TimePicker)findViewById(R.id.simpleTimePicker); // initiate a time pic
ker

Boolean mode=simpleTimePicker.is24HourView(); // check the current mode of the time picker
```

7.setOnTimeChangedListener(TimePicker.OnTimeChangedListener onTimeChangedListener):

This method is used to set the callback that indicates the time has been adjusted by the user. onTimeChanged(TimePicker view, int hourOfDay, int minute) is an override function of this listener in which we have three parameters first is for TimePicker, second for getting hour of the day and last is for getting the minutes after changing the time of the time picker.

Below we show the use of on time changed listener of a time picker.

```
TimePicker simpleTimePicker = (TimePicker)findViewById(R.id.simpleTimePicker); // initiate a time pick
er


simpleTimePicker.setOnTimeChangedListener(new TimePicker.OnTimeChangedListener() {
@Override
public void onTimeChanged(TimePicker view, int hourOfDay, int minute) {


}
});
```

Attributes of TimePicker:

Now let's we discuss about the attributes that helps us to configure a time picker in your xml file (layout).

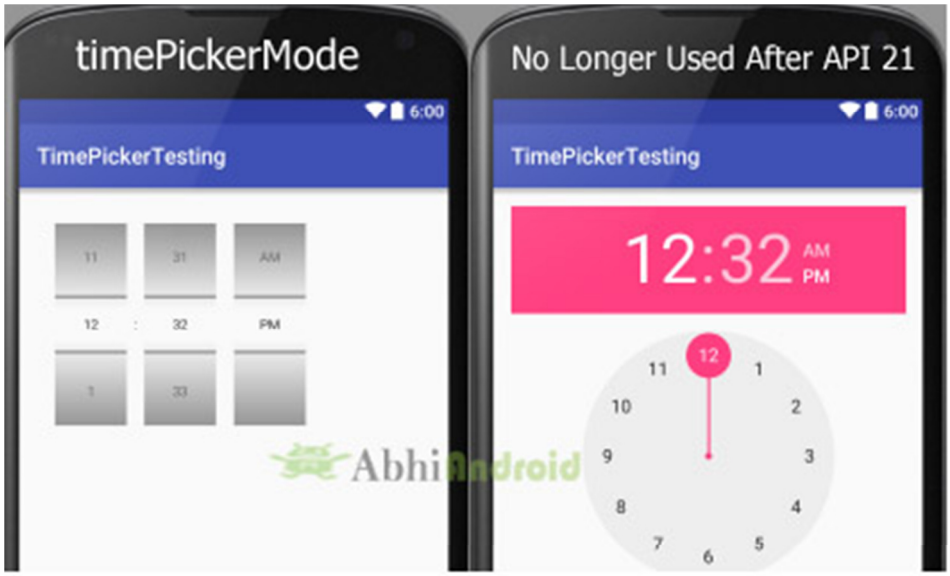
1. id: id is an attribute used to uniquely identify a time picker.

```
<TimePicker
android:id="@+id/simpleTimePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/> <!-- id of a time picker -->
```

2. timePickerMode: time picker mode is an attribute of time picker used to set the mode either spinner or clock. Default mode is clock but this mode is no longer used after api level 21, so from api level 21 you have to set the mode to spinner.

Below we set the mode to spinner.

```
<TimePicker
android:id="@+id/simpleTimePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:timePickerMode="spinner" />
```



3. background: background attribute is used to set the background of a time picker. We can set a color or a drawable image in the background. We can also set the background color programmatically means in java class.

Below we set the orange color for the background of a time picker.

```
<TimePicker
   android:id="@+id/simpleTimePicker"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:timePickerMode="spinner"
   android:background="#F88F00" />
```

Setting TimePicker Background In Java Class:

```
TimePicker simpleTimePicker=(TimePicker)findViewById(R.id.simpleTimePicker); //initiate a time pick
er

simpleTimePicker.setBackgroundColor(Color.YELLOW); //Yellow background color for the background o
f a time picker
```

4. padding: padding attribute is used to set the padding from left, right, top or bottom for a time picker.

- **paddingRight:** set the padding from the right side of the time picker.
- **paddingLeft:** set the padding from the left side of the time picker.
- **paddingTop:** set the padding from the top side of the time picker.
- **paddingBottom:** set the padding from the bottom side of the time picker.
- **Padding:** set the padding from the all side's of the time picker.

Below example we set the 20dp padding from all the side's of the time picker.

```
<TimePicker

android:id="@+id/simpleTimePicker"

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:timePickerMode="spinner"

android:layout_centerHorizontal="true"

android:layout_marginTop="50dp"

android:padding="20dp"/>
```

Example of TimePicker in Android Studio:

Example 1: In the below example of time picker we will show you the use of time picker in our application. For that we display simple time picker and

a textview in our xml file and perform setOnTimeChangedListener() event, so that whenever a user adjust the time the current displayed time of time picker is displayed by using a Toast and also displayed in the textview.

Step 1: Create a new project and name it **TimePickerExample**

Step 2: Open res -> layout -> **activity_main.xml (or) main.xml** and add following code:

In this step we open an xml file and add the code for displaying a time picker with spinner mode and textview for displaying time of time picker.

```
<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">

   <TimePicker
      android:id="@+id/simpleTimePicker"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="50dp"
      android:background="#090"
      android:padding="20dp"
      android:timePickerMode="spinner" />

   <TextView
      android:id="@+id/time"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:text="Time Is ::"
      android:textColor="#090"
      android:textSize="20sp"
      android:textStyle="bold" />
```

```
</RelativeLayout>
```

Step 3: Open app -> package -> **MainActivity.java**

In this step we open MainActivity where we add the code to initiate the time picker and a text view to display time of time picker and then we perform setOnTimeChangedListener() event so whenever a user adjust the time the current displayed time of time picker is displayed by using a Toast and also displayed in the textview.

```
package example.gb.timepickerexample;


import android.app.TimePickerDialog;
import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;
import android.widget.TimePicker;
import android.widget.Toast;


import org.w3c.dom.Text;


import java.util.Calendar;


public class MainActivity extends AppCompatActivity {


   TextView time;
   TimePicker simpleTimePicker;


   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      //  initiate the view's
      time = (TextView) findViewById(R.id.time);
      simpleTimePicker = (TimePicker) findViewById(R.id.simpleTimePicker);
```

```
    simpleTimePicker.setIs24HourView(false); // used to display AM/PM mode
    // perform set on time changed listener event
    simpleTimePicker.setOnTimeChangedListener(new TimePicker.OnTimeChangedListener() {
      @Override
      public void onTimeChanged(TimePicker view, int hourOfDay, int minute) {
        // display a toast with changed values of time picker
        Toast.makeText(getApplicationContext(), hourOfDay + "  " + minute, Toast.LENGTH_SHORT).sho
w();
        time.setText("Time is :: " + hourOfDay + " : " + minute); // set the current time in text view
      }
    });
  }


}
```

Output:

Now run the App in AVD and you will see TimePicker on the screen. Change the time and it will be displayed as Toast and also in TextView.

Example 2: In the second example of TimePicker we will show the use of time picker dialog in our application. To do that we will display edittext in our xml file and perform a click listener event on it, so whenever a user click on it time picker

dialog will appear and from there user can adjust the time and after selecting the time it will be displayed in the edittext.

Step 1: Create a new project and name it **TimePickerExample**

Step 2: Open res -> layout -> **activity_main.xml (or) main.xml** and add following code:

In this step we open an xml file and add the code for displaying a edittext to display time of time picker.

```
<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">

   <EditText
      android:id="@+id/time"
      android:layout_width="200dp"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:hint="Select Time..."
      android:textColor="#090"
      android:textColorHint="#090"
      android:background="#d4d4d4"
      android:padding="15dp"
      android:textSize="20sp"
      android:textStyle="bold" />

</RelativeLayout>
```

Step 3: Open src -> package -> **MainAcivity.java**

In this step we open MainActivity where we add the code to initiate the edittext to display time of time picker and perform click event on edittext, so whenever a user clicks on edittext a time picker dialog will appear from there user can set the time. And finally then the time will be displayed in the edit text.

```
package example.gb.timepickerexample;

import android.app.TimePickerDialog;
import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.TimePicker;
import android.widget.Toast;

import org.w3c.dom.Text;

import java.util.Calendar;

public class MainActivity extends AppCompatActivity {

  EditText time;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //  initiate the edit text
    time = (EditText) findViewById(R.id.time);
    // perform click event listener on edit text
    time.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Calendar mcurrentTime = Calendar.getInstance();
        int hour = mcurrentTime.get(Calendar.HOUR_OF_DAY);
        int minute = mcurrentTime.get(Calendar.MINUTE);
        TimePickerDialog mTimePicker;
```

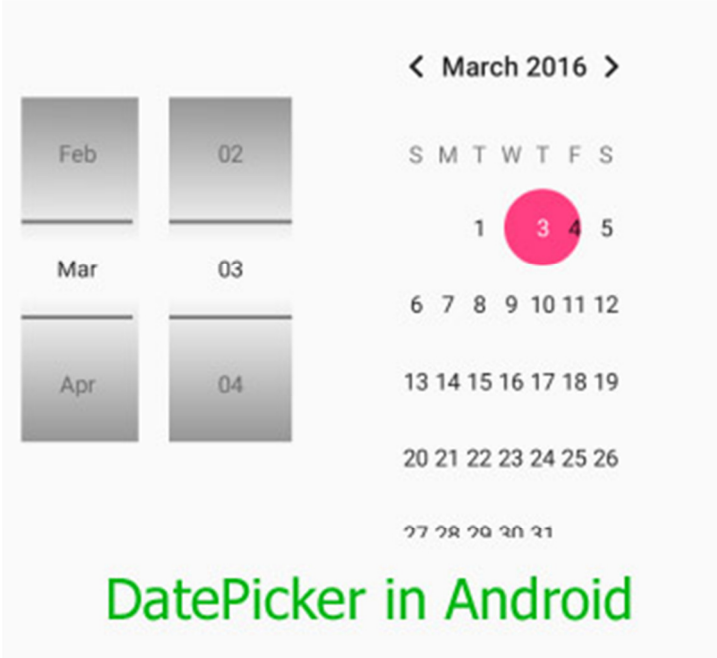
```
        mTimePicker = new TimePickerDialog(MainActivity.this, new TimePickerDialog.OnTimeSetListe
ner() {
                @Override
                public void onTimeSet(TimePicker timePicker, int selectedHour, int selectedMinute) {
                  time.setText(selectedHour + ":" + selectedMinute);
            }
        }, hour, minute, true);//Yes 24 hour time
        mTimePicker.setTitle("Select Time");
        mTimePicker.show();


      }
    });
  }


}
```

Output:

Now run the App in AVD and you will see edittext asking user to select time. When user click on it, timepicker dialog will open from there user can select time. And then this time will be displayed in EditText.

DatePicker In Android Studio

In Android, DatePicker is a widget used to select a date. It allows to select date by day, month and year in your custom UI (user interface). If we need to show this view as a dialog then we have to use a DatePickerDialog class. For selecting time Android also provides timepicker to select time.



DatePicker code in XML:

```
<DatePicker
android:id="@+id/simpleDatePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:datePickerMode="spinner"/>
```

Methods of DatePicker

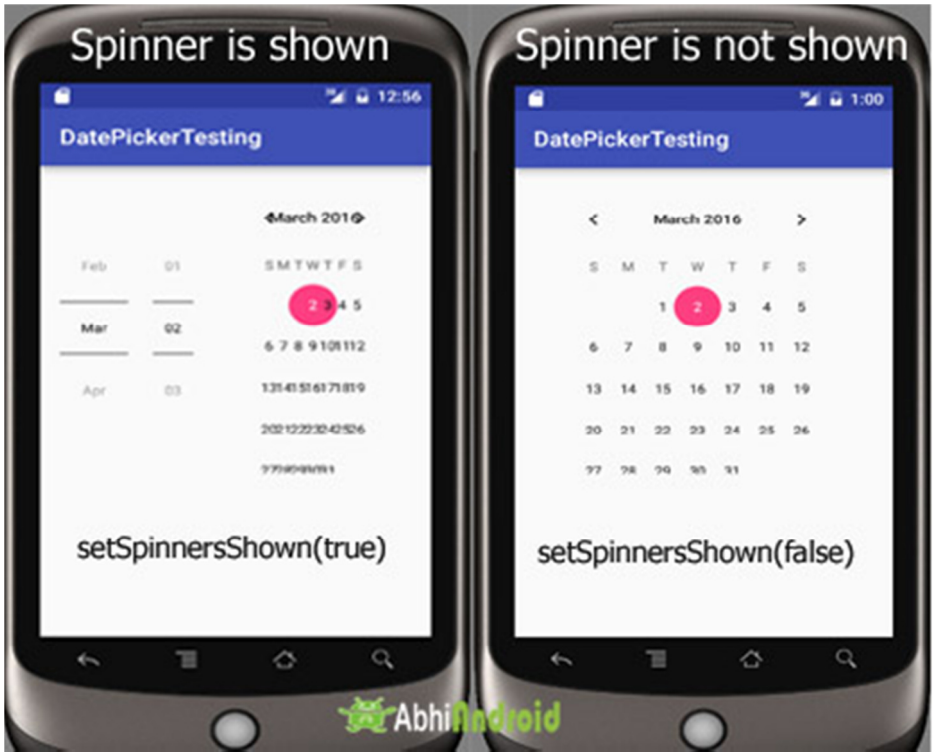
Let's discuss some common methods of a datepicker which are used to configure a DatePicker in our application.

1. setSpinnersShown(boolean shown):

This method is used to set whether the spinner of the date picker in shown or not. In this method you have to set a Boolean value either true or false. True indicates spinner is shown, false value indicates spinner is not shown. Default value for this function is true.

Below we show the use of setSpinnerShown() function by setting false value.

```
DatePicker simpleDatePicker = (DatePicker)findViewById(R.id.simpleDatePicker);
simpleDatePicker.setSpinnersShown(false);
```



2. getDayOfMonth():

This method is used to get the selected day of the month from a date picker. This method returns an integer value.

Below we get the selected day of the month from a date picker.

```
/*Add in Oncreate() funtion after setContentView()*/
DatePicker simpleDatePicker = (DatePicker) findViewById(R.id.simpleDatePicker); // initiate a date pick
er
int day = simpleDatePicker.getDayOfMonth(); // get the selected day of the month
```

3. getMonth():

This method is used to get the selected month from a date picker. This method returns an integer value.

Below we get the selected month from a date picker.

```
DatePicker simpleDatePicker = (DatePicker)findViewById(R.id.simpleDatePicker); // initiate a date picke
r
int month = simpleDatePicker.getMonth(); // get the selected month
```

4. getYear():

This method is used to get the selected year from a date picker. This method returns an integer value.

Below code is used to get the selected year from a date picker.

```
DatePicker simpleDatePicker = (DatePicker)findViewById(R.id.simpleDatePicker); // initiate a date picke
r
```

```
int year = simpleDatePicker.getYear(); // get the selected year
```

5. getFirstDayOfWeek():

This method is used to get the first day of the week. This method returns an integer value.

Below code is used to get the first day of the week.

```
DatePicker simpleDatePicker = (DatePicker)findViewById(R.id.simpleDatePicker); // initiate a date picke
r


int firstDay=simpleDatePicker.getFirstDayOfWeek(); // get the first day of the week
```

Attributes of DatePicker

Now let's we discuss some important attributes that helps us to configure a DatePicker in your XML file (layout).

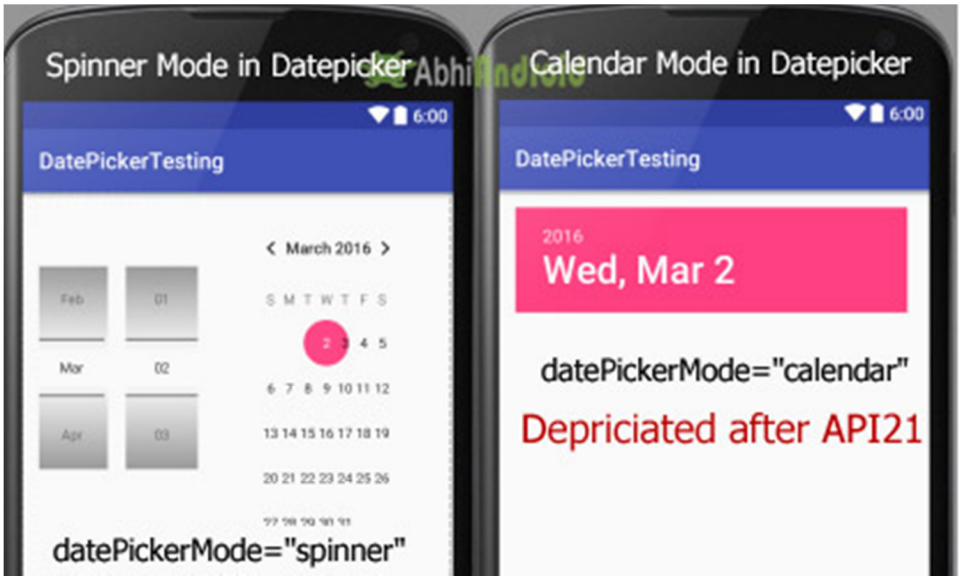
1. id: id is an attribute used to uniquely identify a date picker.

```
<DatePicker
android:id="@+id/simpleDatePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
/>
```

2. datePickerMode: This attribute is used to set the Date Picker in mode either spinner or calendar. Default mode is calendar but this mode is not used after api level 21, so from api level 21 you have to set the mode to spinner.

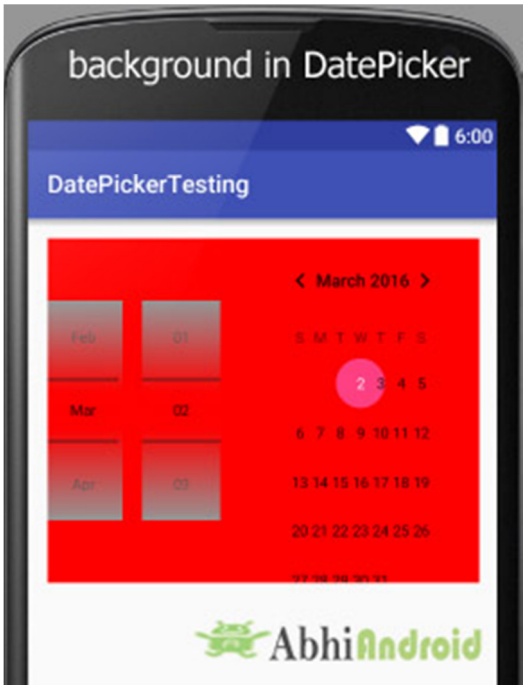
Below is an example code in which we set the mode to spinner for a date picker.

```
<DatePicker
android:id="@+id/simpleDatePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:datePickerMode="spinner" />
```



3. background: background attribute is used to set the background of a date picker. We can set a color or a drawable image in the background.
Below we set the red color for the background of a date picker.

```
<DatePicker
android:id="@+id/simpleDatePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:datePickerMode="spinner"
android:background="#f00"/>
```



Setting background of DatePicker In Java Class:

```
DatePicker simpleDatePicker=(DatePicker)findViewById(R.id.simpleDatePicker); // initiate a date picker
```

```
simpleDatePicker.setBackgroundColor(Color.RED); //  red color for the background of a date picker
```

4. padding: padding attribute is used to set the padding from left, right, top or bottom for a date picker.

- **paddingRight:** set the padding from the right side of the date picker.
- **paddingLeft:** set the padding from the left side of the date picker.
- **paddingTop:** set the padding from the top side of the date picker.
- **paddingBottom:** set the padding from the bottom side of the date picker.
- **Padding:** set the padding from the all side's of the date picker.

Below code of padding attribute set the 40dp padding from all the side's of the date picker.

```
<DatePicker
android:id="@+id/simpleDatePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:datePickerMode="spinner"
android:padding="40dp"/>
```

DatePicker Example in Android Studio:

Example 1: In the first example of DatePicker we show simple date picker and a Button in our xml file and perform click event on button. So whenever a user clicks on a button the day of the month, month and year will be displayed by using a Toast. Below is the final output, download code and step by step explanation:

Step 1: Create a new project and name it **DatePickerExample**

Step 2: Open res -> layout -> **activity_main.xml (or) main.xml** and add following code:

In this step we open an xml file and add the code for displaying a datepicker with spinner mode and a button for getting the date from the datepicker.

```
<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">

  <DatePicker
    android:id="@+id/simpleDatePicker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#150"
    android:datePickerMode="spinner" />

  <Button
    android:id="@+id/submitButton"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    android:layout_below="@+id/simpleDatePicker"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="50dp"
    android:background="#150"
    android:text="SUBMIT"
```

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

Step 3: Open app -> package -> **MainActivity.java**

In this step we open MainActivity where we add the code to initiate the datepicker & a button and then perform onClickListener() event on button so whenever a user clicks on the button the day of the month, month and year will be displayed by using a Toast.

```
package example.abhiandroid.datepickerexample;


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


public class MainActivity extends AppCompatActivity {


  DatePicker simpleDatePicker;
  Button submit;


  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // initiate the date picker and a button
    simpleDatePicker = (DatePicker) findViewById(R.id.simpleDatePicker);
    submit = (Button) findViewById(R.id.submitButton);
    // perform click event on submit button
    submit.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
```

```
            // get the values for day of month , month and year from a date picker
            String day = "Day = " + simpleDatePicker.getDayOfMonth();
            String month = "Month = " + (simpleDatePicker.getMonth() + 1);
            String year = "Year = " + simpleDatePicker.getYear();
            // display the values by using a toast
            Toast.makeText(getApplicationContext(), day + "\n" + month + "\n" + year, Toast.LENGTH_LONG
).show();
        }
    });


  }



}
```

Output:

Now run the App in AVD and you will see datepicker will appear on the screen. Choose the date, month & year and click submit. The date you selected will appear on Screen.

Example 2: In the second example we show the use of date picker dialog in our application, for that we display edittext in our xml file and perform a onClickListener() event on it. So whenever a user click on it date picker dialog is appeared and from there user can adjust the date and after selecting the date it will be displayed in the edit text. Below is the final output, download code and step by step tutorial:

Step 1: Create a new project and name it **DatePickerExample**

Step 2: Open res – > layout – > **activity_main.xml (or) main.xml** and add following code:

In this step we open xml file and add the code for displaying edittext which will be used to display the date of datepicker.

```
<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">

   <EditText
      android:id="@+id/date"
      android:layout_width="200dp"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:background="#d4d4d4"
      android:hint="Select Date..."
      android:padding="15dp"
      android:textColor="#897"
      android:textColorHint="#090"
      android:textSize="20sp"
      android:textStyle="bold" />

</RelativeLayout>
```

Step 3: Open src -> package -> **MainActvity.java**

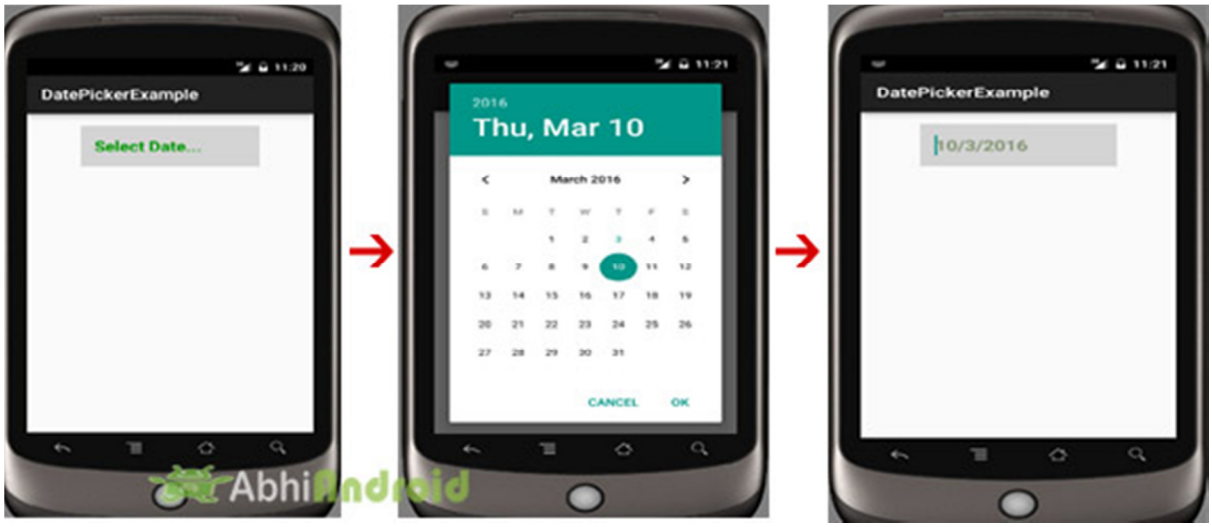
In this step we open MainActivity where we add the code to initiate the edittext to display date(day of month, month and year) from a date picker and perform click event on edit text so whenever a user clicks on edit text a date picker dialog is appeared from there user can set the date by choosing day of month , month and year , after setting the date will be displayed in the edit text.

```
package example.abhiandroid.datepickerexample;

import android.app.DatePickerDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.DatePicker;
import android.widget.EditText;

import java.util.Calendar;

public class MainActivity extends AppCompatActivity {

  EditText date;
  DatePickerDialog datePickerDialog;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // initiate the date picker and a button
    date = (EditText) findViewById(R.id.date);
    // perform click event on edit text
    date.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        // calender class's instance and get current date , month and year from calender
        final Calendar c = Calendar.getInstance();
        int mYear = c.get(Calendar.YEAR); // current year
        int mMonth = c.get(Calendar.MONTH); // current month
        int mDay = c.get(Calendar.DAY_OF_MONTH); // current day
```

```
        // date picker dialog
        datePickerDialog = new DatePickerDialog(MainActivity.this,
                new DatePickerDialog.OnDateSetListener() {


                    @Override
                    public void onDateSet(DatePicker view, int year,  int monthOfYear, int dayOfMonth) {
        date.setText(dayOfMonth + "/"
                            + (monthOfYear + 1) + "/" + year);
                    }
                }, mYear, mMonth, mDay);
        datePickerDialog.show();
      }
    });
  }
}
```

Output:

Now run the App in Emulator and fill the date in EditText option.



Unit-V Activity and Multimedia with Databases

Course Outcome:

Configure Android environment and development tools.

Unit Outcomes:

5a. Apply the given Intents and service in Application development.
5b. Use Fragment to generate the given multiple activities.
5c. Develop programs to play the given multimedia.
5d. Write the query to perform the given database management operation.

Contents:

5.1 Intent, Intent Filter
5.2 Activity Lifecycle; Broadcast lifecycle
5.3 Content Provider; Fragments
5.4 Service: Features Of service, the Android platform service, defining new service, Service Lifecycle, Permission, example of service
5.5 Android System Architecture, Multimedia framework, Play Audio and Video, Text to speech, Sensors, Async tasks
5.6 Audio Capture, Camera
5.7 Bluetooth, Animation
5.8 SQLite Database, necessity of SQLite, Creation and connection of the database, extracting value from cursors, Transactions.

Intent Tutorial in Android And Types

Android uses Intent for communicating between the components of an Application and also from one application to another application.

Intent are the objects which is used in android for passing the information among Activities in an Application and from one app to another also. Intent are used for communicating between the Application components and it also provides the connectivity between two apps.

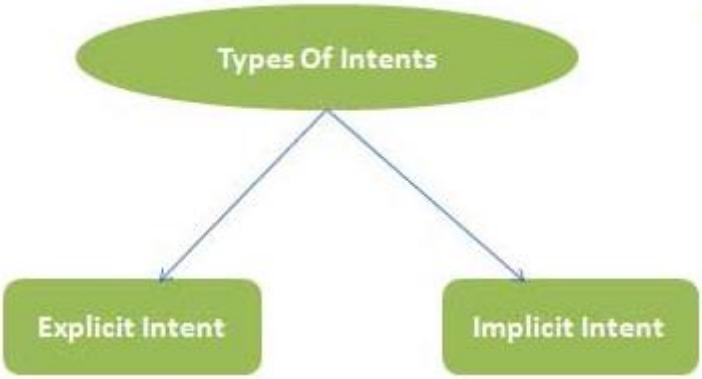
For example: Intent facilitate you to redirect your activity to another activity on occurrence of any event. By calling, startActivity() you can perform this task.

```
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.

Types of Intents:

Intent are of two types: Explicit Intent and Implicit Intent



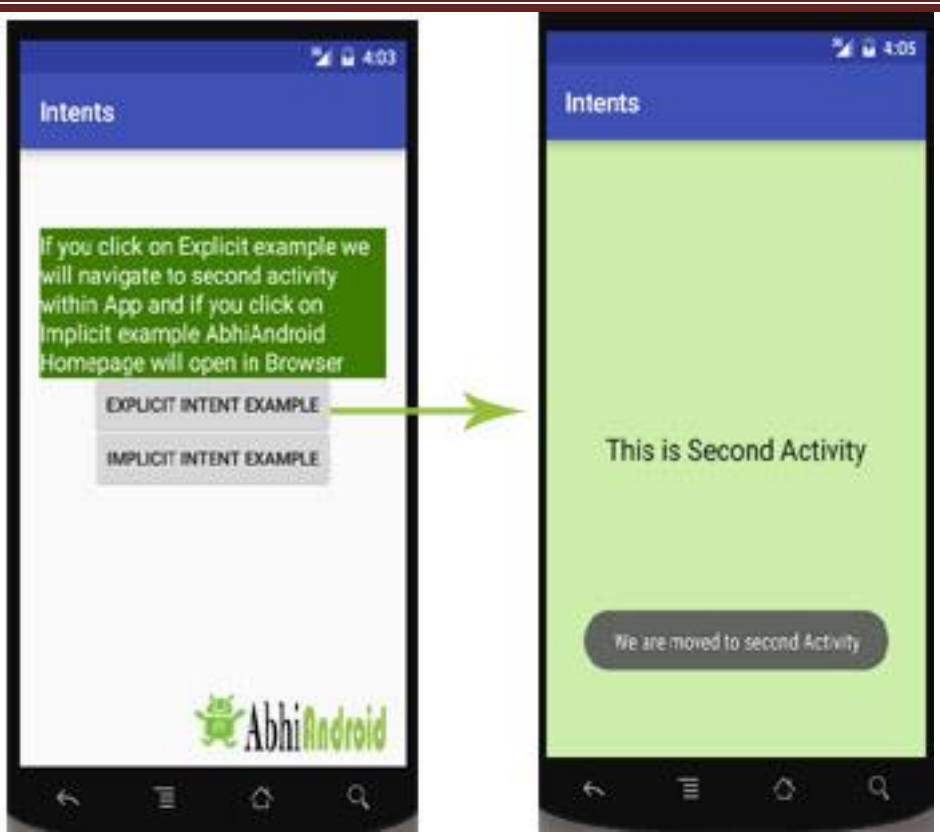
Explicit Intent:

- 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 work 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. Example with code in the end of this post will make it more clear.



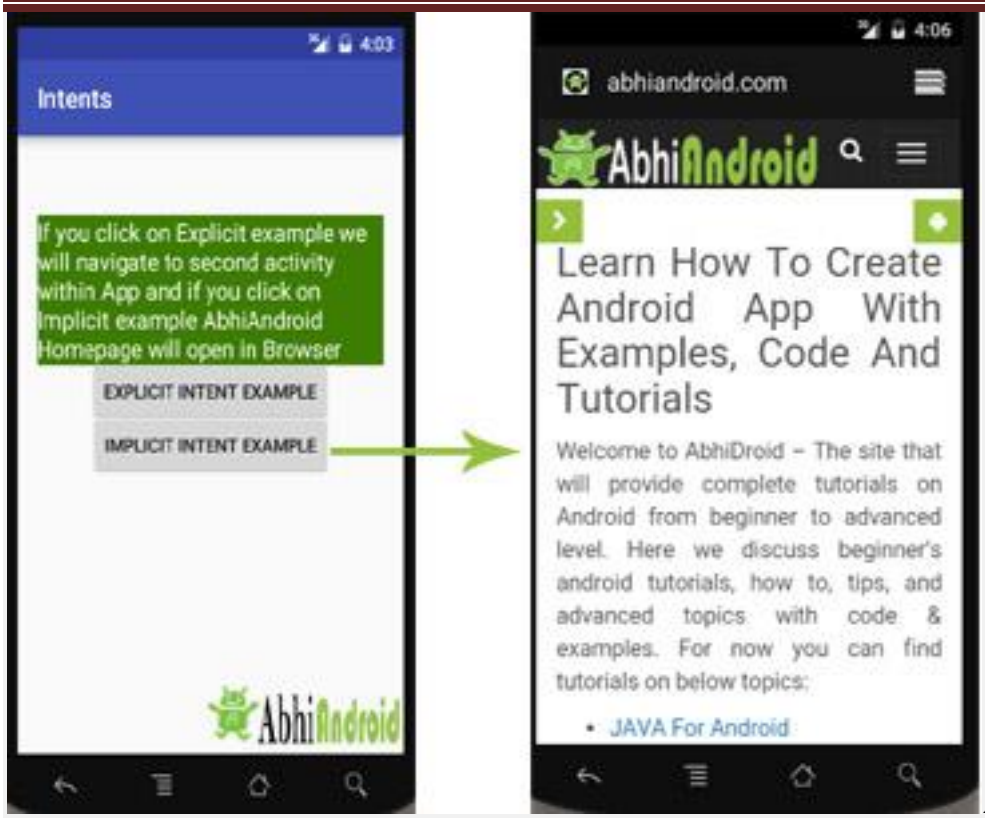
Implicit Intent:

- 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.abhiandroid.com"));
startActivity(intentObj);
```

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



Intent Example In Android:

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 AbhiAndroid using Intent. **The example will show you both implicit and explicit Intent together. Below is the final output:**

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_horizo
ntal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
```

```
android:paddingBottom="@dimen/activity_vertical_margin"  tools:context=".MainActiv
ity">


   <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 with
in App and if you click on Implicit example AbhiAndroid 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 lets 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_horizo
ntal_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

```
package com.example.android.intents;

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.abhiandroid.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:

```
package com.example.android.intents;


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.

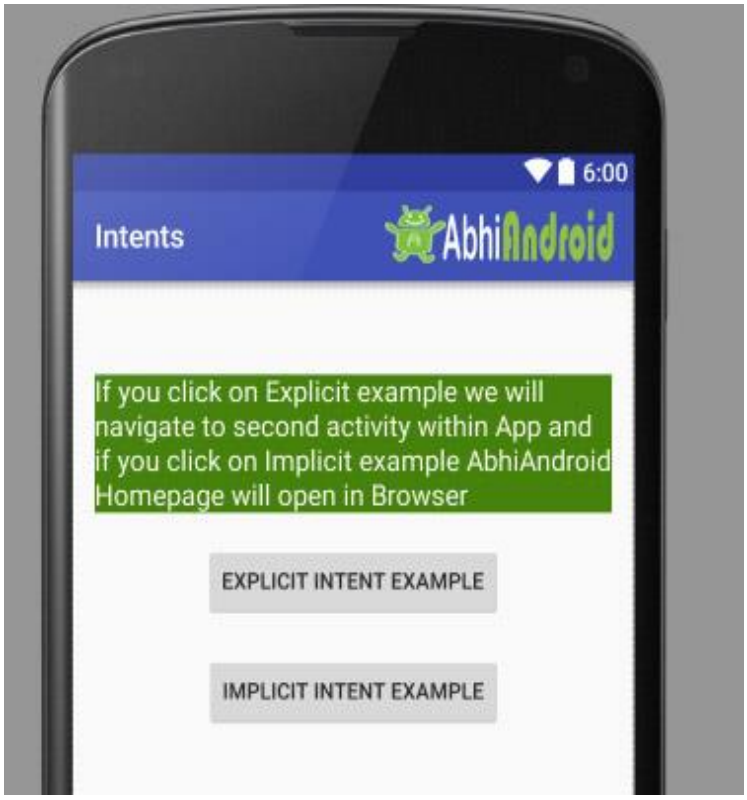
Below is the code of Manifest file:

```
<?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>
```

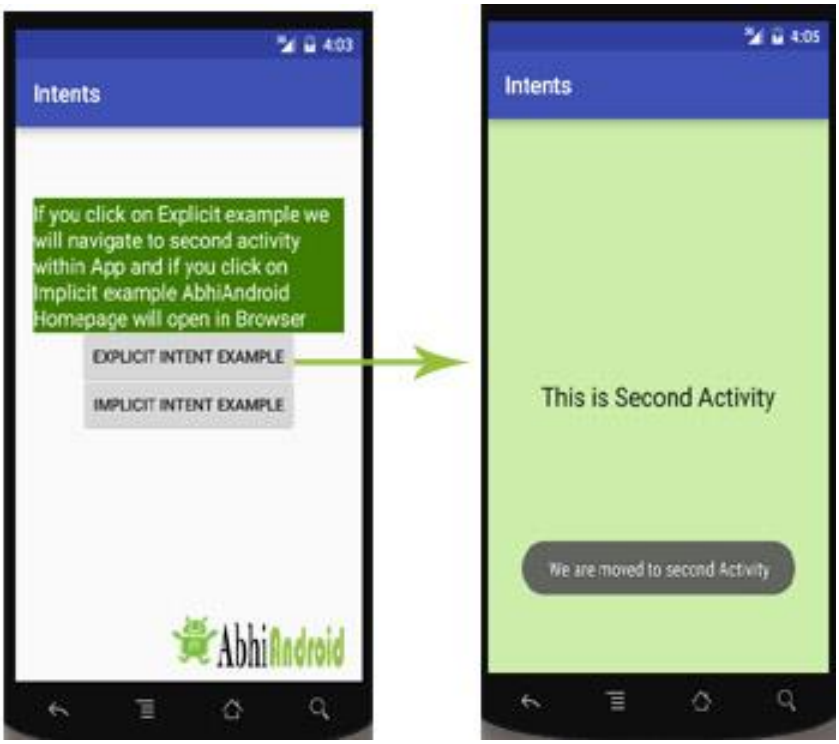
```
</manifest>
```

Output:

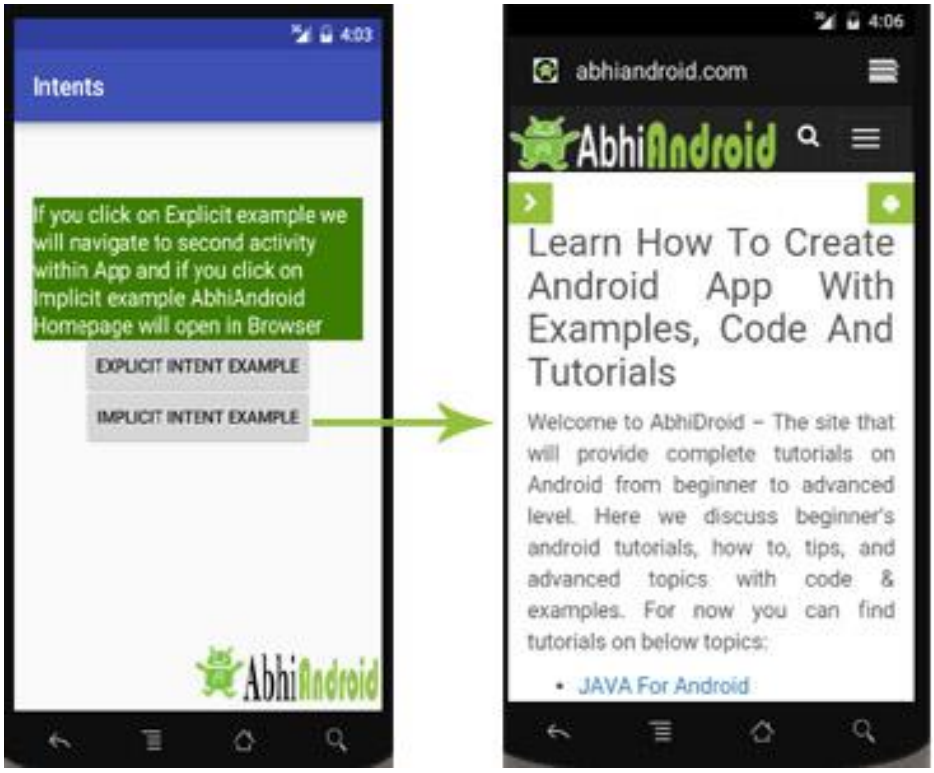
Now run the above program in your Emulator. The App will look like this:



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 AbhiAndroid.com homepage will open in Browser (make sure you have internet):



Intent Uses In Android:

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

Intent 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.

Intent 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.

Intent for Broadcast Receivers:

There are various message 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.

Importance of using Intents in 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.

Intents are really easy to handle and it facilitates communication of components and activities of your application. Moreover you can communicate to another application and send some data to another application using Intents.

Intent Filter in Android Manifest

Intent Filter are the components which decide the behavior of an intent. As we have read in our previous tutorial of Intent about the navigation of one activity to another, that can be achieve by declaring intent filter. We can declare an Intent Filter for an Activity in manifest file.

Intent filters specify the type of intents that an Activity, service or Broadcast receiver can respond to. It declares the functionality of its parent component (i.e. activity, services or broadcast receiver). It declares the capability of any activity or services or a broadcast receiver.

Intent Filter Code Inside Android Manifest:

The code of Intent Filter is used inside AndroidManifest.xml file for an activity. You can see it: open manifests folder >> open AndroidManifest.xml file

Syntax of Intent Filters:

Intent filter is declared inside Manifest file for an Activity.

```
<!--Here Name of Activity is .MainActivity, image of icon name stored in drawable folder, l
abel present inside string name is label-->

<!--If anything is different please customize the code according to your app-->

<activity android:name=".MainActivity">

 <intent-filter android:icon="@drawable/icon"

 android:label="@string/label"

 >

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

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

</intent-filter>
```

```
</activity>
```

Important Note: If you are using above code make sure below things are correct:

icon: This is displayed as icon for activity. You can check or save png image of name icon in drawable folder. If icon image of any other name is stored please replace @drawable/icon with that image name i.e. @drawable/image_name.

label: The label / title that appears at top in Toolbar of that particular Activity. You can check or edit label name by opening String XML file present inside Values folder (Values -> Strings). Replace @string/label to @string/yourlabel.

priority: This is another important attribute in Intent filter used in broadcasting. We will explain it in future topics of broadcasting.

Attributes of Intent Filter:

Below are the attributes of Intent filter:

1. android:icon

An icon represents the activity, service or broadcast receiver when a user interact with it or when it appears to user in an application. To set an icon you need to give reference of drawable resource as declared android:icon="@drawable/icon".

How to add custom icon in your App:

Step 1: Open your android project folder in computer / system

Step 2: Open app>> src >> main >> res >> drawable >> here save custom icon image with name

Step 3: Replace @drawable/icon with @drawable/your_image name in the below code

```
android:icon="@drawable/icon"
```

Important Note: If you have not declared any icon for your activity then android sets the icon of parent component by default. And if no icon is defined by parent component then icon is set as the default icon by <application> element (declared in Manifest file).

2. android:label

A label represents the title of an activity on the toolbar. You can have different Labels for different Activities as per your requirement or choice. The label should be set as a reference to a string resource. However, you can also use a raw string to set a label as declared in the below given code snippet

```
android:label = "@string/label"
```

or

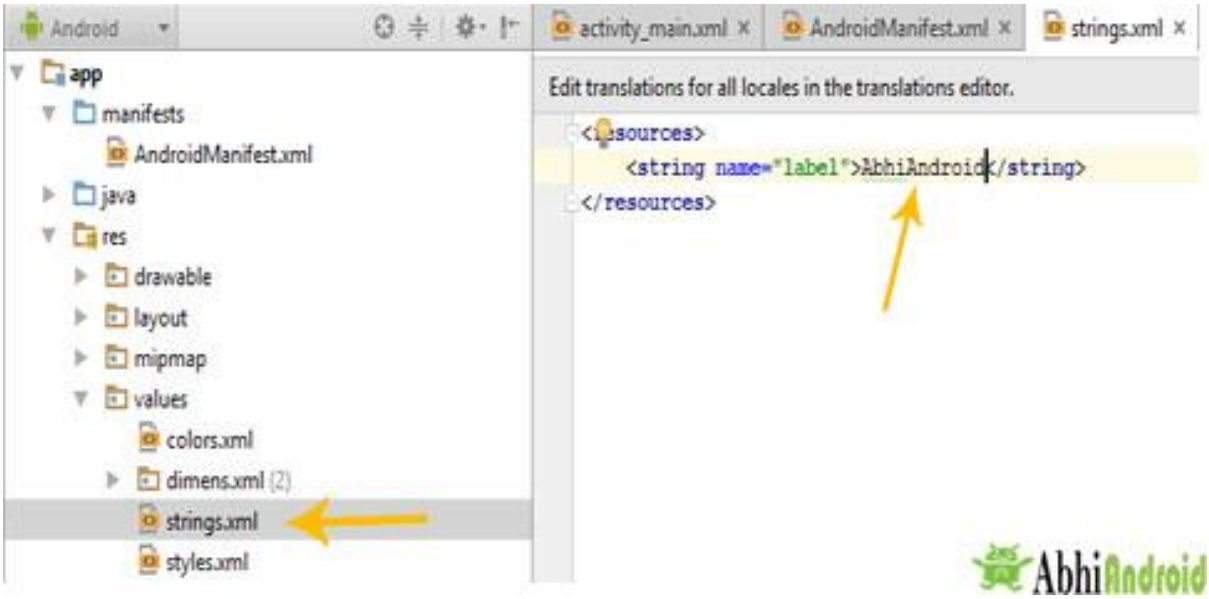
```
android:label = "New Activity"
```

How To Create Custom Label in App:

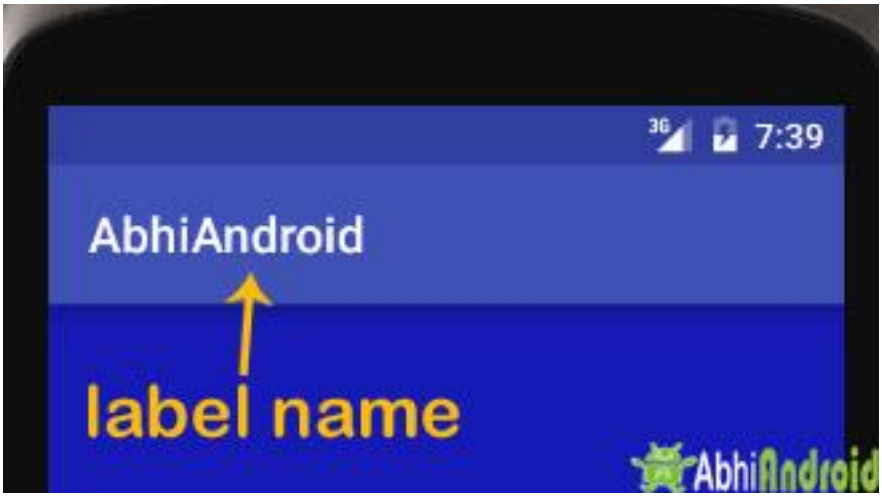
Step 1: Click on values values

Step 2: Open Strings.xml file present inside values

Step 3: Edit String value to your custom name. For example,



Step 4: Now run the App and you will see your custom name in Toolbar. Below you can see AbhiAndroid printed.



Important Note: Just like icon attribute, if you have not declared any label for your activity then it will be same as your parent activity. However if you haven't declared any label to parent activity then label is set by <application> element' label attribute.
Below is sample code of Intent filter for an App having two activity:

```
<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>

<!--Add code here for first activity-->

</intent-filter>
</activity>
<activity android:name=".Main2Activity">
<intent-filter>
<!--Add code here for second activity-->
</intent-filter>
</activity>
</application>
```

Elements In Intent Filter:

There are following three elements in an intent filter:

1. Action
2. Data
3. Category

Important Note: Every intent filter must contain action element in it. Data and category element is optional for it.

1. Action:

It represent an activities action, what an activity is going to do. It is declared with the name attribute as given below

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

An Intent Filter element must contain one or more action element. Action is a string that specifies the action to perform. You can declare your own action as given below. But we usually use action constants defined by Intent class.

```
<intent-filter>
<action android:name="com.example.android.intentfilters.Main2Activity"/>
</intent-filter>
```

There are few common actions for starting an activity like ACTION_VIEW

ACTION_VIEW: This is used in an Intent with startActivity(). This helps when you redirect to see any website, photos in gallery app or an address to view in a map app.

For example: As we have done in previous topic of Intent, we started a website using implicit intent in that Intent we used ACTION_VIEW element to view website in the web browser.

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

There are more actions similar to above like, ACTION_SEND, ACTION_MAIN, ACTION_WEB_SEARCH and many more.

2. Data:

There are two forms in which you can pass the data, using URI(Uniform Resource Identifiers) or MIME type of data. For understanding the concept of URI in better manner check the link.

The syntax of data attribute is as follows:

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

This specifies the format of data associated with an Intent which you use with component. As explained in above code snippet, data passed through Intent is a type of URI. Check the below given table for better clarification

ACTION	DATA	MEANING
Intent.ACTION_CALL	tel:phone_number	Opens phone application and calls phone number
Intent.ACTION_DIAL	tel:phone_number	Opens phone application and dials (but doesn't call) phone_number
Intent.ACTION_DIAL	voicemail:	Opens phone application and dials (but doesn't call) the voice mail number.
Intent.ACTION_VIEW	geo:lat,long	Opens the maps Application centered on (lat, long).
Intent.ACTION_VIEW	geo:0,0?q=address	Opens the maps application centered on the

		specified address.
Intent.ACTION_VIEW	http://url https://url	Opens the browser application to the specified address.
Intent.ACTION_WEB_SEARCH	plain_text	Opens the browser application and uses Google search for given string

3. Category:

This attribute of Intent filter dictates the behavior or nature of an Intent. There is a string which contains some additional information about the intent which will be handled by a component. The syntax of category is as follows:

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

Most of Intents do not require a category for example: CATEGORY_BROWSABLE, CATEGORY_LAUNCHER.

BROWSABLE – Browsable category, activity allows itself to be opened with web browser to open the reference link provided in data.

LAUNCHER – Launcher category puts an activity on the top of stack, whenever application will start, the activity containing this category will be opened first.

```
<intent-filter>
<!--Code here-->
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
<intent-filter>
<!--Code here-->
<category android:name="android.intent.category.BROWSABLE" />
</intent-filter>
```

Intent Filter Example:

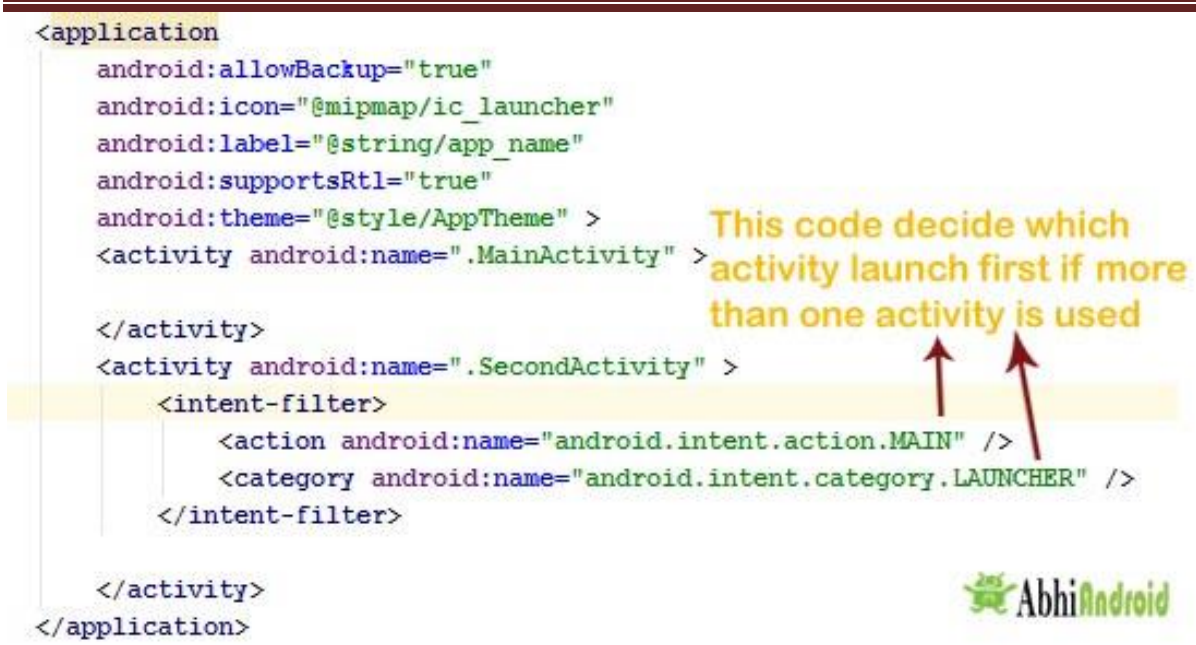
Let's take an example of Intent filter for an Activity. In this example, we will make an Activity a launcher activity by declaring the intent filter. By launcher we mean this Activity will be launched first out of all activities in App.

Below code snippet gives the details for the same. You have to make changes in Android Manifest file to make your activity launcher activity.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.android.intentfilters">
<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=".Main2Activity">
<intent-filter>
<action android:name="com.example.android.intentfilters.Main2Activity"/>
<category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
</activity>
</application>
</manifest>
```

Important Note: In above mentioned code snippet we are using two activities which you have to declare it in the Manifest that which activity has to launch first when your application starts. In above code we have made MainActivity as a Launcher Activity by declaring Intent Filter:




```
<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" >

    </activity>
    <activity android:name=".SecondActivity" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>

    </activity>
</application>
```

Activity Lifecycle With Example In Android – Tutorial, Code And Importance

Activity Lifecycle: Activity is one of the building blocks of Android OS. In simple words Activity is a screen that user interact with. *Every Activity in android has lifecycle like created, started, resumed, paused, stopped or destroyed. These different states are known as Activity Lifecycle.* In other words we can say Activity is a class pre-written in Java Programming.

Below is Activity Lifecycle Table:

Short description of Activity Lifecycle example:

onCreate() – Called when the activity is first created

onStart() – Called just after it's creation or by restart method after onStop(). Here Activity start becoming visible to user

onResume() – Called when Activity is visible to user and user can interact with it

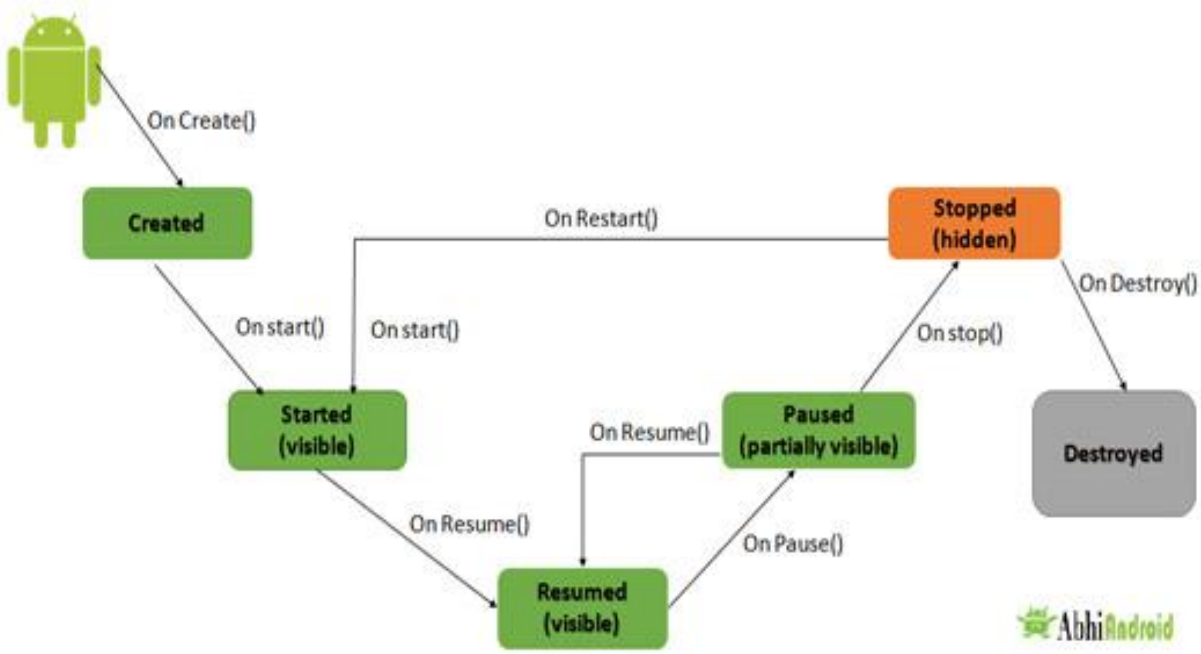
onPause() – Called when Activity content is not visible because user resume previous activity

onStop() – Called when activity is not visible to user because some other activity takes place of it

onRestart() – Called when user comes on screen or resume the activity which was stopped

onDestroy – Called when Activity is not in background

Below Activity Lifecycle Diagram Shows Different States:



Different Types of Activity Lifecycle States:

Activity have different states or it's known as Activity life cycle. All life cycle methods aren't required to override but it's quite important to understand them. Lifecycles methods can be overridden according to requirements.

LIST OF ACTIVITY LIFECYCLE METHODS OR STATES:

Activity Created: onCreate(Bundle savedInstanceState):

onCreate() method is called when activity gets memory in the OS. **To use create state we need to override onCreate(Bundle savedInstanceState) method.** Now there will be question in mind what is Bundle here, so Bundle is a data repository object that can store any kind of primitive data and this object will be null until some data isn't saved in that.

- When an Activity first call or launched then onCreate(Bundle savedInstanceState) method is responsible to create the activity.
- When ever orientation(i.e. from horizontal to vertical or vertical to horizontal) of activity gets changed or when an Activity gets forcefully terminated by any Operating System then savedInstanceState i.e. object of Bundle Class will save the state of an Activity.
- It is best place to put initialization code.

Learn More About onCreate(Bundle savedInstanceState) With Example

Activity Started: onStart():

onStart() method is called just after it's creation. In other case Activity can also be started by calling restart method i.e after activity stop. So this means onStart() gets called by Android OS when user switch between applications. For example, if a user was using Application A and then a notification comes and user clicked on notification and moved to

Application B, in this case Application A will be paused. And again if a user again click on app icon of Application A then Application A which was stopped will again gets started.

Learn More About onStart() With Example

Activity Resumed:.onResume():

Activity resumed is that situation when it is actually visible to user means the data displayed in the activity is visible to user. In lifecycle it always gets called after activity start and in most use case after activity paused (**onPause**).

Activity Paused: onPause():

Activity is called paused when it's content is not visible to user, in most case onPause() method called by Android OS when user press Home button (Center Button on Device) to make hide.

Activity also gets paused before stop called in case user press the back navigation button. The activity will go in paused state for these reasons also if a notification or some other dialog is overlaying any part (top or bottom) of the activity (screen). Similarly, if the other screen or dialog is transparent then user can see the screen but cannot interact with it. For example, if a call or notification comes in, the user will get the opportunity to take the call or ignore it.

Learn More About onPause() With Example

Activity Stopped: onStop():

Activity is called stopped when it's not visible to user. Any activity gets stopped in case some other activity takes place of it. For example, if a user was on screen 1 and click on some button and moves to screen 2. In this case Activity displaying content for screen 1 will be stopped.

Every activity gets stopped before destroy in case of when user press back navigation button. So Activity will be in stopped state when hidden or replaced by other activities that have been launched or switched by user. In this case application will not present anything useful to the user directly as it's going to stop.

Learn More About onStop() With Example

Activity Restarted: onRestart():

Activity is called in restart state after stop state. So activity's onRestart() function gets called when user comes on screen or resume the activity which was stopped. In other words, when Operating System starts the activity for the first time onRestart() never gets called. It gets called only in case when activity is resumes after stopped state.

Activity Destroyed: onDestroy():

Any activity is known as in destroyed state when it's not in background. There can different cases at what time activity get destroyed.

First is if user pressed the back navigation button then activity will be destroyed after completing the lifecycle of pause and stop.

In case if user press the home button and app moves to background. User is not using it no more and it's being shown in recent apps list. So in this case if system required resources need to use somewhere else then OS can destroy the Activity.

After the Activity is destroyed if user again click the app icon, in this case activity will be recreated and follow the same lifecycle again. Another use case is with Splash Screens if there is call to finish() method from onCreate() of an activity then OS can directly call onDestroy() with calling onPause() and onStop().

Activity Lifecycle Example:

In the below example we have used the below JAVA and Android topics:

JAVA Topics Used: Method Overriding, static variable, package, Inheritance, method and class.

Android Topic Used: We have used Log class which is used to printout message in Logcat. One of the important use of Log is in debugging.

First we will create a new Android Project and name the activity as HomeActivity. In our case we have named our App project as Activity Lifecycle Example.

We will initialize a static String variable with the name of the underlying class using getSimpleName() method. In our case HOME_ACTIVITY_TAG is the name of the String variable which store class name HomeActivity.

```
private static final String HOME_ACTIVITY_TAG = HomeActivity.class.getSimpleName();
```

Now we will create a new method which will print message in Logcat.

```
private void showLog(String text){


  Log.d(HOME_ACTIVITY_TAG, text);


}
```

Now we will override all activity lifecycle method in Android and use showLog() method which we creating for printing message in Logcat.

Complete JAVA code of HomeActivity.java:

```
package com.abhiandroid.activitylifecycleexample;

import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.util.Log;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;
```

```
public class HomeActivity extends AppCompatActivity {
    private static final String HOME_ACTIVITY_TAG = HomeActivity.class.getSimpleName
();

    private void showLog(String text){
        Log.d(HOME_ACTIVITY_TAG, text);
    }
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        showLog("Activity Created");
    }
    @Override
    protected void onRestart(){
        super.onRestart();//call to restart after onStop
        showLog("Activity restarted");
    }
    @Override
    protected void onStart() {
        super.onStart();//soon be visible
        showLog("Activity started");
    }
    @Override
    protected void onResume() {
        super.onResume();//visible
```

```
        showLog("Activity resumed");
    }
    @Override
    protected void onPause() {
        super.onPause();//invisible
        showLog("Activity paused");
    }
    @Override
    protected void onStop() {
        super.onStop();
        showLog("Activity stopped");
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        showLog("Activity is being destroyed");
    }
}
```

When creating an Activity we need to register this in AndroidManifest.xml file. Now question is why need to register? **Its actually because manifest file has the information which Android OS read very first.** When registering an activity other information can also be defined within manifest like Launcher Activity (An activity that should start when user click on app icon).

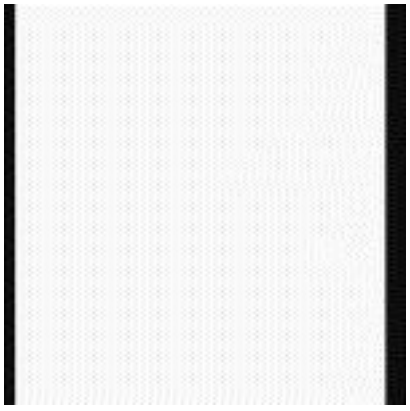
Here is declaration example in AndroidManifest.xml file

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

  <application
```

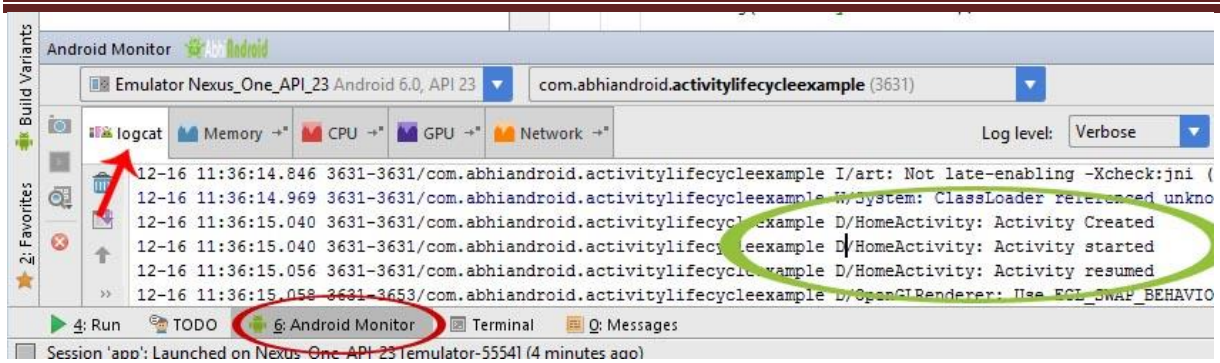
```
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity
      android:name=".HomeActivity"
      android:label="@string/app_name"
      android:theme="@style/AppTheme.NoActionBar">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

Output Of Activity Lifecycle:

When you will run the above program you will notice a blank white screen will open up in Emulator. *You might be wondering where is default Hello world screen. Actually we have removed it by overriding onCreate() method.* Below is the blank white screen that will pop up.



Now go to Logcat present inside Android Monitor: Scroll up and you will notice three methods which were called: Activity Created, Activity started and Activity resumed.

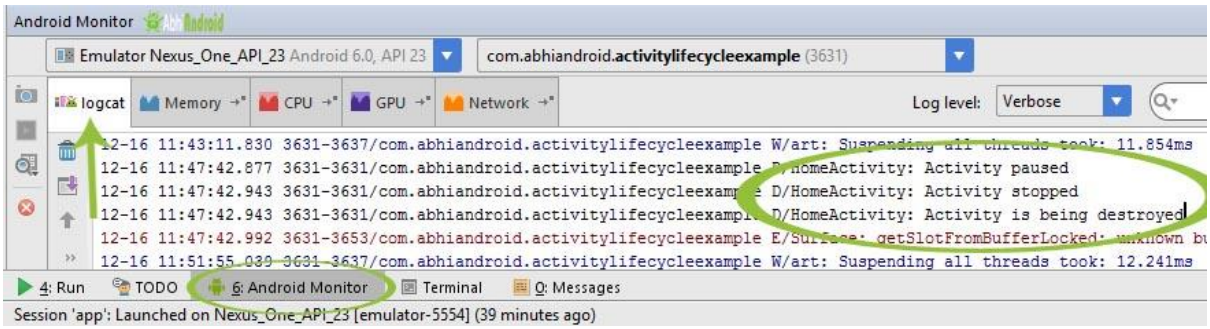


So this clears:

- first onCreate() method was called when activity was created
- second onStart() method was called when activity start becoming visible to user
- Finally onResume() method was called when activity is visible to user and user can interact with it

Now press the back button on the Emulator and exit the App:

Go to Logcat again and scroll down to bottom. You will see 3 more methods were called: Activity paused, Activity stopped and Activity is being destroyed.



So this clears:

- onPause() method was called when user resume previous activity
- onStop() method was called when activity is not visible to user
- Last onDestroy() method was called when Activity is not in background

Important Note: In the above example onRestart() won't be called because there was no situation when we can resume the onStart() method again. In future example we will show you onRestart() in action as well.

Importance Of Activity Life Cycle:
Activity is the main component of Android Application, as every screen is an activity so to create any simple app first we have to start with Activities. Every lifecycle method is quite important to implement according to requirements, However onCreate(Bundle state) is always needed to implement to show or display some content on screen.

BroadcastReceivers

In android, **Broadcast Receiver** is a component which will allow android system or other apps to deliver events to the app like sending a low battery message or screen turned off message to the app. The apps can also initiate broadcasts to let other apps know that required data available in a device to use it.

Generally, we use Intents to deliver broadcast events to other apps and Broadcast Receivers use status bar notifications to let user know that broadcast event occurs.

In android, Broadcast Receiver is implemented as a subclass of **BroadcastReceiver** and each broadcast is delivered as an Intent object.

We can register an app to receive only few broadcast messages based on our requirements. When a new broadcast received, the system will check for specified broadcasts have subscribed or not based on that it will routes the broadcasts to the apps.

Receiving Broadcasts

In android, we can receive broadcasts by registering in two ways.

One way is by registering a broadcasts using android application manifest file (**AndroidManifest.xml**). We need to specify **<receiver>** element in apps manifest file like as shown below.

```
<receiver android:name=".SampleBroadcastReceiver">
  <intent-filter>
    <action android:name="android.intent.action.BOOT_COMPLETED"/>
  </intent-filter>
</receiver>
```

The above statement will fire the defined system broadcast event whenever the boot process is completed.

Another way is to register a receiver dynamically
via **Context.registerReceiver()** method. To register broadcast receiver we need to extend our class using **BroadcastReceiver** and need to implement a **onReceive(Context, Intent)** method like as shown below.

```
public class MainActivity extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
      Toast.makeText(context, log, Toast.LENGTH_LONG).show();
   }
}
```

Suppose if we register for **ACTION_BOOT_COMPLETED** event, whenever the boot process is completed the broadcast receiver's method **onReceive()** method will be invoked.

Sending Broadcasts

In android, we can send a broadcasts in apps using three different ways, those are

Method	Description
sendOrderedBroadcast(Intent, String)	This method is used to send broadcasts to one receiver at a time.
sendBroadcast(Intent)	This method is used to send broadcasts to all receivers in an undefined order.
LoadBroadcastManager.sendBroadcast	This method is used to send broadcasts to receivers that are in the same app as the sender.

Broadcasting Custom Intents

In android we can create our own custom broadcasts using intents. Following is the simple code snippet of sending a broadcast by creating an intent using **sendBroadcast(Intent)** method.

```
Intent sintent = new Intent();
intent.setAction("com.tutlane.broadcast.MY_NOTIFICATION");
intent.putExtra("data","Welcome to Tutlane");
sendBroadcast(intent);
```

If you observe above code snippet we create a custom Intent "**sintent**". We need to register our intent action in android manifest file like as shown below

```
<receiver android:name=".SampleBroadcastReceiver">
   <intent-filter>
      <action android:name=" com.tutlane.broadcast.MY_NOTIFICATION"/>
   </intent-filter>
</receiver>
```

This is how we can create our own custom broadcasts using Intents in android applications.

System Broadcasts

In android, several system events are defined as final static fields in the Intent class. Following are the some of system events available in android applications.

Event	Description
android.intent.action.BOOT_COMPLETED	It raise an event, once boot completed.
android.intent.action.POWER_CONNECTED	It is used to trigger an event when power connected to the device.
android.intent.action.POWER_DISCONNECTED	It is used to trigger an event when power got disconnected from the device.
android.intent.action.BATTERY_LOW	It is used to call an event when battery is low on device.
android.intent.action.BATTERY_OKAY	It is used to call an event when battery is OKAY again.
android.intent.action.REBOOT	It call an event when the device rebooted again.

Likewise we have many system events available in android application.

Android BroadcastReceiver Example

To setup a Broadcast Receiver in our android application we need to do following things.

- Create a BroadcastReceiver
- Register a BroadcastReceiver

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

Now we need to create our own broadcast content file **MyBroadcastReceiver.java** in **\src\main\java\com.tutlane.broadcastreceiver** pa th to define our actual provider and associated methods for that right click on your application folder Go to **New** select **Java Class** and give name as **MyBroadcastReceiver.java**.

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

MyBroadcastReceiver.java

```
package com.tutlane.broadcastreceiver;
import android.content.BroadcastReceiver;
```

```
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

/**
 * Created by Tutlane on 01-08-2017.
 */

public class MyBroadcastReceiver extends BroadcastReceiver {
  @Override
   public void onReceive(Context context, Intent intent){
     CharSequence iData = intent.getCharSequenceExtra("msg");
     Toast.makeText(context,"Tutlane Received Message: "+iData,Toast.LENGTH_LONG).s
how();
  }
}
```

Now open **activity_main.xml** file from **\src\main\res\layout** path and write the following code.

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">
  <EditText
     android:id="@+id/txtMsg"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_marginLeft="100dp"
     android:layout_marginTop="180dp"
     android:ems="10"/>
  <Button
     android:id="@+id/btnShow"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:onClick="onClickShowBroadcast"
     android:layout_marginLeft="130dp"
     android:text="Show Broadcast"/>
</LinearLayout>
```

Now open **MainActivity.java** file from **\java\com.tutlane.broadcastreceiver** path and write following to implement custom broadcast intents.

MainActivity.java

```
package com.tutlane.broadcastreceiver;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
import android.view.View;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
  public void onClickShowBroadcast(View view){
    EditText st = (EditText)findViewById(R.id.txtMsg);
    Intent intent = new Intent();
    intent.putExtra("msg",(CharSequence)st.getText().toString());
    intent.setAction("com.tutlane.CUSTOM_INTENT");
    sendBroadcast(intent);
  }
}
```

Now we need to register our broadcast receiver in android manifest file (**AndroidManifest.xml**) using **<receiver>** attribute 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.tutlane.broadcastreceiver">

  <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>
    <receiver android:name="MyBroadcastReceiver">
      <intent-filter>
        <action android:name="com.tutlane.CUSTOM_INTENT">
        </action>
      </intent-filter>
    </receiver>
  </application>
</manifest>
```

Output of Android BroadcastReceiver Example

When we run above example in android emulator we will get a result like as shown below

Content Providers

In android, **Content Provider** will act as a central repository to store the applications data in one place and make that data available for different applications to access whenever it's required.

In android, we can configure Content Providers to allow other applications securely access and modify our app data based on our requirements.

Generally, the **Content Provider** is a part of an android application and it will act as more like relational database to store the app data. We can perform a multiple operations like insert, update, delete and edit on the data stored in content provider using **insert()**, **update()**, **delete()** and **query()** methods.

In android, we can use content provider whenever we want to share our app data with other apps and it allow us to make a modifications to our application data without effecting other applications which depends on our app.

In android, content provider is having different ways to store app data. The app data can be stored in a SQLite database or in files or even over a network based on our requirements. By using content providers we can manage data such as audio, video, images and personal contact information.

We have a different type of access permissions available in content provider to share the data. We can set a restrict access permissions in content provider to restrict data access limited to only our application and we can configure different permissions to read or write a data.

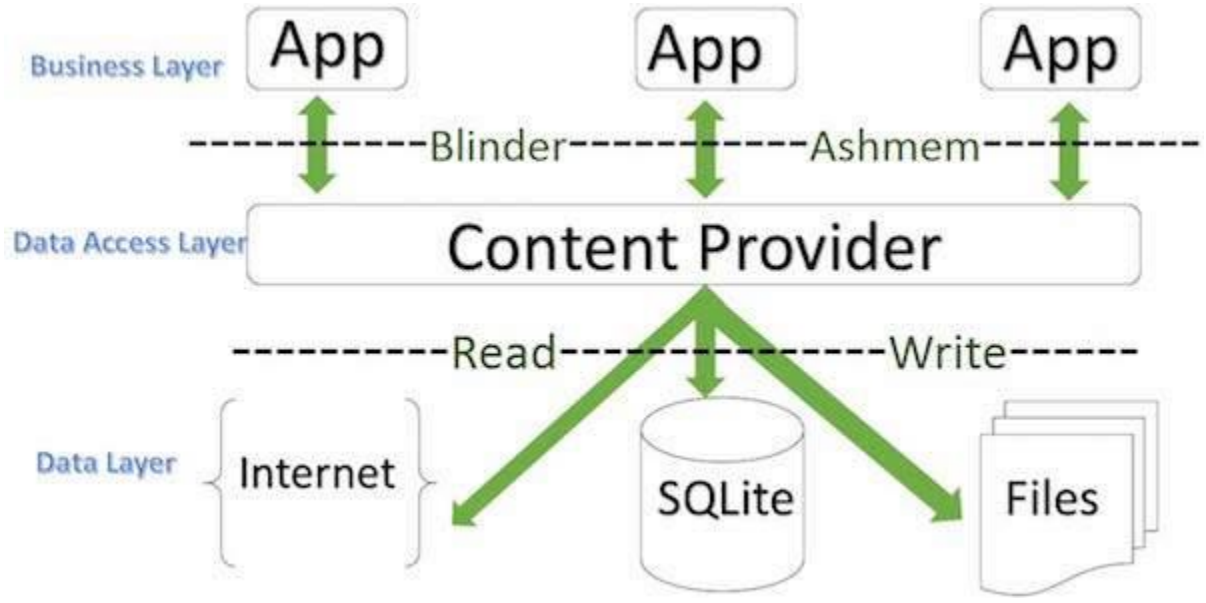
Access Data from Content Provider

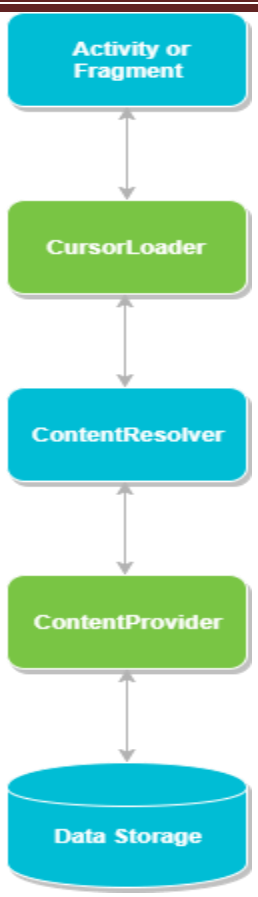
To access a data from content provider, we need to use ContentResolver object in our application to communicate with the provider as a client. The ContentResolver object will communicate with the provider object (ContentProvider) which is implemented by instance of class.

Generally, in android to send a request from UI to ContentResolver we have another object called **CursorLoader** which is used to run the query asynchronously in background. In android application the UI components such as Activity or Fragment will call a **CursorLoader** to query and get a required data from ContentProvider using ContentResolver.

The ContentProvider object will receive a data requests from client, performs the requested actions (create, update, delete, retrieve) and return the result.

Following is the pictorial representation of requesting an operation from UI using Activity or Fragment to get the data from ContentProvider object.





This is how the interaction will happen between android application UI and content providers to perform required actions to get a data.

Content URIs

In android, **Content URI** is an URI which is used to query a content provider to get the required data. The Content URIs will contain the name of entire provider (**authority**) and the name that points to a table (**path**).

Generally the format of URI in android applications will be like as shown below

```
content://authority/path
```

Following are the details about various parts of an URI in android application.

content:// - The string **content://** is always present in the URI and it is used to represent the given URI is a content URI.

authority - It represents the name of content provider, for example phone, contacts, etc. and we need to use fully qualified name for third party content providers like com.tutlane.contactprovider

path - It represents the table's path.

The ContentResolver object use the URI's **authority** to find the appropriate provider and send the query objects to the correct provider. After that ContentProvider uses the **path** of content URI to choose the right table to access.

Following is the example of simple example of URI in android applications.

```
content://contacts_info/users
```

Here the string **content://** is used to represent URI is a content URI, **contacts_info** string is the name of provider's authority and **users** string is the table's path.

Creating a Content Provider

To create a content provider in android applications we should follow below steps.

- We need to create a content provider class that extends the ContentProvider base class.
- We need to define our content provider URI to access the content.
- The ContentProvider class defines a six abstract methods (insert(), update(), delete(), query(), getType()) which we need to implement all these methods as a part of our subclass.
- We need to register our content provider in AndroidManifest.xml using **<provider>** tag.

Following are the list of methods which need to implement as a part of ContentProvider class.

query() - It receives a request from the client. By using arguments it will get a data from requested table and return the data as a **Cursor** object.

insert() - This method will insert a new row into our content provider and it will return the content URI for newly inserted row.

update() - This method will update an existing rows in our content provider and it return the number of rows updated.

delete() - This method will delete the rows in our content provider and it return the number of rows deleted.

getType() - This method will return the MIME type of data to given content URI.

onCreate() - This method will initialize our provider. The android system will call this method immediately after it creates our provider.

Android Content Provider Example

Following is the example of using **Content Provider** in android applications. Here we will create our own content provider to insert and access data in android application.

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

Now we need to create our own content provider file **UserProvider.java** in **\src\main\java\com.tutlane.contentprovider** path to define our actual provider and associated methods for that right click on your application folder Go to **New** select **Java Class** and give name as **UserProvider.java**.

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

UserProvider.java

```
package com.tutlane.contentprovider;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import java.util.HashMap;

/**
```

```
 * Created by sureshdasari on 29-07-2017.
 */

public class UsersProvider extends ContentProvider {
   static final String PROVIDER_NAME = "com.tutlane.contentprovider.UserProvider";
   static final String URL = "content://" + PROVIDER_NAME + "/users";
   static final Uri CONTENT_URI = Uri.parse(URL);

   static final String id = "id";
   static final String name = "name";
   static final int uriCode = 1;
   static final UriMatcher uriMatcher;
   private static HashMap<String, String> values;
   static {
      uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
      uriMatcher.addURI(PROVIDER_NAME, "users", uriCode);
      uriMatcher.addURI(PROVIDER_NAME, "users/*", uriCode);
   }

  @Override
   public String getType(Uri uri) {
      switch (uriMatcher.match(uri)) {
         case uriCode:
            return "vnd.android.cursor.dir/users";
         default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
      }
   }

   @Override
   public boolean onCreate() {
      Context context = getContext();
      DatabaseHelper dbHelper = new DatabaseHelper(context);
      db = dbHelper.getWritableDatabase();
      if (db != null) {
         return true;
      }
      return false;
   }

   @Override
   public Cursor query(Uri uri, String[] projection, String selection,
                       String[] selectionArgs, String sortOrder) {
      SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
      qb.setTables(TABLE_NAME);

      switch (uriMatcher.match(uri)) {
         case uriCode:
            qb.setProjectionMap(values);
            break;
```

```
        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
    }
    if (sortOrder == null || sortOrder == "") {
        sortOrder = id;
    }
    Cursor c = qb.query(db, projection, selection, selectionArgs, null,
            null, sortOrder);
    c.setNotificationUri(getContext().getContentResolver(), uri);
    return c;
}
@Override
public Uri insert(Uri uri, ContentValues values) {
    long rowID = db.insert(TABLE_NAME, "", values);
    if (rowID > 0) {
        Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
        getContext().getContentResolver().notifyChange(_uri, null);
        return _uri;
    }
    throw new SQLiteException("Failed to add a record into " + uri);
}
@Override
public int update(Uri uri, ContentValues values, String selection,
                String[] selectionArgs) {
    int count = 0;
    switch (uriMatcher.match(uri)) {
        case uriCode:
            count = db.update(TABLE_NAME, values, selection, selectionArgs);
            break;
        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    int count = 0;
    switch (uriMatcher.match(uri)) {
        case uriCode:
            count = db.delete(TABLE_NAME, selection, selectionArgs);
            break;
        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}
private SQLiteDatabase db;
static final String DATABASE_NAME = "EmpDB";
```

```
  static final String TABLE_NAME = "Employees";
  static final int DATABASE_VERSION = 1;
  static final String CREATE_DB_TABLE = " CREATE TABLE " + TABLE_NAME
        + " (id INTEGER PRIMARY KEY AUTOINCREMENT, "
        + " name TEXT NOT NULL);";

  private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context) {
      super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
      db.execSQL(CREATE_DB_TABLE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
      db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
      onCreate(db);
    }
  }
}
```

Now open an **activity_main.xml** file from **\src\main\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:orientation="vertical" android:layout_width="match_parent"
  android:layout_height="match_parent">

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Name"
    android:layout_marginLeft="100dp"
    android:layout_marginTop="100dp"/>
  <EditText
    android:id="@+id/txtName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="100dp"
    android:ems="10"/>
  <Button
    android:id="@+id/btnAdd"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClickAddDetails"
```

```
    android:layout_marginLeft="100dp"
    android:text="Add User"/>

  <Button
    android:id="@+id/btnRetrieve"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClickShowDetails"
    android:layout_marginLeft="100dp"
    android:text="Show Users"/>
  <TextView
    android:id="@+id/res"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="100dp"
    android:clickable="false"
    android:ems="10"/>
</LinearLayout>
```

Now open **MainActivity.java** file and write the code like as shown below

MainActivity.java

```
package com.tutlane.contentprovider;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.view.inputmethod.InputMethodManager;
import android.widget.EditText;
import android.widget.TextView;
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 onTouchEvent(MotionEvent event) {
    InputMethodManager imm = (InputMethodManager)getSystemService(Context.INPU
T_METHOD_SERVICE);
    imm.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(), 0);
    return true;
```

```
  }
  public void onClickAddDetails(View view) {
    ContentValues values = new ContentValues();
    values.put(UsersProvider.name, ((EditText) findViewById(R.id.txtName)).getText().t
oString());
    getContentResolver().insert(UsersProvider.CONTENT_URI, values);
    Toast.makeText(getBaseContext(), "New Record Inserted", Toast.LENGTH_LONG).sho
w();
  }

  public void onClickShowDetails(View view) {
    // Retrieve employee records
    TextView resultView= (TextView) findViewById(R.id.res);
    Cursor cursor = getContentResolver().query(Uri.parse("content://com.tutlane.conte
ntprovider.UserProvider/users"), null, null, null, null);
    if(cursor.moveToFirst()) {
      StringBuilder strBuild=new StringBuilder();
      while (!cursor.isAfterLast()) {
        strBuild.append("\n"+cursor.getString(cursor.getColumnIndex("id"))+ "-"+ curso
r.getString(cursor.getColumnIndex("name")));
        cursor.moveToNext();
      }
      resultView.setText(strBuild);
    }
    else {
      resultView.setText("No Records Found");
    }
  }
}
```

We need to include this newly created content provider in android manifest file (**ActivityManifest.xml**) using **<provider>** attribute like as shown below.

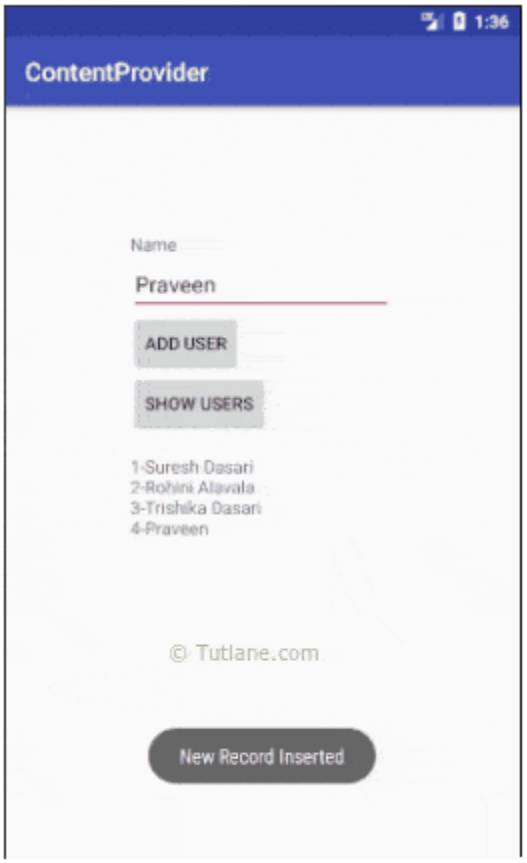
AndroidManifest.xml

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

```
    <provider
      android:authorities="com.tutlane.contentprovider.UserProvider"
      android:name=".UsersProvider">
      </provider>
  </application>
</manifest>
```

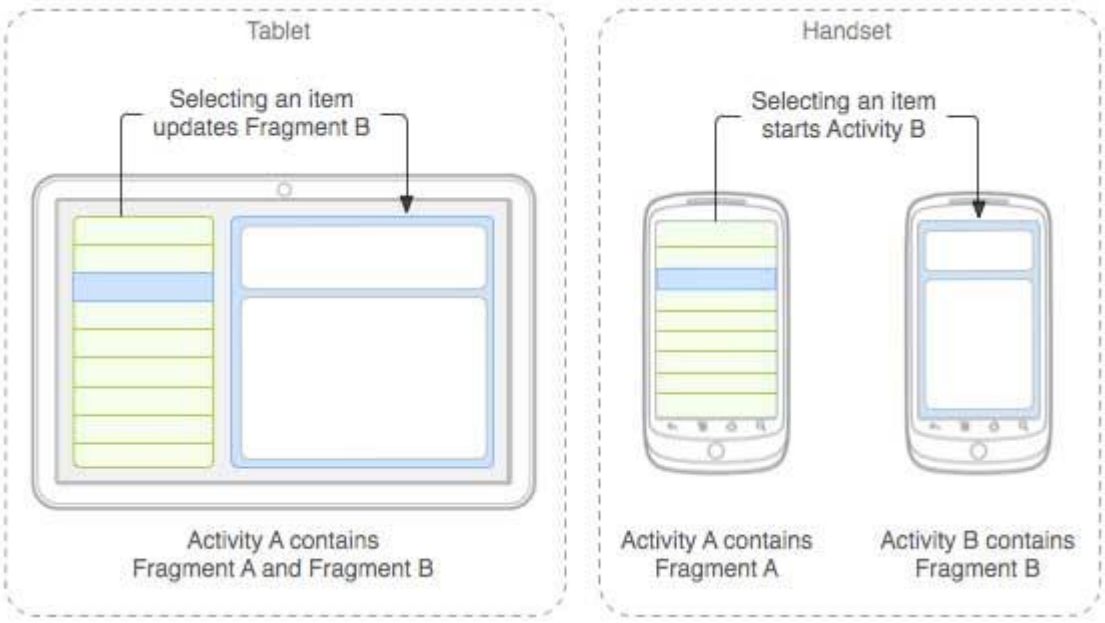
Output of Android Content Provider Example

When we run above example in android emulator we will get a result like as shown below



Fragment In Android Studio

In Android, Fragment is a part of an activity which enables more modular activity design. It will not be wrong if we say a fragment is a kind of sub-activity. It represents behaviour or a portion of user interface in an Activity. We can combine multiple Fragments in Single Activity to build a multi panel UI and reuse a Fragment in multiple Activities. We always need to embed Fragment in an activity and the fragment lifecycle is directly affected by the host activity's lifecycle.



We can create Fragments by extending Fragment class or by inserting a Fragment into our Activity layout by declaring the Fragment in the activity's layout file, as a <fragment> element. We can manipulate each Fragment independently, such as add or remove them.

While performing Fragment Transaction we can add a Fragment into back stack that's managed by the Activity. back stack allow us to reverse a Fragment transaction on pressing Back button of device. For Example if we replace a Fragment and add it in back stack then on pressing the Back button on device it display the previous Fragment.

Some Important Points about Fragment in Android:

1. Fragments were added in Honeycomb version of Android i.e API version 11.

2. We can add, replace or remove Fragment's in an Activity while the activity is running. For performing these operations we need a Layout(Relative Layout, FrameLayout or any other layout) in xml file and then replace that layout with the required Fragment.

3. Fragments has its own layout and its own behaviour with its own life cycle callbacks.

4. Fragment can be used in multiple activities.

5. We can also combine multiple Fragments in a single activity to build a multi-plane UI.

Need of Fragments in Android:

Before the introduction of Fragment's we can only show a single Activity on the screen at one given point of time so we were not able to divide the screen and control different parts separately. With the help of Fragment's we can divide the screens in different parts and controls different parts separately.

By using Fragments we can comprise multiple Fragments in a single Activity. Fragments have their own events, layouts and complete life cycle. It provide flexibility and also removed the limitation of single Activity on the screen at a time.

Basic Fragment Code in XML:

```
<fragment
android:id="@+id/fragments"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

Create A Fragment Class In Android Studio:

For creating a Fragment firstly we extend the Fragment class, then override key lifecycle methods to insert our app logic, similar to the way we would with an Activity class. While creating a Fragment we must use onCreateView() callback to define the layout and in order to run a Fragment.

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.ViewGroup;


public class FirstFragment extends Fragment {
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate(R.layout.fragment_first, container, false);
}
}
```

Here the inflater parameter is a LayoutInflater used to inflate the layout, container parameter is the parent ViewGroup (from the activity's layout) in which our Fragment layout will be inserted.

The savedInstanceState parameter is a Bundle that provides data about the previous instance of the Fragment. The inflate() method has three arguments first one is the resource layout which we want to inflate, second is the ViewGroup to be the parent of the inflated layout. Passing the container is important in order for the system to apply layout parameters to the root view of the inflated layout, specified by the parent view in which it's going and the third parameter is a boolean value indicating whether the inflated layout should be attached to the ViewGroup (the second parameter) during inflation.

Implementation of Fragment in Android Require Honeycomb (3.0) or Later:

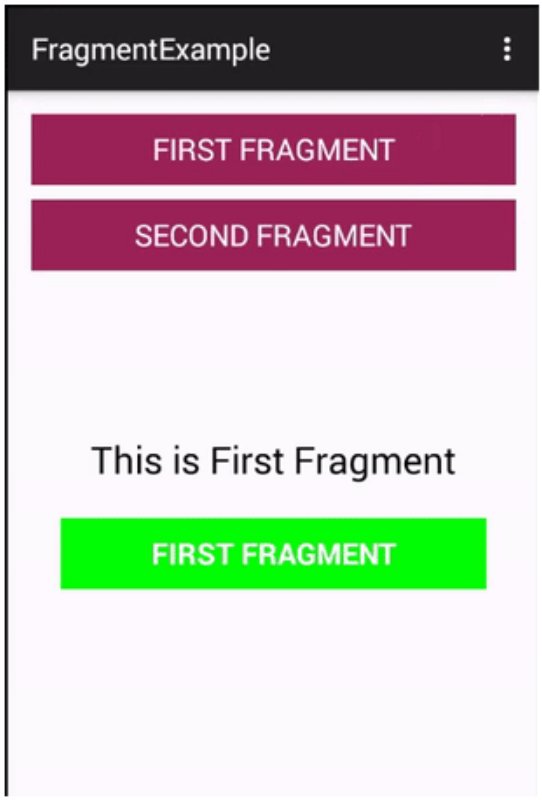
Fragments were added in in Honeycomb version of Android i.e API version 11. There are some primary classes related to Fragment's are:

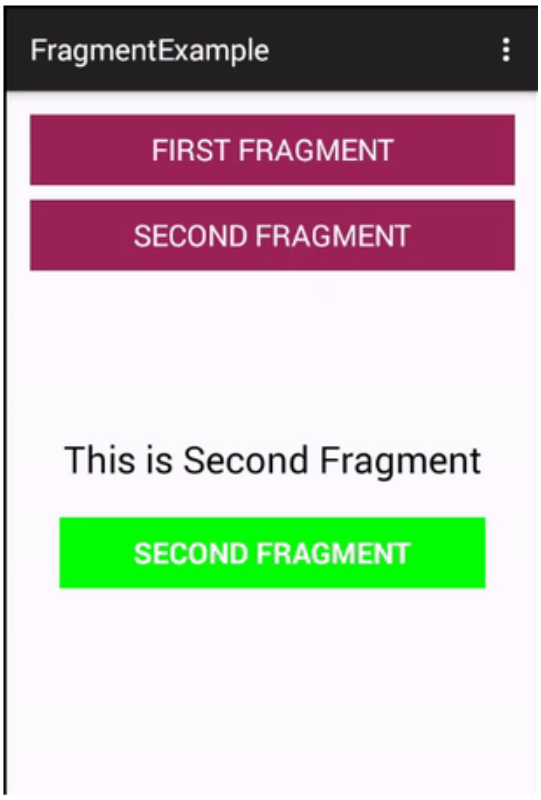
1. FragmentActivity: The base class for all activities using compatibility based Fragment (and loader) features.

2. Fragment: The base class for all Fragment definitions

3. FragmentManager: The class for interacting with Fragment objects inside an activity

4. FragmentTransaction: The class for performing an atomic set of Fragment operations such as Replace or Add a Fragment.





Fragment Example 1 In Android Studio:

Below is the example of Fragment's. In this example we create two Fragments and load them on the click of Button's. We display two Button's and a FrameLayout in our Activity and perform setOnClickListener event on both Button's. On the click of First Button we replace the First Fragment and on click of Second Button we replace the Second Fragment with the layout(FrameLayout). In the both Fragment's we display a TextView and a Button and onclick of Button we display the name of the Fragment with the help of Toast.

Below you can download code, see final output and read step by step explanation:

Step 1: Create a new project and name it FragmentExample

Step 2: Open res -> layout ->activity_main.xml (or) main.xml and add following code:

```
<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="vertical"
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">
<!-- display two Button's and a FrameLayout to replace the Fragment's  -->
<Button
android:id="@+id/firstFragment"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:background="@color/button_background_color"
android:text="First Fragment"
android:textColor="@color/white"
android:textSize="20sp" />

<Button
android:id="@+id/secondFragment"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginTop="10dp"
android:background="@color/button_background_color"
android:text="Second Fragment"
android:textColor="@color/white"
android:textSize="20sp" />

<FrameLayout
android:id="@+id/frameLayout"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_marginTop="10dp" />
```

```
</LinearLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code for initiate the Button's. After that we perform setOnClickListener event on both Button's. On the click of First Button we replace the First Fragment and on click of Second Button we replace the Second Fragment with the layout(FrameLayout). For replacing a Fragment with FrameLayout firstly we create a Fragment Manager and then begin the transaction using Fragment Transaction and finally replace the Fragment with the layout i.e FrameLayout.

```
package com.abhiandroid.fragmentexample;


import android.app.Fragment;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {


Button firstFragment, secondFragment;


@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
// get the reference of Button's
firstFragment = (Button) findViewById(R.id.firstFragment);
secondFragment = (Button) findViewById(R.id.secondFragment);


// perform setOnClickListener event on First Button
firstFragment.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
```

```
// load First Fragment
loadFragment(new FirstFragment());
}
});
// perform setOnClickListener event on Second Button
secondFragment.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// load Second Fragment
loadFragment(new SecondFragment());
}
});

}

private void loadFragment(Fragment fragment) {
// create a FragmentManager
FragmentManager fm = getFragmentManager();
// create a FragmentTransaction to begin the transaction and replace the Fragment
FragmentTransaction fragmentTransaction = fm.beginTransaction();
// replace the FrameLayout with new Fragment
fragmentTransaction.replace(R.id.frameLayout, fragment);
fragmentTransaction.commit(); // save the changes
}
}
```

Step 4: Now we need 2 fragments and 2 xml layouts. So create two fragments by right click on your package folder and create classes and name them as FirstFragment and SecondFragment and add the following code respectively.

FirstFragment.class

In this Fragment firstly we inflate the layout and get the reference of Button. After that we perform setOnClickListener event on Button so whenever a user click on the button a message "First Fragment" is displayed on the screen by using a Toast.

```
package com.abhiandroid.fragmentexample;
```

```
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.Toast;

public class FirstFragment extends Fragment {

View view;
Button firstButton;

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
view = inflater.inflate(R.layout.fragment_first, container, false);
// get the reference of Button
firstButton = (Button) view.findViewById(R.id.firstButton);
// perform setOnClickListener on first Button
firstButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// display a message by using a Toast
Toast.makeText(getActivity(), "First Fragment", Toast.LENGTH_LONG).show();
}
});
return view;
}
}
```

SecondFragment.class

In this Fragment firstly we inflate the layout and get the reference of Button. After that we perform setOnClickListener event on Button so whenever a user click on the button a message “Second Fragment“ is displayed on the screen by using a Toast.

```
package com.abhiandroid.fragmentexample;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.Toast;

public class SecondFragment extends Fragment {
View view;
Button secondButton;

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
view = inflater.inflate(R.layout.fragment_second, container, false);
// get the reference of Button
secondButton = (Button) view.findViewById(R.id.secondButton);
// perform setOnClickListener on second Button
secondButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// display a message by using a Toast
Toast.makeText(getActivity(), "Second Fragment", Toast.LENGTH_LONG).show();
}
});
return view;
```

```
}
}
```

Step 5: Now create 2 xml layouts by right clicking on res/layout -> New -> Layout Resource File and name them as fragment_first and fragment_second and add the following code in respective files.

Here we will design the basic simple UI by using TextView and Button in both xml's.

fragment_first.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"
tools:context="com.abhiandroid.fragmentexample.FirstFragment">

<!--TextView and Button displayed in First Fragment -->
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:layout_marginTop="100dp"
android:text="This is First Fragment"
android:textColor="@color/black"
android:textSize="25sp" />

<Button
android:id="@+id/firstButton"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:layout_marginLeft="20dp"
android:layout_marginRight="20dp"
android:background="@color/green"
android:text="First Fragment"
android:textColor="@color/white"
```

```
android:textSize="20sp"
android:textStyle="bold" />
</RelativeLayout>
```

fragment_second.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"
tools:context="com.abhiandroid.fragmentexample.SecondFragment">

<!--TextView and Button displayed in Second Fragment -->
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:layout_marginTop="100dp"
android:text="This is Second Fragment"
android:textColor="@color/black"
android:textSize="25sp" />

<Button
android:id="@+id/secondButton"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:layout_marginLeft="20dp"
android:layout_marginRight="20dp"
android:background="@color/green"
android:text="Second Fragment"
android:textColor="@color/white"
android:textSize="20sp"
android:textStyle="bold" />
```

```
</RelativeLayout>
```

Step 6: Open res ->values ->colors.xml

In this step we define the color's that used in our xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<!-- color's used in our project -->
<color name="black">#000</color>
<color name="green">#0f0</color>
<color name="white">#fff</color>
<color name="button_background_color">#925</color>
</resources>
```

Step 7: Open AndroidManifest.xml

In this step we show the Android Manifest file in which do nothing because we need only one Activitty i.e MainActivity which is already defined in it. In our project we create two Fragment's but we don't need to define the Fragment's in manifest because Fragment is a part of an Activity.

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

<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme" >
<activity
android:name=".MainActivity"
android:label="@string/app_name" >
<intent-filter>
<action android:name="android.intent.action.MAIN" />

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

```
</intent-filter>
</activity>
</application>
</manifest>
```

Step 8: Now run the App and you will two button. Clicking on first button shows First Fragment and on click of Second Button shows the Second Fragment which is actually replacing layout (FrameLayout).

Android Service

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 has 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).

Features of Service

- Service is an Android Component without an UI
- It is used to perform long running operations in background. Services run indefinitly unless they are explicitly stopped or destroyed
- It can be started by any other application component. Components can even infact bind to a service to perform Interprocess- Comminication
- It can still be running even if the application is killed unless it stops itself by calling *stopself*() or is stopped by a Android component by calling *stopService*().
- If not stopped it goes on running unless is terminated by Android due to resource shortage
- The android.app.Service is subclass of ContextWrapper class.

Android platform service

The Android platform provides and runs predefined system services and every Android application can use them, given the right permissions. These system services are usually exposed via a specific Manager class. Access to them can be gained via the getSystemService() method.

Permission

The purpose of a *permission* is to protect the privacy of an Android user. Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet). Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request.

Add permissions to the manifest

On all versions of Android, to declare that your app needs a permission, put a <uses-permission> element in your app manifest, as a child of the top-level <manifest> element. For example, an app that needs to access the internet would have this line in the manifest:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.example.snazzyapp">

   <uses-permission android:name="android.permission.INTERNET"/>
   <!-- other permissions go here -->

   <application ...>
      ...
   </application>
</manifest>
```

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.

1. Started
2. Bound

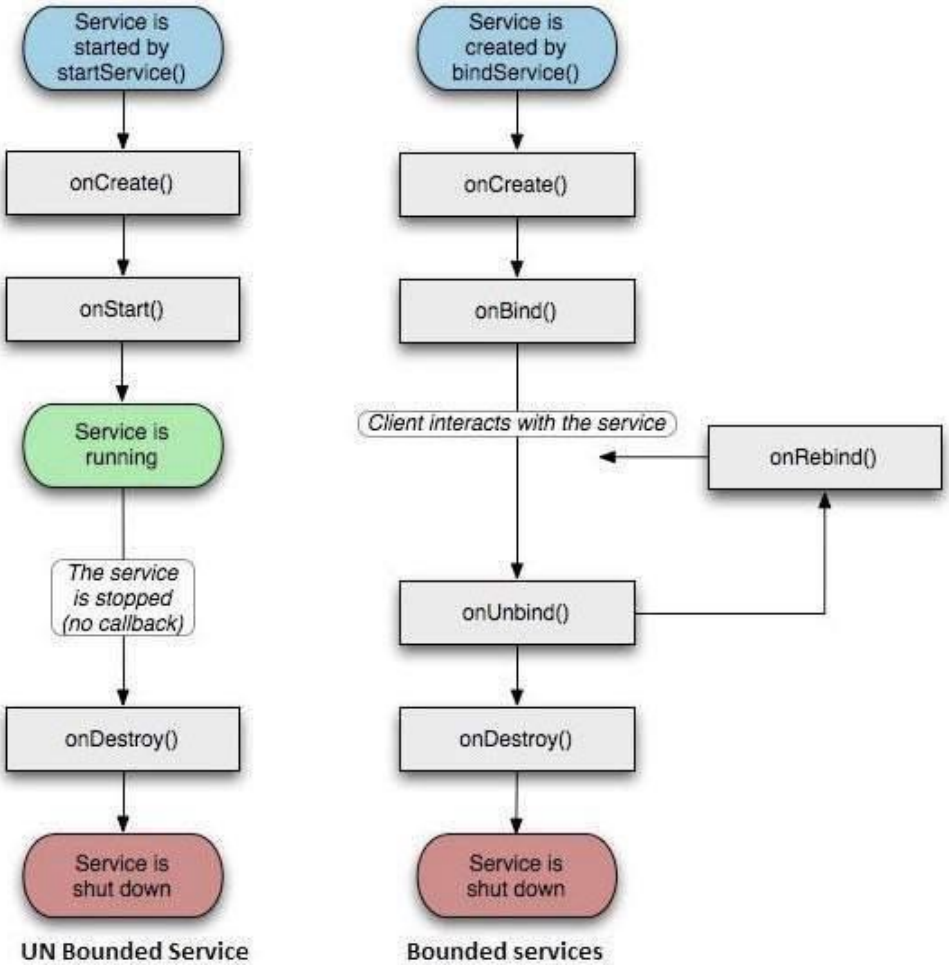
1) Started Service

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.



Like any other components service also has callback methods. These will be invoked while the service is running to inform the application of its state. Implementing these in your custom service would help you in performing the right operation in the right state.

onCreate()

This is the first callback which will be invoked when any component starts the service. If the same service is called again while it is still running this method wont be invoked. Ideally one time setup and intializing should be done in this callback.

onStartCommand()

This callback is invoked when service is started by any component by calling startService(). It basically indicates that the service has started and can now run indefinetly.

onBind()

This is invoked when any component starts the service by calling *onBind*.

onUnbind()

This is invoked when all the clients are disconnected from the service.

onRebind()

This is invoked when new clients are connected to the service. It is called after onUnbind

onDestroy()

This is a final clean up call from the system. This is invoked just before the service is being destroyed. Could be very useful to cleanup any resources such as threads, registered listeners, or receivers.

Android 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.

activity_main.xml

Drag the 3 buttons from the pallete, now the activity_main.xml will look like this:

File: activity_main.xml

```
<?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.javatpoint.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>
```

activity_next.xml

It is the layout file of next activity.

File: activity_next.xml

It contains only one textview displaying the message Next Page

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

Service class

Now create the service implemenation class by inheriting the Service class and overridding its callback methods.

File: MyService.java

```
package example.javatpoint.com.androidservice;

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();
   }
}
```

Activity class

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.

File: MainActivity.java

```
package example.javatpoint.com.androidservice;

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;
      }
   }
}
```

NextPage class

Now, create another activity.

File: NextPage.java

```
package example.javatpoint.com.androidservice;
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);
  }
}
```

Declare the Service in the AndroidManifest.xml file

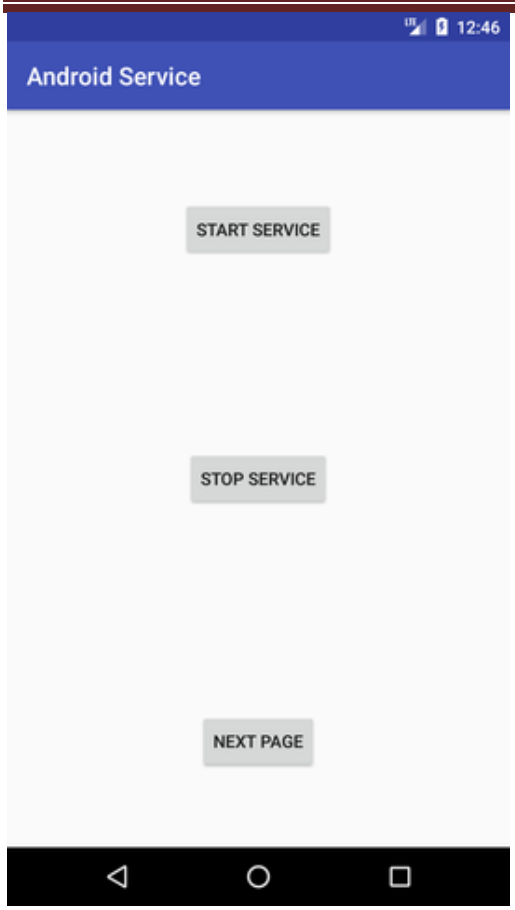
Finally, declare the service in the manifest file.

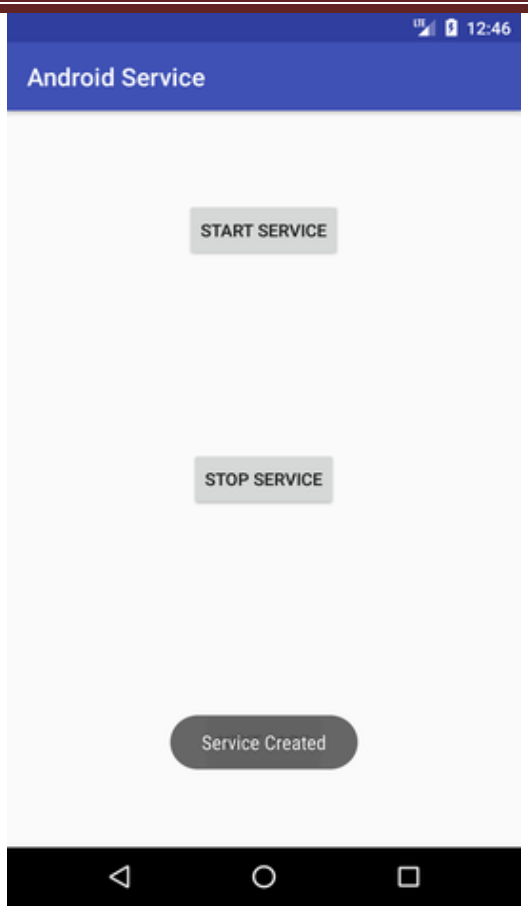
File: AndroidManifest.xml

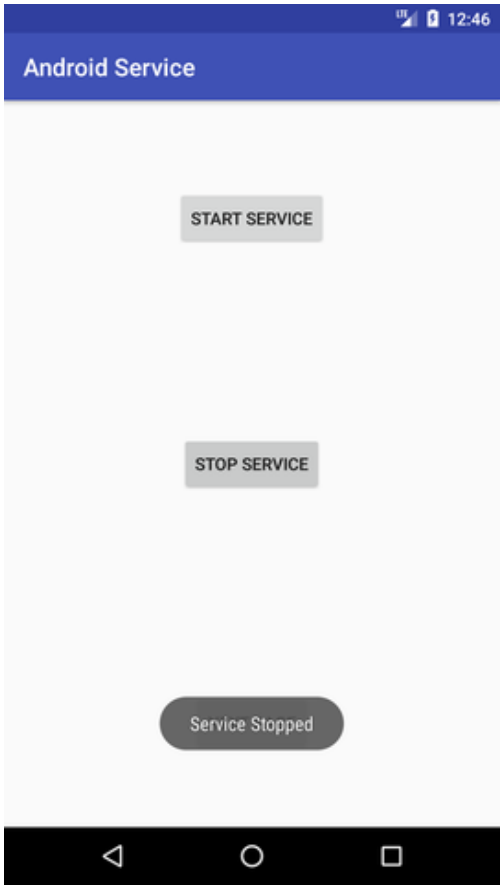
Let's see the complete AndroidManifest.xml file

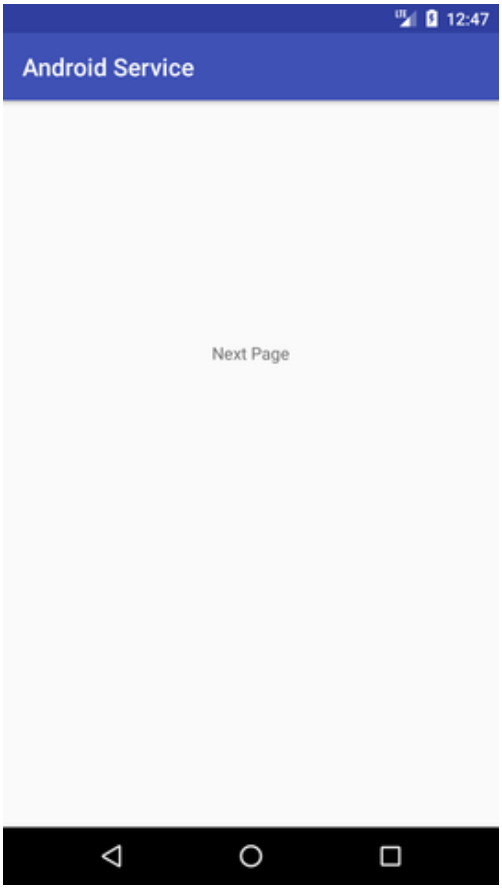
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="example.javatpoint.com.androidservice">

  <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>
```

Output:


12:46
Android Service
START SERVICE
STOP SERVICE
NEXT PAGE


12:46
Android Service
START SERVICE
STOP SERVICE
Service Created


12:46
Android Service
START SERVICE
STOP SERVICE
Service Stopped

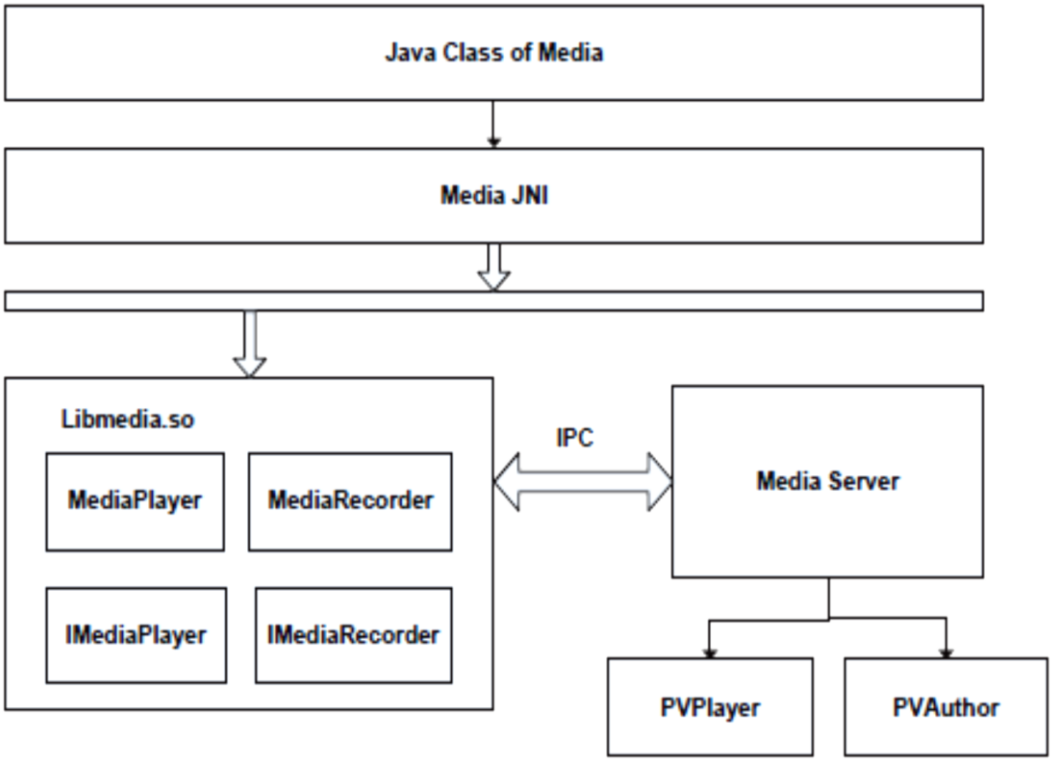

12:47
Android Service
Next Page

Android System Architecture

The android system consists of five layers and each layer consists of some core components. Figure 1 shows the architecture of android. From top to down, the core components are: Applications, Application Framework, Native C libraries, Android Runtime Environment (JVM), HAL (Hardware Abstract Layer), Linux Kernel.
1) Applications. Application layer consists of many core Java-based applications, such as calendar, web browser, SMS application, E-mail, etc.
2) Application Framework. Application framework consists of many components and Java classes to allow android application developers to develop various kinds of applications. By using Java language, it hides the internal implementation of system core functions and provides the developers an easy-use API. Basically, it includes Java core class and some special components of android. Some typical components are as follows: View (List, Grids), Content Provider, Resource Manager, Activity Manager.
3) Native C Libraries. In Native C library layer, it consists of many C/C++ libraries. And the core functions of android are implemented by those libraries. Some typical core libraries are as follows: Bionic C lib, OpenCore, SQLite, Surface Manager, WebKit, 3D library.
4) Android Runtime Environment. Runtime environment consists of Dalvik Java virtual machine and some implementations of Java core libraries.
5) HAL. This layer abstracts different kinds of hardwares and provides an unified program interface to Native C libraries. HAL can make Android port on different platforms more easily.
6) Linux Kernel. Android's core system functions (e.g., safety management, RAM management, process management, network stack) depend on Linux kernels.

Android Multimedia framework

The android multimedia system includes multimedia applications, multimedia framework, OpenCore engine and hardware abstract for audio/video input/output devices. And the goal of the android multimedia framework is to provide a consistent interface for Java services. The multimedia framework consists of several core dynamic libraries such as libmediajni, libmedia, libmediaplayservice and so on. A general multimedia framework architecture is shown in Figure



Java classes call the Native C library Libmedia through Java JNI (Java Native Interface). Libmedia library communicates with Media Server guard process through Android's Binder IPC (inter process communication) mechanism. Media Server process creates the corresponding multimedia service according to the Java multimedia applications. The whole communication between Libmedia and Media Server forms a Client/Server model. In Media Server guard process, it calls OpenCore multimedia engine to realize the specific multimedia processing functions. And the OpenCore engine refers to the PVPlayer and PVAuthor.

Play Audio and Video

We can play and control the audio files in android by the help of **MediaPlayer class**.

Here, we are going to see a simple example to play the audio file. In the next page, we will see the example to control the audio playback like start, stop, pause etc.

MediaPlayer class

The **android.media.MediaPlayer** class is used to control the audio or video files.

Methods of MediaPlayer class

There are many methods of MediaPlayer class. Some of them are as follows:

Method	Description
public void setDataSource(String path)	Sets the data source (file path or http url) to use.
public void prepare()	Prepares the player for playback synchronously.

public void start()	It starts or resumes the playback.
public void stop()	It stops the playback.
public void pause()	It pauses the playback.
public boolean isPlaying()	Checks if media player is playing.
public void seekTo(int millis)	Seeks to specified time in miliseconds.
public void setLooping(boolean looping)	Sets the player for looping or non-looping.
public boolean isLooping()	Checks if the player is looping or non-looping.
public void selectTrack(int index)	It selects a track for the specified index.
public int getCurrentPosition()	Returns the current playback position.
public int getDuration()	Returns duration of the file.
public void setVolume(float leftVolume,float rightVolume)	Sets the volume on this player.

Activity class

Let's write the code of to play the audio file. Here, we are going to play maine.mp3 file located inside the sdcard/Music directory.

File: MainActivity.java

```
package com.example.audiomediaplayer1;

import android.media.MediaPlayer;
import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.MediaController;
import android.widget.VideoView;

public class MainActivity extends Activity {

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

    MediaPlayer mp=new MediaPlayer();
    try{
      mp.setDataSource("/sdcard/Music/maine.mp3");//Write your location here
      mp.prepare();
      mp.start();

    }catch(Exception e){e.printStackTrace();}

  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
```

```
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }

}
```

Android MediaPlayer Example of controlling the audio

Let's see a simple example to start, stop and pause the audio play.

activity_main.xml

Drag three buttons from pallete to start, stop and pause the audio play. Now the xml file will look like this:

File: MainActivity.java

```
<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_alignParentTop="true"
    android:layout_marginTop="30dp"
    android:text="Audio Controller" />

  <Button
    android:id="@+id/button1"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_below="@+id/textView1"
    android:layout_marginTop="48dp"
    android:text="start" />
```

```
  <Button
    android:id="@+id/button2"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/button1"
    android:layout_toRightOf="@+id/button1"
    android:text="pause" />

  <Button
    android:id="@+id/button3"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/button2"
    android:layout_toRightOf="@+id/button2"
    android:text="stop" />

</RelativeLayout>
```

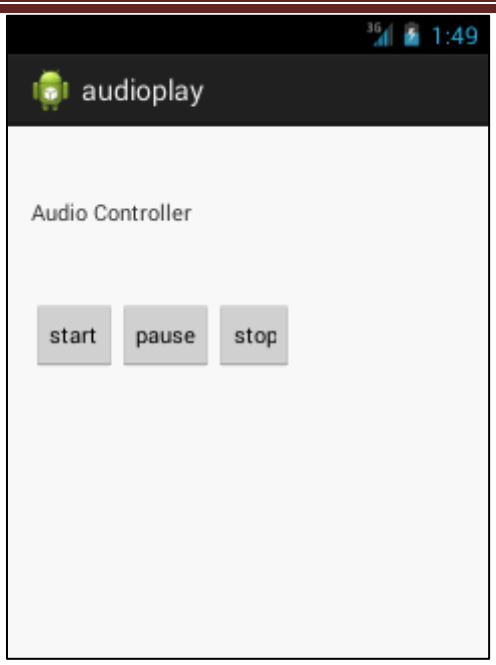
Activity class

Let's write the code to start, pause and stop the audio player.

File: MainActivity.java

```
package com.example.audioplay;

import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.Environment;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity {
  Button start,pause,stop;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```
    start=(Button)findViewById(R.id.button1);
    pause=(Button)findViewById(R.id.button2);
    stop=(Button)findViewById(R.id.button3);
    //creating media player
    final MediaPlayer mp=new MediaPlayer();
    try{
        //you can change the path, here path is external directory(e.g. sdcard) /Music/maine
.mp3
    mp.setDataSource(Environment.getExternalStorageDirectory().getPath()+"/Music/main
e.mp3");

    mp.prepare();
    }catch(Exception e){e.printStackTrace();}

    start.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        mp.start();
      }
    });
    pause.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        mp.pause();
      }
    });
    stop.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        mp.stop();
      }
    });
  }
}
```



Android Video Player Example

By the help of **MediaController** and **VideoView** classes, we can play the video files in android.

MediaController class

The **android.widget.MediaController** is a view that contains media controls like play/pause, previous, next, fast-forward, rewind etc.

VideoView class

The **android.widget.VideoView** class provides methods to play and control the video player. The commonly used methods of VideoView class are as follows:

Method	Description
public void setMediaController(MediaController controller)	Sets the media controller to the video view.
public void setVideoURI (Uri uri)	Sets the URI of the video file.
public void start()	Starts the video view.
public void stopPlayback()	Stops the playback.
public void pause()	Pauses the playback.
public void suspend()	Suspends the playback.
public void resume()	Resumes the playback.
public void seekTo(int millis)	Seeks to specified time in miliseconds.

activity_main.xml

Drag the VideoView from the pallete, now the activity_main.xml file will like this:

File: 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"
  tools:context=".MainActivity" >

  <VideoView
    android:id="@+id/videoView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_centerVertical="true" />

</RelativeLayout>
```

Activity class

Let's write the code of to play the video file. Here, we are going to play 1.mp4 file located inside the sdcard/media directory.

File: MainActivity.java

```
package com.example.video1;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.MediaController;
import android.widget.VideoView;

public class MainActivity extends Activity {

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

    VideoView videoView =(VideoView)findViewById(R.id.videoView1);
```

```
        //Creating MediaController
    MediaController mediaController= new MediaController(this);
      mediaController.setAnchorView(videoView);

       //specify the location of media file
      Uri uri=Uri.parse(Environment.getExternalStorageDirectory().getPath()+"/media/1.m
p4");

       //Setting MediaController and URI, then starting the videoView
      videoView.setMediaController(mediaController);
      videoView.setVideoURI(uri);
      videoView.requestFocus();
      videoView.start();

  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.activity_main, menu);
    return true;
  }

}
```

Text To Speech

Android allows you convert your text into voice. Not only you can convert it but it also allows you to speak text in variety of different languages.

Android provides **TextToSpeech** class for this purpose. In order to use this class, you need to instantiate an object of this class and also specify the **initListener**. Its syntax is given below –

```
private EditText write;
ttobj=new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
  @Override
  public void onInit(int status) {
  }
});
```

In this listener, you have to specify the properties for TextToSpeech object , such as its language ,pitch e.t.c. Language can be set by calling **setLanguage()** method. Its syntax is given below –

```
ttobj.setLanguage(Locale.UK);
```

The method setLanguage takes an Locale object as parameter. The list of some of the locales available are given below –

Sr.No.	Locale
1	US
2	CANADA_FRENCH
3	GERMANY
4	ITALY
5	JAPAN
6	CHINA

Once you have set the language, you can call **speak** method of the class to speak the text. Its syntax is given below –

```
ttobj.speak(toSpeak, TextToSpeech.QUEUE_FLUSH, null);
```

Apart from the speak method, there are some other methods available in the TextToSpeech class. They are listed below –

Sr.No	Method & description
1	**addSpeech(String text, String filename)** This method adds a mapping between a string of text and a sound file.
2	**getLanguage()** This method returns a Locale instance describing the language.
3	**isSpeaking()** This method checks whether the TextToSpeech engine is busy speaking.
4	**setPitch(float pitch)** This method sets the speech pitch for the TextToSpeech engine.
5	**setSpeechRate(float speechRate)** This method sets the speech rate.
6	**shutdown()** This method releases the resources used by the TextToSpeech engine.
7	**stop()** This method stop the speak.

Example

The below example demonstrates the use of TextToSpeech class. It crates a basic application that allows you to set write text and speak it.

To experiment with this example , you need to run this on an actual device.

Steps	Description
1	You will use Android studio to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add TextToSpeech code.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/MainActivity.java**.

```
package com.example.sairamkrishna.myapplication;
```

```
import android.app.Activity;
import android.os.Bundle;
import android.speech.tts.TextToSpeech;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import java.util.Locale;
import android.widget.Toast;

public class MainActivity extends Activity {
  TextToSpeech t1;
  EditText ed1;
  Button b1;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ed1=(EditText)findViewById(R.id.editText);
    b1=(Button)findViewById(R.id.button);

    t1=new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
      @Override
      public void onInit(int status) {
        if(status != TextToSpeech.ERROR) {
          t1.setLanguage(Locale.UK);
        }
      }
    });

    b1.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        String toSpeak = ed1.getText().toString();
        Toast.makeText(getApplicationContext(), toSpeak,Toast.LENGTH_SHORT).show();
        t1.speak(toSpeak, TextToSpeech.QUEUE_FLUSH, null);
      }
    });
  }

  public void onPause(){
    if(t1 !=null){
      t1.stop();
      t1.shutdown();
    }
    super.onPause();
  }
}
```

Here is the content of **activity_main.xml**

In the following code **abc**indicates the logo of tutorialspoint.com

```
<?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"
  tools:context=".MainActivity"
  android:transitionGroup="true">

  <TextView android:text="Text to Speech" android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textview"
    android:textSize="35dp"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials point"
    android:id="@+id/textView"
    android:layout_below="@+id/textview"
    android:layout_centerHorizontal="true"
    android:textColor="#ff7aff24"
    android:textSize="35dp" />

  <ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageView"
    android:src="@drawable/abc"
    android:layout_below="@+id/textView"
    android:layout_centerHorizontal="true"
    android:theme="@style/Base.TextAppearance.AppCompat" />

  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:layout_below="@+id/imageView"
    android:layout_marginTop="46dp"
    android:hint="Enter Text"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:layout_alignParentLeft="true"
```

```
    android:layout_alignParentStart="true"
    android:textColor="#ff7aff10"
    android:textColorHint="#ffff23d1" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text to Speech"
    android:id="@+id/button"
    android:layout_below="@+id/editText"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="46dp" />

</RelativeLayout>
```

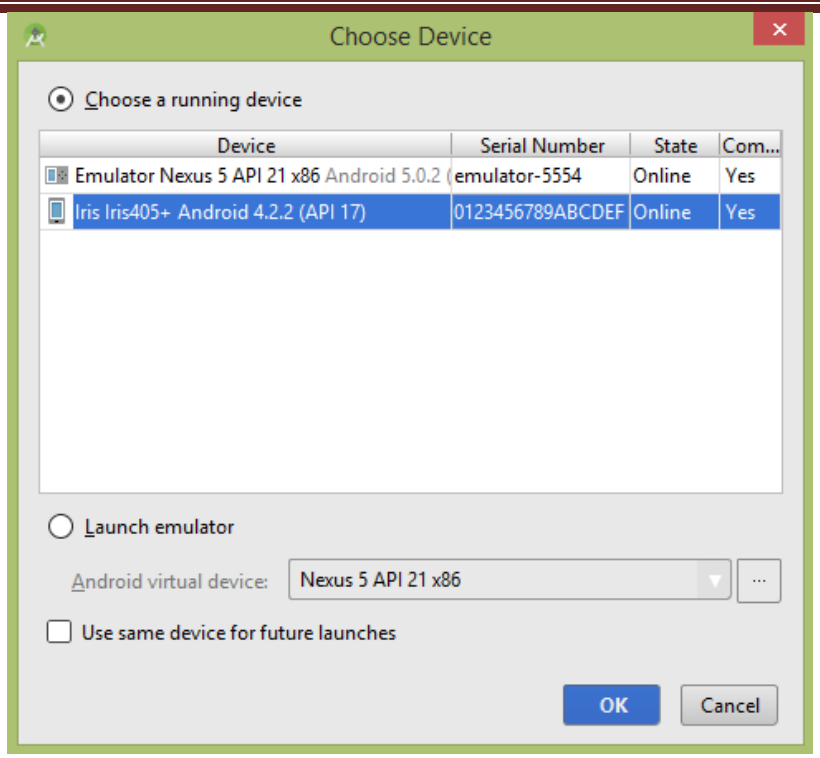
Here is the content of **Strings.xml**.

```
<resources>
  <string name="app_name">My Application</string>
</resources>
```

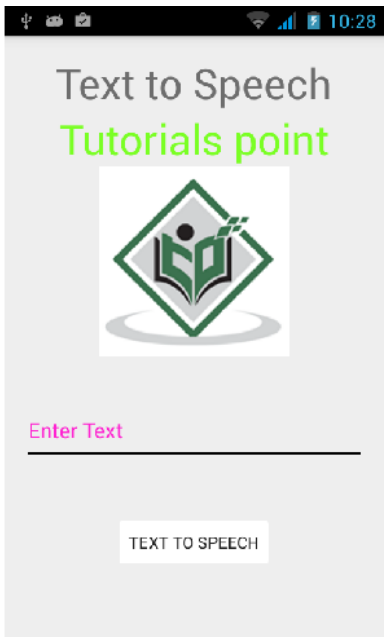
Here is the content of **AndroidManifest.xml**

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

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

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

    </activity>

  </application>
</manifest>
```

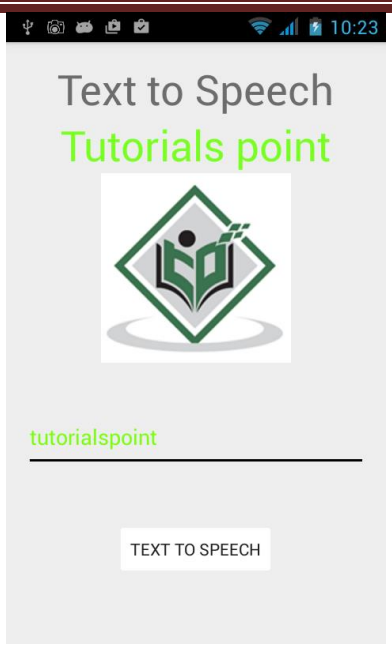
Let's try to run your application. I assume you have connected your actual Android Mobile device with your computer. To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Before starting your application, android studio will display following window to select an option where you want to run your Android application.



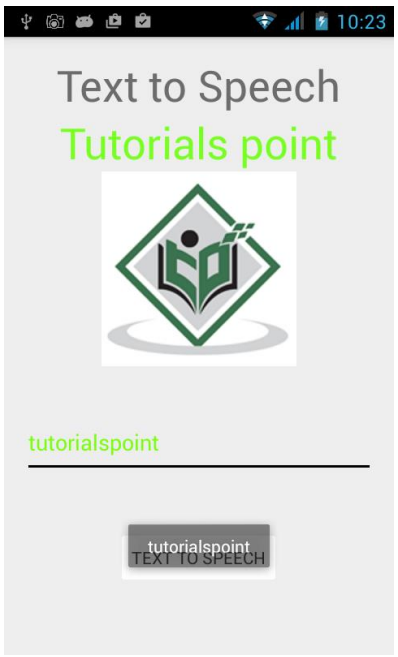
Select your mobile device as an option and then check your mobile device which will display following screen.



Now just type some text in the field and click on the text to speech button below. A notification would appear and text will be spoken. It is shown in the image below –



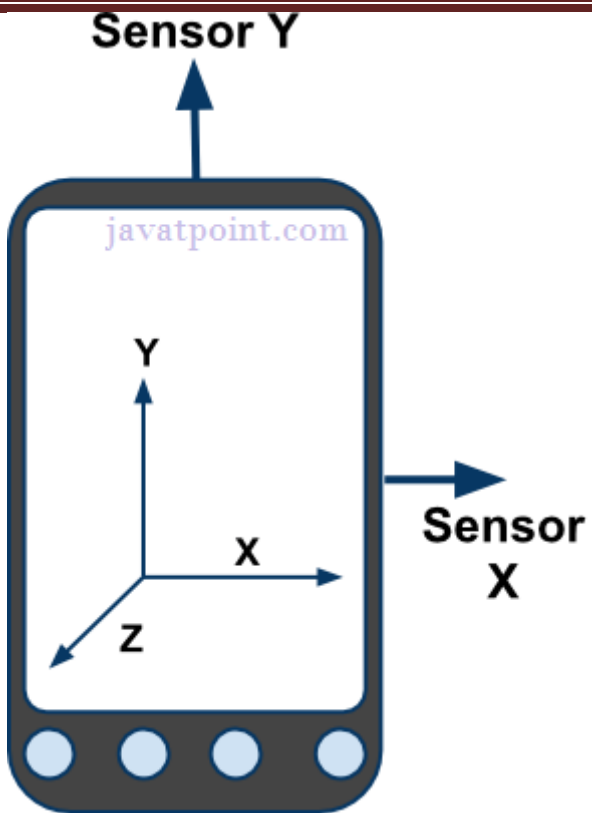
Now type something else and repeat the step again with different locale. You will again hear sound. This is shown below –



Android Sensor

Sensors can be used to monitor the three-dimensional device movement or change in the environment of the device.

Android provides sensor api to work with different types of sensors.



Types of Sensors

Android supports three types of sensors:

1) Motion Sensors

These are used to measure acceleration forces and rotational forces along with three axes.

2) Position Sensors

These are used to measure the physical position of device.

3) Environmental Sensors

These are used to measure the environmental changes such as temperature, humidity etc.

Android Sensor API

Android sensor api provides many classes and interface. The important classes and interfaces of sensor api are as follows:

1) SensorManager class

The **android.hardware.SensorManager** class provides methods :

- to get sensor instance,

- to access and list sensors,
- to register and unregister sensor listeners etc.

You can get the instance of SensorManager by calling the method getSystemService() and passing the SENSOR_SERVICE constant in it.

SensorManager sm = (SensorManager)getSystemService(SENSOR_SERVICE);

2) Sensor class

The **android.hardware.Sensor** class provides methods to get information of the sensor such as sensor name, sensor type, sensor resolution, sensor type etc.

3) SensorEvent class

Its instance is created by the system. It provides information about the sensor.

4) SensorEventListener interface

It provides two call back methods to get information when sensor values (x,y and z) change or sensor accuracy changes.

Public and abstract methods	Description
void onAccuracyChanged(Sensor sensor, int accuracy)	it is called when sensor accuracy is changed.
void onSensorChanged(SensorEvent event)	it is called when sensor values are changed.

Android simple sensor app example

Let's see the two sensor examples.

1. A sensor example that prints x, y and z axis values. Here, we are going to see that.
2. A sensor example that changes the background color when device is shuffled. Click for changing background color of activity sensor example

activity_main.xml

There is only one textview in this file.

File: 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"
  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="92dp"
    android:layout_marginTop="114dp"
    android:text="TextView" />

</RelativeLayout>
```

Activity class

Let's write the code that prints values of x axis, y axis and z axis.

File: MainActivity.java

```
package com.example.sensorsimple;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;
import android.hardware.SensorManager;
import android.hardware.SensorEventListener;
import android.hardware.SensorEvent;
import android.hardware.Sensor;
import java.util.List;
public class MainActivity extends Activity {
  SensorManager sm = null;
  TextView textView1 = null;
  List list;

  SensorEventListener sel = new SensorEventListener(){
    public void onAccuracyChanged(Sensor sensor, int accuracy) {}
    public void onSensorChanged(SensorEvent event) {
      float[] values = event.values;
      textView1.setText("x: "+values[0]+"\ny: "+values[1]+"\nz: "+values[2]);
    }
  };

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

```
    /* Get a SensorManager instance */
    sm = (SensorManager)getSystemService(SENSOR_SERVICE);

    textView1 = (TextView)findViewById(R.id.textView1);

    list = sm.getSensorList(Sensor.TYPE_ACCELEROMETER);
    if(list.size()>0){
      sm.registerListener(sel, (Sensor) list.get(0), SensorManager.SENSOR_DELAY_NORMAL
);
    }else{
      Toast.makeText(getBaseContext(), "Error: No Accelerometer.", Toast.LENGTH_LONG).s
how();
    }
  }

  @Override
  protected void onStop() {
    if(list.size()>0){
     sm.unregisterListener(sel);
    }
    super.onStop();
  }
}
```

Output:

Android Sensor Example

In this example, we are going to create a sensor application that changes the background color of activity when device is shaked.

For understanding about sensor basics, visit the previous page that provides details about sensor api.

Android Sensor Example

activity_main.xml

File: 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"
   tools:context=".MainActivity" >

  <TextView
      android:id="@+id/textView"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:text="Shake to switch color" />

</RelativeLayout>
```

Activity class

File: MainActivity.java

```
package com.example.sensor;

import android.app.Activity;
import android.graphics.Color;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends Activity implements SensorEventListener{
   private SensorManager sensorManager;
    private boolean isColor = false;
```

```
private View view;
private long lastUpdate;

@Override
public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 view = findViewById(R.id.textView);
 view.setBackgroundColor(Color.GREEN);

 sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
 lastUpdate = System.currentTimeMillis();
}
//overriding two methods of SensorEventListener
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {}
@Override
public void onSensorChanged(SensorEvent event) {
 if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
  getAccelerometer(event);
 }

}

private void getAccelerometer(SensorEvent event) {
 float[] values = event.values;
 // Movement
 float x = values[0];
 float y = values[1];
 float z = values[2];

 float accelationSquareRoot = (x * x + y * y + z * z)
    / (SensorManager.GRAVITY_EARTH * SensorManager.GRAVITY_EARTH);

 long actualTime = System.currentTimeMillis();
 Toast.makeText(getApplicationContext(),String.valueOf(accelationSquareRoot)+" "+
         SensorManager.GRAVITY_EARTH,Toast.LENGTH_SHORT).show();

 if (accelationSquareRoot >= 2) //it will be executed if you shuffle
 {

  if (actualTime - lastUpdate < 200) {
   return;
  }
  lastUpdate = actualTime;//updating lastUpdate for next shuffle
```

```
        if (isColor) {
         view.setBackgroundColor(Color.GREEN);

        } else {
         view.setBackgroundColor(Color.RED);
        }
        isColor = !isColor;
       }
      }

      @Override
      protected void onResume() {
       super.onResume();
       // register this class as a listener for the orientation and
       // accelerometer sensors
       sensorManager.registerListener(this,sensorManager.getDefaultSensor(Sensor.TYPE_AC
CELEROMETER),
                       SensorManager.SENSOR_DELAY_NORMAL);
      }

      @Override
      protected void onPause() {
       // unregister listener
       super.onPause();
       sensorManager.unregisterListener(this);
      }
     }
```

AsyncTask

Android AsyncTask is an abstract class provided by Android which gives us the liberty to perform heavy tasks in the background and keep the UI thread light thus making the application more responsive.

Android application runs on a single thread when launched. Due to this single thread model tasks that take longer time to fetch the response can make the application non-responsive. To avoid this we use android AsyncTask to perform the heavy tasks in background on a dedicated thread and passing the results back to the UI thread. Hence use of AsyncTask in android application keeps the UI thread responsive at all times.

The basic methods used in an android AsyncTask class are defined below :

-

The three generic types used in an android AsyncTask class are given below :

- **Params** : The type of the parameters sent to the task upon execution
- **Progress** : The type of the progress units published during the background computation
- **Result** : The type of the result of the background computation

Android AsyncTask Example

To start an AsyncTask the following snippet must be present in the MainActivity class :

```
MyTask myTask = new MyTask();
myTask.execute();
```

In the above snippet we've used a sample classname that extends AsyncTask and execute method is used to start the background thread.

Note:

- The AsyncTask instance must be created and invoked in the UI thread.
- The methods overridden in the AsyncTask class should never be called. They're called automatically
- AsyncTask can be called only once. Executing it again will throw an exception

In this tutorial we'll implement an AsyncTask that makes a process to go to sleep for a given period of time as set by the user.

Android AsyncTask Example Code

The xml layout is defined in the activity_main.xml and its given below:

activity_main.xml

```
<RelativeLayout xmlns:android="https://schemas.android.com/apk/res/android"
  xmlns:tools="https://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity" >

  <TextView
    android:id="@+id/tv_time"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="10pt"
    android:textColor="#444444"
    android:layout_alignParentLeft="true"
    android:layout_marginRight="9dip"
    android:layout_marginTop="20dip"
    android:layout_marginLeft="10dip"
    android:text="Sleep time in Seconds:"/>
  <EditText
    android:id="@+id/in_time"
    android:layout_width="150dip"
    android:layout_height="wrap_content"
    android:background="@android:drawable/editbox_background"
    android:layout_toRightOf="@id/tv_time"
    android:layout_alignTop="@id/tv_time"
    android:inputType="number"
    />
  <Button
    android:id="@+id/btn_run"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Run Async task"
    android:layout_below="@+id/in_time"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="64dp" />
  <TextView
    android:id="@+id/tv_result"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="7pt"
    android:layout_below="@+id/btn_run"
    android:layout_centerHorizontal="true" />
</RelativeLayout>
```

In the above layout we've used a predefined drawable as the border of the EditText.

MainActivity.java

```
package com.journaldev.asynctask;

import android.app.ProgressDialog;
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;

public class MainActivity extends AppCompatActivity {
   private Button button;
   private EditText time;
   private TextView finalResult;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      time = (EditText) findViewById(R.id.in_time);
      button = (Button) findViewById(R.id.btn_run);
      finalResult = (TextView) findViewById(R.id.tv_result);
      button.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            AsyncTaskRunner runner = new AsyncTaskRunner();
            String sleepTime = time.getText().toString();
            runner.execute(sleepTime);
         }
      });
   }

   private class AsyncTaskRunner extends AsyncTask<String, String, String> {

      private String resp;
      ProgressDialog progressDialog;

      @Override
      protected String doInBackground(String... params) {
         publishProgress("Sleeping..."); // Calls onProgressUpdate()
         try {
            int time = Integer.parseInt(params[0])*1000;

            Thread.sleep(time);
            resp = "Slept for " + params[0] + " seconds";
         } catch (InterruptedException e) {
```

```
            e.printStackTrace();
            resp = e.getMessage();
        } catch (Exception e) {
            e.printStackTrace();
            resp = e.getMessage();
        }
        return resp;
    }


    @Override
    protected void onPostExecute(String result) {
        // execution of result of Long time consuming operation
        progressDialog.dismiss();
        finalResult.setText(result);
    }


    @Override
    protected void onPreExecute() {
        progressDialog = ProgressDialog.show(MainActivity.this,
                "ProgressDialog",
                "Wait for "+time.getText().toString()+ " seconds");
    }


    @Override
    protected void onProgressUpdate(String... text) {
        finalResult.setText(text[0]);

    }
  }
}
```

In the above code we've used AsyncTaskRunner class to perform the AsyncTask operations. The time in seconds is passed as a parameter to the class and a ProgressDialog is displayed for the given amount of time.

The images given below are the outputs produced by the project where the time set by the user is 5 seconds.

Audio Capture

Android has a built in microphone through which you can capture audio and store it , or play it in your phone. There are many ways to do that but the most common way is through MediaRecorder class.

Android provides MediaRecorder class to record audio or video. In order to use MediaRecorder class ,you will first create an instance of MediaRecorder class. Its syntax is given below.

```
MediaRecorder myAudioRecorder = new MediaRecorder();
```

Now you will set the source , output and encoding format and output file. Their syntax is given below.

```
myAudioRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
myAudioRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
myAudioRecorder.setAudioEncoder(MediaRecorder.OutputFormat.AMR_NB);
myAudioRecorder.setOutputFile(outputFile);
```

After specifying the audio source and format and its output file, we can then call the two basic methods prepare and start to start recording the audio.

```
myAudioRecorder.prepare();
myAudioRecorder.start();
```

Apart from these methods , there are other methods listed in the MediaRecorder class that allows you more control over audio and video recording.

Sr.No	Method & description
1	**setAudioSource()** This method specifies the source of audio to be recorded
2	**setVideoSource()** This method specifies the source of video to be recorded
3	**setOutputFormat()** This method specifies the audio format in which audio to be stored
4	**setAudioEncoder()** This method specifies the audio encoder to be used
5	**setOutputFile()** This method configures the path to the file into which the recorded audio is to be stored
6	**stop()** This method stops the recording process.
7	**release()** This method should be called when the recorder instance is needed.

Example

This example provides demonstration of MediaRecorder class to capture audio and then MediaPlayer class to play that recorded audio.

To experiment with this example , you need to run this on an actual device.

Steps	Description
1	You will use Android studio IDE to create an Android application and name it as AudioCapture under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add AudioCapture code
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	Modify AndroidManifest.xml to add necessary permissions.
5	Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/MainActivity.java**

```
package com.example.sairamkrishna.myapplication;

import android.media.MediaPlayer;
import android.media.MediaRecorder;

import android.os.Environment;
import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
import android.view.View;

import android.widget.Button;
import android.widget.Toast;

import java.io.IOException;
import java.util.Random;

import static android.Manifest.permission.RECORD_AUDIO;
import static android.Manifest.permission.WRITE_EXTERNAL_STORAGE;

import android.support.v4.app.ActivityCompat;
import android.content.pm.PackageManager;
import android.support.v4.content.ContextCompat;

public class MainActivity extends AppCompatActivity {

  Button buttonStart, buttonStop, buttonPlayLastRecordAudio,
    buttonStopPlayingRecording ;
  String AudioSavePathInDevice = null;
  MediaRecorder mediaRecorder ;
  Random random ;
  String RandomAudioFileName = "ABCDEFGHIJKLMNOP";
  public static final int RequestPermissionCode = 1;
  MediaPlayer mediaPlayer ;

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

    buttonStart = (Button) findViewById(R.id.button);
    buttonStop = (Button) findViewById(R.id.button2);
    buttonPlayLastRecordAudio = (Button) findViewById(R.id.button3);
    buttonStopPlayingRecording = (Button)findViewById(R.id.button4);

    buttonStop.setEnabled(false);
    buttonPlayLastRecordAudio.setEnabled(false);
    buttonStopPlayingRecording.setEnabled(false);

    random = new Random();

    buttonStart.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {

        if(checkPermission()) {

          AudioSavePathInDevice =
```

```
        Environment.getExternalStorageDirectory().getAbsolutePath() + "/" +
          CreateRandomAudioFileName(5) + "AudioRecording.3gp";

      MediaRecorderReady();

      try {
        mediaRecorder.prepare();
        mediaRecorder.start();
      } catch (IllegalStateException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }

      buttonStart.setEnabled(false);
      buttonStop.setEnabled(true);

      Toast.makeText(MainActivity.this, "Recording started",
        Toast.LENGTH_LONG).show();
    } else {
      requestPermission();
    }

  }
});

buttonStop.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View view) {
    mediaRecorder.stop();
    buttonStop.setEnabled(false);
    buttonPlayLastRecordAudio.setEnabled(true);
    buttonStart.setEnabled(true);
    buttonStopPlayingRecording.setEnabled(false);

    Toast.makeText(MainActivity.this, "Recording Completed",
      Toast.LENGTH_LONG).show();
  }
});

buttonPlayLastRecordAudio.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View view) throws IllegalArgumentException,
    SecurityException, IllegalStateException {

    buttonStop.setEnabled(false);
    buttonStart.setEnabled(false);
    buttonStopPlayingRecording.setEnabled(true);
```

```
        mediaPlayer = new MediaPlayer();
        try {
          mediaPlayer.setDataSource(AudioSavePathInDevice);
          mediaPlayer.prepare();
        } catch (IOException e) {
          e.printStackTrace();
        }

        mediaPlayer.start();
        Toast.makeText(MainActivity.this, "Recording Playing",
          Toast.LENGTH_LONG).show();
      }
    });

    buttonStopPlayingRecording.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        buttonStop.setEnabled(false);
        buttonStart.setEnabled(true);
        buttonStopPlayingRecording.setEnabled(false);
        buttonPlayLastRecordAudio.setEnabled(true);

        if(mediaPlayer != null){
          mediaPlayer.stop();
          mediaPlayer.release();
          MediaRecorderReady();
        }
      }
    });

  }

  public void MediaRecorderReady(){
    mediaRecorder=new MediaRecorder();
    mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
    mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    mediaRecorder.setAudioEncoder(MediaRecorder.OutputFormat.AMR_NB);
    mediaRecorder.setOutputFile(AudioSavePathInDevice);
  }

  public String CreateRandomAudioFileName(int string){
    StringBuilder stringBuilder = new StringBuilder( string );
    int i = 0 ;
    while(i < string ) {
      stringBuilder.append(RandomAudioFileName.
        charAt(random.nextInt(RandomAudioFileName.length())));

      i++ ;
    }
```

```
    return stringBuilder.toString();
  }

  private void requestPermission() {
    ActivityCompat.requestPermissions(MainActivity.this, new
      String[]{WRITE_EXTERNAL_STORAGE, RECORD_AUDIO}, RequestPermissionCode);
  }

  @Override
  public void onRequestPermissionsResult(int requestCode,
    String permissions[], int[] grantResults) {
    switch (requestCode) {
      case RequestPermissionCode:
        if (grantResults.length> 0) {
        boolean StoragePermission = grantResults[0] ==
          PackageManager.PERMISSION_GRANTED;
        boolean RecordPermission = grantResults[1] ==
          PackageManager.PERMISSION_GRANTED;

        if (StoragePermission && RecordPermission) {
          Toast.makeText(MainActivity.this, "Permission Granted",
            Toast.LENGTH_LONG).show();
        } else {
          Toast.makeText(MainActivity.this,"Permission
            Denied",Toast.LENGTH_LONG).show();
        }
      }
      break;
    }
  }

  public boolean checkPermission() {
    int result = ContextCompat.checkSelfPermission(getApplicationContext(),
      WRITE_EXTERNAL_STORAGE);
    int result1 = ContextCompat.checkSelfPermission(getApplicationContext(),
      RECORD_AUDIO);
    return result == PackageManager.PERMISSION_GRANTED &&
      result1 == PackageManager.PERMISSION_GRANTED;
  }
}
```

Here is the content of **activity_main.xml**

In the below code **abc** indicates the logo of tutorialspoint

```
<?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:paddingBottom="@dimen/activity_vertical_margin"
```

```
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin">

<ImageView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/imageView"
  android:layout_alignParentTop="true"
  android:layout_centerHorizontal="true"
  android:src="@drawable/abc"/>

<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Record"
  android:id="@+id/button"
  android:layout_below="@+id/imageView"
  android:layout_alignParentLeft="true"
  android:layout_marginTop="37dp"
/>

<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="STOP"
  android:id="@+id/button2"
  android:layout_alignTop="@+id/button"
  android:layout_centerHorizontal="true"
/>

<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Play"
  android:id="@+id/button3"
  android:layout_alignTop="@+id/button2"
  android:layout_alignParentRight="true"
  android:layout_alignParentEnd="true"
/>

<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="STOP PLAYING RECORDING "
  android:id="@+id/button4"
  android:layout_below="@+id/button2"
  android:layout_centerHorizontal="true"
  android:layout_marginTop="10dp"
/>
```

```
</RelativeLayout>
```

Here is the content of **Strings.xml**

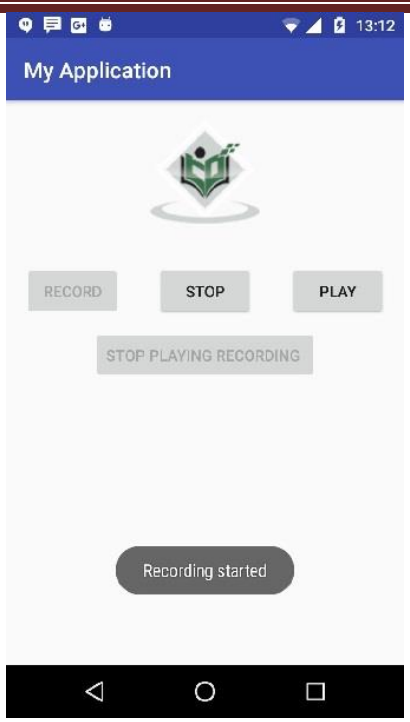
```
<resources>
  <string name="app_name">My Application</string>
</resources>
```

Here is the content of **AndroidManifest.xml**

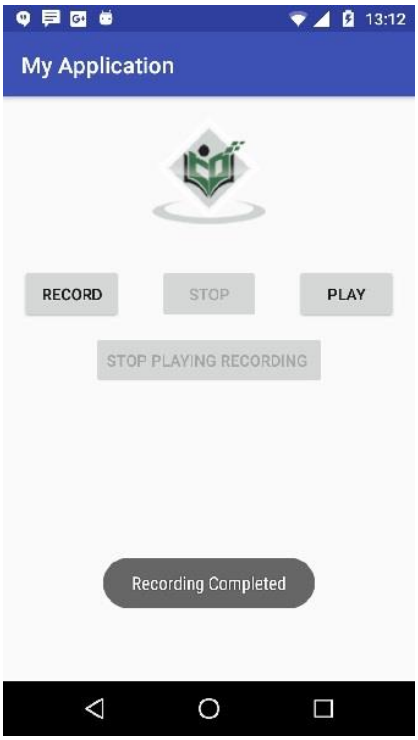
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.sairamkrishna.myapplication" >

  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.RECORD_AUDIO" />
            <uses-permission android:name="android.permission.STORAGE" />

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.sairamkrishna.myapplication.MainActivity"
      android:label="@string/app_name" >

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

    </activity>

  </application>
</manifest>
```

Let's try to run your application. I assume you have connected your actual Android Mobile device with your computer. To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Before starting your application, Android studio will display following images.

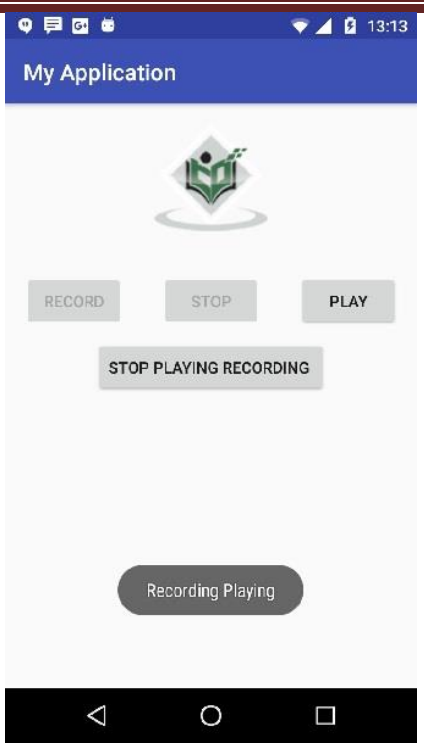
Now by default you will see stop and play button disable. Just press the Record button and your application will start recording the audio. It will display the following screen.



Now just press stop button and it will save the recorded audio to external sd card. When you click on stop button , the following screen would appear.



Now just press the play button and and recorded audio will just start playing on the device. The following message appears when you click on play button.



Camera In Android

In Android, Camera is a hardware device that allows capturing pictures and videos in your applications. Follow this tutorial to easily understand how to use a camera in your own Android App.

Android provides the facility to work on camera by 2 ways:

1. By Camera Intent
2. By Camera API

1 Using Camera By Using Camera Application
We can capture pictures without using the instance of Camera class. Here you will use an intent action type of MediaStore.ACTION_IMAGE_CAPTURE to launch an existing Camera application on your phone. In Android MediaStore is a type of DataBase which stores pictures and videos in android.

Intent cameraIntent = new
Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);

2 Using Camera By using Camera Api
This class is used for controlling device cameras. It can be used to take pictures when you are building a camera application.
Camera API works in following ways:

1.Camera Manager: This is used to get all the cameras available in the device like front camera back camera each having the camera id.

2.CameraDevice: You can get it from Camera Manager class by its id.

3.CaptureRequest: You can create a capture request from camera device to capture images.

4.CameraCaptureSession: To get capture request's from Camera Device create a CameraCaptureSession.

5.CameraCaptureSession.CaptureCallback: This is going to provide the Capture session results.

Camera Permission Declarations In Manifest
First, you should declare the Camera requirement in your Manifest file if Camera is compulsory for your application and you don't want your application to be installed on a device that does not support Camera.

Before you start development on your application you have to make sure that your Manifest has appropriate declarations in it that will allow you to use Camera feature in your Application.

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

Android camera app example by camera intent

activity_main.xml

Drag one imageview and one button from the pallete, now the xml file will look like this:

File: 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"
  tools:context=".MainActivity" >

  <Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:text="Take a Photo" >
  </Button>

  <ImageView
    android:id="@+id/imageView1"
```

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_above="@+id/button1"
    android:layout_alignParentTop="true"
    android:src="@drawable/ic_launcher" >
  </ImageView>
</RelativeLayout>
```

Activity class

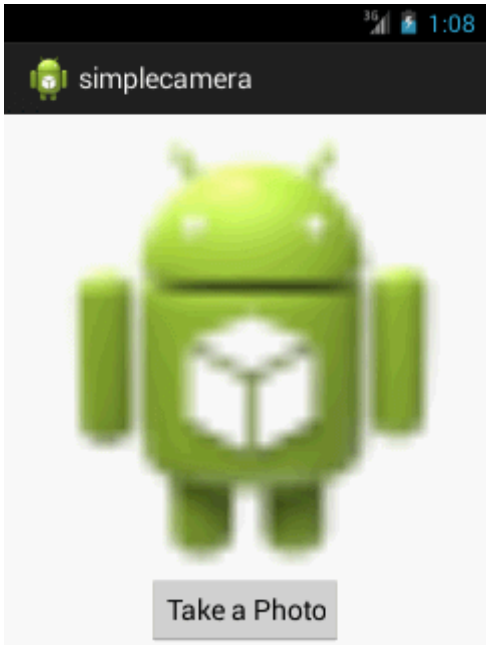
Let's write the code to capture image using camera and displaying it on the image view.

File: MainActivity.java

```
package com.example.simplecamera;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;

public class MainActivity extends Activity {
   private static final int CAMERA_REQUEST = 1888;
   ImageView imageView;
   public void onCreate(Bundle savedInstanceState) {

     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);

     imageView = (ImageView) this.findViewById(R.id.imageView1);
     Button photoButton = (Button) this.findViewById(R.id.button1);

     photoButton.setOnClickListener(new View.OnClickListener() {

     @Override
     public void onClick(View v) {
        Intent cameraIntent = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAP
TURE);
        startActivityForResult(cameraIntent, CAMERA_REQUEST);
     }
     });
    }
```

```
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
     if (requestCode == CAMERA_REQUEST) {
      Bitmap photo = (Bitmap) data.getExtras().get("data");
      imageView.setImageBitmap(photo);
     }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }

}
```

Output:



Android Bluetooth Tutorial

Bluetooth is a way to exchange data with other devices wirelessly. Android provides Bluetooth API to perform several tasks such as:

- scan bluetooth devices
- connect and transfer data from and to other devices
- manage multiple connections etc.

Android Bluetooth API

The android.bluetooth package provides a lot of interfaces classes to work with bluetooth such as:

- BluetoothAdapter
- BluetoothDevice
- BluetoothSocket
- BluetoothServerSocket
- BluetoothClass
- BluetoothProfile
- BluetoothProfile.ServiceListener
- BluetoothHeadset
- BluetoothA2dp
- BluetoothHealth
- BluetoothHealthCallback
- BluetoothHealthAppConfiguration

---

android-preferences-example

BluetoothAdapter class

By the help of BluetoothAdapter class, we can perform fundamental tasks such as initiate device discovery, query a list of paired (bonded) devices, create a BluetoothServerSocket instance to listen for connection requests etc.

Constants of BluetoothAdapter class

BluetoothAdapter class provides many constants. Some of them are as follows:

- String ACTION_REQUEST_ENABLE
- String ACTION_REQUEST_DISCOVERABLE
- String ACTION_DISCOVERY_STARTED
- String ACTION_DISCOVERY_FINISHED

Methods of BluetoothAdapter class

Commonly used methods of BluetoothAdapter class are as follows:

- **static synchronized BluetoothAdapter getDefaultAdapter()** returns the instance of BluetoothAdapter.
- **boolean enable()** enables the bluetooth adapter if it is disabled.
- **boolean isEnabled()** returns true if the bluetooth adapter is enabled.

- **boolean disable()** disables the bluetooth adapter if it is enabled.
- **String getName()** returns the name of the bluetooth adapter.
- **boolean setName(String name)** changes the bluetooth name.
- **int getState()** returns the current state of the local bluetooth adapter.
- **Set<BluetoothDevice> getBondedDevices()** returns a set of paired (bonded) BluetoothDevice objects.
- **boolean startDiscovery()** starts the discovery process.

Android Bluetooth Example: enable, disable and make discovrable bluetooth programmatically

You need to write few lines of code only, to enable or disable the bluetooth.

activity_main.xml

Drag one textview and three buttons from the pallete, now the activity_main.xml file will like this:

File: 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"
  tools:context=".MainActivity" >

  <TextView android:text=""
   android:id="@+id/out"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content">
  </TextView>
  <Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="30dp"
    android:layout_marginTop="49dp"
    android:text="TURN_ON" />

  <Button
    android:id="@+id/button2"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/button1"
    android:layout_below="@+id/button1"
    android:layout_marginTop="27dp"
    android:text="DISCOVERABLE" />

  <Button
    android:id="@+id/button3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/button2"
    android:layout_below="@+id/button2"
    android:layout_marginTop="28dp"
    android:text="TURN_OFF" />

</RelativeLayout>
```

Provide Permission

You need to provide following permissions in AndroidManifest.xml file.

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

The full code of AndroidManifest.xml file is given below.

File: AndroidManifest.xml

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

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="16" />

  <uses-permission android:name="android.permission.BLUETOOTH" />
  <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

  <application
    android:allowBackup="true"
```

```
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name="com.example.bluetooth.MainActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Activity class

Let's write the code to enable, disable and make bluetooth discoverable.

File: MainActivity.java

```
package com.example.bluetooth;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {
    private static final int REQUEST_ENABLE_BT = 0;
    private static final int REQUEST_DISCOVERABLE_BT = 0;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```
  final TextView out=(TextView)findViewById(R.id.out);
  final Button button1 = (Button) findViewById(R.id.button1);
  final Button button2 = (Button) findViewById(R.id.button2);
  final Button button3 = (Button) findViewById(R.id.button3);
  final BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
  if (mBluetoothAdapter == null) {
    out.append("device not supported");
  }
  button1.setOnClickListener(new View.OnClickListener() {
     public void onClick(View v) {
       if (!mBluetoothAdapter.isEnabled()) {
          Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
          startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
       }
     }
  });
  button2.setOnClickListener(new View.OnClickListener() {
   @Override
     public void onClick(View arg0) {
       if (!mBluetoothAdapter.isDiscovering()) {
            //out.append("MAKING YOUR DEVICE DISCOVERABLE");
             Toast.makeText(getApplicationContext(), "MAKING YOUR DEVICE DISCOVERABLE
",
         Toast.LENGTH_LONG);

          Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVER
ABLE);
          startActivityForResult(enableBtIntent, REQUEST_DISCOVERABLE_BT);

       }
     }
  });
  button3.setOnClickListener(new View.OnClickListener() {
     @Override
     public void onClick(View arg0) {
       mBluetoothAdapter.disable();
       //out.append("TURN_OFF BLUETOOTH");
       Toast.makeText(getApplicationContext(), "TURNING_OFF BLUETOOTH", Toast.LENGT
H_LONG);

       }
  });
}

  @Override
```

```
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }

}
```

Android Animation

Animation in android apps is the process of creating motion and shape change.

Android Animation Example XML

We create a resource directory under the res folder names **anim** to keep all the xml files containing the animation logic. Following is a sample xml file showing an android animation code logic.

sample_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<scale xmlns:android="https://schemas.android.com/apk/res/android"
 android:interpolator="@android:anim/accelerate_decelerate_interpolator"
 android:duration="300"
 android:fillAfter="true"
 android:fromXScale="0.0"
 android:fromYScale="0.0"
 android:toXScale="1.0"
 android:toYScale="1.0" />
```

- **android:interpolator** : It is the rate of change in animation. We can define our own interpolators using the time as the constraint. In the above xml code an inbuilt interpolator is assigned
- **android:duration** : Duration of the animation in which the animation should complete. It is 300ms here. This is generally the ideal duration to show the transition on the screen.

The start and end of the animation are set using:

android:fromTRANSFORMATION

android:toTRANSFORMATION

- **TRANSFORMATION** : is the transformation that we want to specify. In our case we start with an x and y scale of 0 and end with an x and y scale of 1
- **android:fillAfter** : property specifies whether the view should be visible or hidden at the end of the animation. We've set it visible in the above code. If it sets to false, the element changes to its previous state after the animation
- **android:startOffset** : It is the waiting time before an animation starts. This property is mainly used to perform multiple animations in a sequential manner
- **android:repeatMode** : This is useful when you want the animation to be repeat
- **android:repeatCount** : This defines number of repetitions on animation. If we set this value to infinite then animation will repeat infinite times

Loading Animation when UI widget is clicked

Our aim is to show an animation when any widget(lets say TextView) is clicked. For that we need to use the Animation Class. The xml file that contains the animation logic is loaded using **AnimationUtils** class by calling the loadAnimation() function. The below snippet shows this implementation.

```
Animation animation;

animation = AnimationUtils.loadAnimation(getApplicationContext(),

        R.anim.sample_animation);
```

To start the animation we need to call the startAnimation() function on the UI element as shown in the snippet below:

```
sampleTextView.startAnimation(animation);
```

Here we perform the animation on a textview component by passing the type of Animation as the parameter.

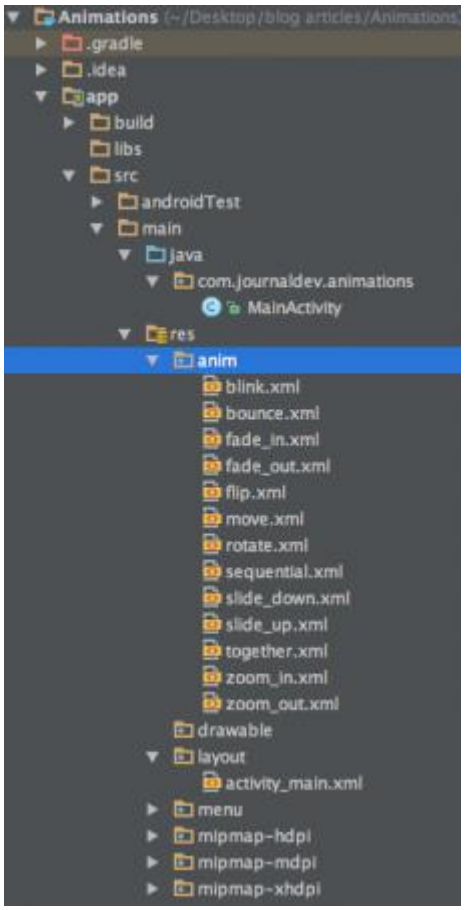
Setting the Animation Listeners

This is only needed if we wish to listen to events like start, end or repeat. For this the activity must implement **AnimationListener** and the following methods need to overridden.

- **onAnimationStart** : This will be triggered once the animation started
- **onAnimationEnd** : This will be triggered once the animation is over

- **onAnimationRepeat** : This will be triggered if the animation repeats

Android Animation Project Structure



As you can see, we've included the xml of all the major types of animations covered above.

Android Animation Examples XML Code

Here I am providing sample code for most of the common android animations.

Fade In Animation

fade_in.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
   android:fillAfter="true" >

   <alpha
```

```
    android:duration="1000"
    android:fromAlpha="0.0"
    android:interpolator="@android:anim/accelerate_interpolator"
    android:toAlpha="1.0" />

</set>
```

Here **alpha** references the opacity of an object. An object with lower alpha values is more transparent, while an object with higher alpha values is less transparent, more opaque. Fade in animation is nothing but increasing alpha value from 0 to 1.

Fade Out Animation

fade_out.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
  android:fillAfter="true" >

  <alpha
    android:duration="1000"
    android:fromAlpha="1.0"
    android:interpolator="@android:anim/accelerate_interpolator"
    android:toAlpha="0.0" />

</set>
```

Fade out android animation is exactly opposite to fade in, where we need to decrease the alpha value from 1 to 0.

Cross Fading Animation

Cross fading is performing fade in animation on one TextView while other TextView is fading out. This can be done by using fade_in.xml and fade_out.xml on the two TextViews. The code will be discussed in the MainActivity.java

Blink Animation

blink.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android">
   <alpha android:fromAlpha="0.0"
      android:toAlpha="1.0"
      android:interpolator="@android:anim/accelerate_interpolator"
      android:duration="600"
      android:repeatMode="reverse"
      android:repeatCount="infinite"/>
</set>
```

Here fade in and fade out are performed infinitely in reverse mode each time.

Zoom In Animation

zoom_in.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
   android:fillAfter="true" >

   <scale
      xmlns:android="https://schemas.android.com/apk/res/android"
      android:duration="1000"
      android:fromXScale="1"
```

```
    android:fromYScale="1"
    android:pivotX="50%"
    android:pivotY="50%"
    android:toXScale="3"
    android:toYScale="3" >
  </scale>

</set>
```

We use pivotX="50%" and pivotY="50%" to perform zoom from the center of the element.

Zoom Out Animation

zoom_out.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
  android:fillAfter="true" >

  <scale
    xmlns:android="https://schemas.android.com/apk/res/android"
    android:duration="1000"
    android:fromXScale="1.0"
    android:fromYScale="1.0"
    android:pivotX="50%"
    android:pivotY="50%"
    android:toXScale="0.5"
    android:toYScale="0.5" >
  </scale>
```

```
</set>
```

Notice that **android:from** and **android:to** are opposite in zoom_in.xml and zoom_out.xml.

Rotate Animation

rotate.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android">
   <rotate android:fromDegrees="0"
      android:toDegrees="360"
      android:pivotX="50%"
      android:pivotY="50%"
      android:duration="600"
      android:repeatMode="restart"
      android:repeatCount="infinite"
      android:interpolator="@android:anim/cycle_interpolator"/>

</set>
```

A **from/toDegrees** tag is used here to specify the degrees and a cyclic interpolator is used.

Move Animation

move.xml

```
<set
   xmlns:android="https://schemas.android.com/apk/res/android"
   android:interpolator="@android:anim/linear_interpolator"
   android:fillAfter="true">
```

```
  <translate
      android:fromXDelta="0%p"
      android:toXDelta="75%p"
      android:duration="800" />
</set>
```

Slide Up Animation

slide_up.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
   android:fillAfter="true" >

   <scale
      android:duration="500"
      android:fromXScale="1.0"
      android:fromYScale="1.0"
      android:interpolator="@android:anim/linear_interpolator"
      android:toXScale="1.0"
      android:toYScale="0.0" />

</set>
```

It's achieved by setting **android:fromYScale="1.0"** and **android:toYScale="0.0"** inside the **scale** tag.

Slide Down Animation

slide_down.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
   android:fillAfter="true">

   <scale
      android:duration="500"
      android:fromXScale="1.0"
      android:fromYScale="0.0"
      android:toXScale="1.0"
      android:toYScale="1.0" />

</set>
```

This is just the opposite of slide_up.xml.

Bounce Animation

bounce.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
   android:fillAfter="true"
   android:interpolator="@android:anim/bounce_interpolator">

   <scale
      android:duration="500"
      android:fromXScale="1.0"
      android:fromYScale="0.0"
      android:toXScale="1.0"
      android:toYScale="1.0" />
```

```
</set>
```

Here bounce interpolator is used to complete the animation in bouncing fashion.

Sequential Animation

sequential.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
   android:fillAfter="true"
   android:interpolator="@android:anim/linear_interpolator" >


   <!-- Move -->
   <translate
      android:duration="800"
      android:fillAfter="true"
      android:fromXDelta="0%p"
      android:startOffset="300"
      android:toXDelta="75%p" />
   <translate
      android:duration="800"
      android:fillAfter="true"
      android:fromYDelta="0%p"
      android:startOffset="1100"
      android:toYDelta="70%p" />
   <translate
      android:duration="800"
      android:fillAfter="true"
```

```
    android:fromXDelta="0%p"
    android:startOffset="1900"
    android:toXDelta="-75%p" />
  <translate
    android:duration="800"
    android:fillAfter="true"
    android:fromYDelta="0%p"
    android:startOffset="2700"
    android:toYDelta="-70%p" />

  <!-- Rotate 360 degrees -->
  <rotate
    android:duration="1000"
    android:fromDegrees="0"
    android:interpolator="@android:anim/cycle_interpolator"
    android:pivotX="50%"
    android:pivotY="50%"
    android:startOffset="3800"
    android:repeatCount="infinite"
    android:repeatMode="restart"
    android:toDegrees="360" />

</set>
```

Here a different **android:startOffset** is used from the transitions to keep them sequential.

Together Animation

together.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
   android:fillAfter="true"
   android:interpolator="@android:anim/linear_interpolator" >



   <!-- Move -->
   <scale
      xmlns:android="https://schemas.android.com/apk/res/android"
      android:duration="4000"
      android:fromXScale="1"
      android:fromYScale="1"
      android:pivotX="50%"
      android:pivotY="50%"
      android:toXScale="4"
      android:toYScale="4" >
   </scale>

   <!-- Rotate 180 degrees -->
   <rotate
      android:duration="500"
      android:fromDegrees="0"
      android:pivotX="50%"
      android:pivotY="50%"
      android:repeatCount="infinite"
      android:repeatMode="restart"
      android:toDegrees="360" />
```

```
</set>
```

Android Animation is used to give the UI a rich look and feel. Animations in android apps can be performed through XML or android code. In this android animation tutorial we'll go with XML codes for adding animations into our application.

Android Animation

Animation in android apps is the process of creating motion and shape change. The basic ways of animation that we'll look upon in this tutorial are:

1. Fade In Animation
2. Fade Out Animation
3. Cross Fading Animation
4. Blink Animation
5. Zoom In Animation
6. Zoom Out Animation
7. Rotate Animation
8. Move Animation
9. Slide Up Animation
10. Slide Down Animation
11. Bounce Animation
12. Sequential Animation
13. Together Animation

Android Animation Example XML

We create a resource directory under the res folder names **anim** to keep all the xml files containing the animation logic. Following is a sample xml file showing an android animation code logic.

sample_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<scale xmlns:android="https://schemas.android.com/apk/res/android"
 android:interpolator="@android:anim/accelerate_decelerate_interpolator"
```

```
android:duration="300"

android:fillAfter="true"

android:fromXScale="0.0"

android:fromYScale="0.0"

android:toXScale="1.0"

android:toYScale="1.0" />
```

- **android:interpolator** : It is the rate of change in animation. We can define our own interpolators using the time as the constraint. In the above xml code an inbuilt interpolator is assigned
- **android:duration** : Duration of the animation in which the animation should complete. It is 300ms here. This is generally the ideal duration to show the transition on the screen.

The start and end of the animation are set using:

android:fromTRANSFORMATION

android:toTRANSFORMATION

- **TRANSFORMATION** : is the transformation that we want to specify. In our case we start with an x and y scale of 0 and end with an x and y scale of 1
- **android:fillAfter** : property specifies whether the view should be visible or hidden at the end of the animation. We've set it visible in the above code. If it sets to false, the element changes to its previous state after the animation
- **android:startOffset** : It is the waiting time before an animation starts. This property is mainly used to perform multiple animations in a sequential manner
- **android:repeatMode** : This is useful when you want the animation to be repeat
- **android:repeatCount** : This defines number of repetitions on animation. If we set this value to infinite then animation will repeat infinite times

Loading Animation when UI widget is clicked

Our aim is to show an animation when any widget(lets say TextView) is clicked. For that we need to use the Animation Class. The xml file that contains the animation logic is loaded using **AnimationUtils** class by calling the loadAnimation() function. The below snippet shows this implementation.

```
Animation animation;

animation = AnimationUtils.loadAnimation(getApplicationContext(),

        R.anim.sample_animation);
```

To start the animation we need to call the startAnimation() function on the UI element as shown in the snippet below:

```
sampleTextView.startAnimation(animation);
```

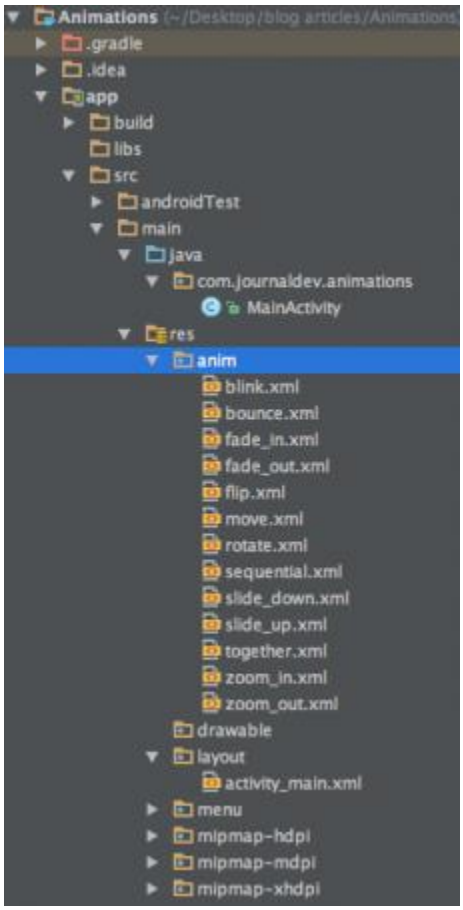
Here we perform the animation on a textview component by passing the type of Animation as the parameter.

Setting the Animation Listeners

This is only needed if we wish to listen to events like start, end or repeat. For this the activity must implement **AnimationListener** and the following methods need to overridden.

- **onAnimationStart** : This will be triggered once the animation started
- **onAnimationEnd** : This will be triggered once the animation is over
- **onAnimationRepeat** : This will be triggered if the animation repeats

Android Animation Project Structure



As you can see, we've included the xml of all the major types of animations covered above.

Android Animation Examples XML Code

Here I am providing sample code for most of the common android animations.

Fade In Animation

fade_in.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
   android:fillAfter="true" >

   <alpha
      android:duration="1000"
      android:fromAlpha="0.0"
      android:interpolator="@android:anim/accelerate_interpolator"
      android:toAlpha="1.0" />

</set>
```

Here **alpha** references the opacity of an object. An object with lower alpha values is more transparent, while an object with higher alpha values is less transparent, more opaque. Fade in animation is nothing but increasing alpha value from 0 to 1.

Fade Out Animation

fade_out.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
   android:fillAfter="true" >

   <alpha
      android:duration="1000"
```

```
    android:fromAlpha="1.0"
    android:interpolator="@android:anim/accelerate_interpolator"
    android:toAlpha="0.0" />

</set>
```

Fade out android animation is exactly opposite to fade in, where we need to decrease the alpha value from 1 to 0.

Cross Fading Animation

Cross fading is performing fade in animation on one TextView while other TextView is fading out. This can be done by using fade_in.xml and fade_out.xml on the two TextViews. The code will be discussed in the MainActivity.java

Blink Animation

blink.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android">
  <alpha android:fromAlpha="0.0"
    android:toAlpha="1.0"
    android:interpolator="@android:anim/accelerate_interpolator"
    android:duration="600"
    android:repeatMode="reverse"
    android:repeatCount="infinite"/>
</set>
```

Here fade in and fade out are performed infinitely in reverse mode each time.

Zoom In Animation

zoom_in.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
  android:fillAfter="true" >

  <scale
    xmlns:android="https://schemas.android.com/apk/res/android"
    android:duration="1000"
    android:fromXScale="1"
    android:fromYScale="1"
    android:pivotX="50%"
    android:pivotY="50%"
    android:toXScale="3"
    android:toYScale="3" >
  </scale>

</set>
```

We use pivotX="50%" and pivotY="50%" to perform zoom from the center of the element.

Zoom Out Animation

zoom_out.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
  android:fillAfter="true" >

  <scale
    xmlns:android="https://schemas.android.com/apk/res/android"
    android:duration="1000"
    android:fromXScale="1.0"
```

```
    android:fromYScale="1.0"
    android:pivotX="50%"
    android:pivotY="50%"
    android:toXScale="0.5"
    android:toYScale="0.5" >
  </scale>

</set>
```

Notice that **android:from** and **android:to** are opposite in zoom_in.xml and zoom_out.xml.

Rotate Animation

rotate.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android">
  <rotate android:fromDegrees="0"
    android:toDegrees="360"
    android:pivotX="50%"
    android:pivotY="50%"
    android:duration="600"
    android:repeatMode="restart"
    android:repeatCount="infinite"
    android:interpolator="@android:anim/cycle_interpolator"/>

</set>
```

A **from/toDegrees** tag is used here to specify the degrees and a cyclic interpolator is used.

Move Animation

move.xml

```
<set
   xmlns:android="https://schemas.android.com/apk/res/android"
   android:interpolator="@android:anim/linear_interpolator"
   android:fillAfter="true">

  <translate
      android:fromXDelta="0%p"
      android:toXDelta="75%p"
      android:duration="800" />
</set>
```

Slide Up Animation

slide_up.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
   android:fillAfter="true" >

   <scale
      android:duration="500"
      android:fromXScale="1.0"
      android:fromYScale="1.0"
      android:interpolator="@android:anim/linear_interpolator"
      android:toXScale="1.0"
      android:toYScale="0.0" />
```

```
</set>
```

It's achieved by setting **android:fromYScale="1.0"** and **android:toYScale="0.0"** inside the **scale** tag.

Slide Down Animation

slide_down.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
   android:fillAfter="true">

   <scale
      android:duration="500"
      android:fromXScale="1.0"
      android:fromYScale="0.0"
      android:toXScale="1.0"
      android:toYScale="1.0" />

</set>
```

This is just the opposite of slide_up.xml.

Bounce Animation

bounce.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
   android:fillAfter="true"
   android:interpolator="@android:anim/bounce_interpolator">
```

```
<scale
    android:duration="500"
    android:fromXScale="1.0"
    android:fromYScale="0.0"
    android:toXScale="1.0"
    android:toYScale="1.0" />

</set>
```

Here bounce interpolator is used to complete the animation in bouncing fashion.

Sequential Animation

sequential.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
  android:fillAfter="true"
  android:interpolator="@android:anim/linear_interpolator" >

  <!-- Move -->
  <translate
    android:duration="800"
    android:fillAfter="true"
    android:fromXDelta="0%p"
    android:startOffset="300"
    android:toXDelta="75%p" />
  <translate
    android:duration="800"
```

```
    android:fillAfter="true"
    android:fromYDelta="0%p"
    android:startOffset="1100"
    android:toYDelta="70%p" />
  <translate
    android:duration="800"
    android:fillAfter="true"
    android:fromXDelta="0%p"
    android:startOffset="1900"
    android:toXDelta="-75%p" />
  <translate
    android:duration="800"
    android:fillAfter="true"
    android:fromYDelta="0%p"
    android:startOffset="2700"
    android:toYDelta="-70%p" />

  <!-- Rotate 360 degrees -->
  <rotate
    android:duration="1000"
    android:fromDegrees="0"
    android:interpolator="@android:anim/cycle_interpolator"
    android:pivotX="50%"
    android:pivotY="50%"
    android:startOffset="3800"
    android:repeatCount="infinite"
    android:repeatMode="restart"
```

```
    android:toDegrees="360" />


</set>
```

Here a different **android:startOffset** is used from the transitions to keep them sequential.

Together Animation

together.xml

```
<set xmlns:android="https://schemas.android.com/apk/res/android"
  android:fillAfter="true"
  android:interpolator="@android:anim/linear_interpolator" >



  <!-- Move -->
  <scale
    xmlns:android="https://schemas.android.com/apk/res/android"
    android:duration="4000"
    android:fromXScale="1"
    android:fromYScale="1"
    android:pivotX="50%"
    android:pivotY="50%"
    android:toXScale="4"
    android:toYScale="4" >
  </scale>

  <!-- Rotate 180 degrees -->
  <rotate
```

```
    android:duration="500"
    android:fromDegrees="0"
    android:pivotX="50%"
    android:pivotY="50%"
    android:repeatCount="infinite"
    android:repeatMode="restart"
    android:toDegrees="360" />

</set>
```

Here android:startOffset is removed to let them happen simultaneously.

Code

The activity_main.xml layout consists of a ScrollView and RelativeLayout (we'll discuss this in a later tutorial) in which every animation type is invoked on the text using their respective buttons. The xml file is shown below :

activity_main.xml

```
<ScrollView xmlns:android="https://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >

  <RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/btnFadeIn"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:text="Fade In" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Fade In"
        android:id="@+id/txt_fade_in"
      android:layout_alignBottom="@+id/btnFadeIn"
      android:layout_alignLeft="@+id/txt_fade_out"
      android:layout_alignStart="@+id/txt_fade_out" />


    <Button
        android:id="@+id/btnFadeOut"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:layout_below="@id/btnFadeIn"
        android:text="Fade Out" />


    <Button
        android:id="@+id/btnCrossFade"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:layout_below="@id/btnFadeOut"
        android:text="Cross Fade" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Cross Fade In"
        android:id="@+id/txt_out"
        android:visibility="gone"
        android:layout_gravity="center_horizontal"
        android:layout_alignTop="@+id/txt_in"
        android:layout_alignLeft="@+id/txt_in"
        android:layout_alignStart="@+id/txt_in" />

    <Button
        android:id="@+id/btnBlink"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:layout_below="@id/btnCrossFade"
        android:text="Blink" />

    <Button
```

```
        android:id="@+id/btnZoomIn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:layout_below="@id/btnBlink"
        android:text="Zoom In" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Blink"
        android:id="@+id/txt_blink"
        android:layout_gravity="center_horizontal"
      android:layout_alignBottom="@+id/btnBlink"
      android:layout_alignLeft="@+id/txt_zoom_in"
      android:layout_alignStart="@+id/txt_zoom_in" />

    <Button
        android:id="@+id/btnZoomOut"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:layout_below="@id/btnZoomIn"
        android:text="Zoom Out" />

    <Button
```

```
        android:id="@+id/btnRotate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:layout_below="@id/btnZoomOut"
        android:text="Rotate" />

    <Button
        android:id="@+id/btnMove"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:layout_below="@id/btnRotate"
        android:text="Move" />

    <Button
        android:id="@+id/btnSlideUp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:layout_below="@id/btnMove"
        android:text="Slide Up" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
```

```
        android:text="Fade Out"
        android:id="@+id/txt_fade_out"
        android:layout_gravity="center_horizontal"
      android:layout_alignBottom="@+id/btnFadeOut"
      android:layout_alignLeft="@+id/txt_in"
      android:layout_alignStart="@+id/txt_in" />

    <Button
        android:id="@+id/btnSlideDown"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:layout_below="@id/btnSlideUp"
        android:text="Slide Down" />

    <Button
        android:id="@+id/btnBounce"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:layout_below="@id/btnSlideDown"
        android:text="Bounce" />

    <Button
        android:id="@+id/btnSequential"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_margin="5dp"
        android:layout_below="@id/btnBounce"
        android:text="Sequential Animation" />

    <Button
        android:id="@+id/btnTogether"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/btnSequential"
        android:layout_margin="5dp"
        android:text="Together Animation" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Cross Fade Out"
        android:id="@+id/txt_in"
        android:layout_gravity="center_horizontal"
      android:layout_alignBottom="@+id/btnCrossFade"
      android:layout_alignLeft="@+id/txt_blink"
      android:layout_alignStart="@+id/txt_blink" />

    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textAppearance="?android:attr/textAppearanceMedium"
```

```
    android:text="Zoom In"
    android:id="@+id/txt_zoom_in"
    android:layout_alignBottom="@+id/btnZoomIn"
    android:layout_alignLeft="@+id/txt_zoom_out"
    android:layout_alignStart="@+id/txt_zoom_out" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Zoom Out"
    android:id="@+id/txt_zoom_out"
    android:layout_alignBottom="@+id/btnZoomOut"
    android:layout_toRightOf="@+id/btnSequential"
    android:layout_toEndOf="@+id/btnSequential" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Rotate"
    android:id="@+id/txt_rotate"
    android:layout_above="@+id/btnMove"
    android:layout_toRightOf="@+id/btnSequential"
    android:layout_toEndOf="@+id/btnSequential" />

  <TextView
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Move"
    android:id="@+id/txt_move"
    android:layout_alignBottom="@+id/btnMove"
    android:layout_alignLeft="@+id/txt_slide_up"
    android:layout_alignStart="@+id/txt_slide_up" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Slide Up"
    android:id="@+id/txt_slide_up"
    android:layout_alignBottom="@+id/btnSlideUp"
    android:layout_toRightOf="@+id/btnSequential"
    android:layout_toEndOf="@+id/btnSequential" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Slide Down"
    android:id="@+id/txt_slide_down"
    android:layout_alignBottom="@+id/btnSlideDown"
    android:layout_alignLeft="@+id/txt_slide_up"
```

```
    android:layout_alignStart="@+id/txt_slide_up" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Bounce"
    android:id="@+id/txt_bounce"
    android:layout_alignBottom="@+id/btnBounce"
    android:layout_alignLeft="@+id/txt_slide_down"
    android:layout_alignStart="@+id/txt_slide_down" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Sequential"
    android:id="@+id/txt_seq"
    android:layout_alignBottom="@+id/btnSequential"
    android:layout_alignLeft="@+id/txt_bounce"
    android:layout_alignStart="@+id/txt_bounce" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Together"
```

```
        android:id="@+id/txt_tog"
        android:layout_alignBottom="@+id/btnTogether"
        android:layout_toRightOf="@+id/btnSequential"
        android:layout_toEndOf="@+id/btnSequential" />

    </RelativeLayout>

</ScrollView>
```

To sum up, a RelativeLayout, as the name suggests the arrangement of UI Components is relative to each other.
The MainActivity.java file contains the onClick Listeners for every button related to its animation type. It's source code is given below.

```
package com.journaldev.animations;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {

    Button btnFadeIn, btnFadeOut, btnCrossFade, btnBlink, btnZoomIn,
        btnZoomOut, btnRotate, btnMove, btnSlideUp, btnSlideDown,
        btnBounce, btnSequential, btnTogether;
```

```
Animation
animFadeIn,animFadeOut,animBlink,animZoomIn,animZoomOut,animRotate

,animMove,animSlideUp,animSlideDown,animBounce,animSequential,animTogether,ani
mCrossFadeIn,animCrossFadeOut;

TextView
txtFadeIn,txtFadeOut,txtBlink,txtZoomIn,txtZoomOut,txtRotate,txtMove,txtSlideUp,

        txtSlideDown,txtBounce,txtSeq,txtTog,txtIn,txtOut;


@Override

protected void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);

  setContentView(R.layout.activity_main);


  btnFadeIn = (Button) findViewById(R.id.btnFadeIn);

  btnFadeOut = (Button) findViewById(R.id.btnFadeOut);

  btnCrossFade = (Button) findViewById(R.id.btnCrossFade);

  btnBlink = (Button) findViewById(R.id.btnBlink);

  btnZoomIn = (Button) findViewById(R.id.btnZoomIn);

  btnZoomOut = (Button) findViewById(R.id.btnZoomOut);

  btnRotate = (Button) findViewById(R.id.btnRotate);

  btnMove = (Button) findViewById(R.id.btnMove);

  btnSlideUp = (Button) findViewById(R.id.btnSlideUp);

  btnSlideDown = (Button) findViewById(R.id.btnSlideDown);

  btnBounce = (Button) findViewById(R.id.btnBounce);

  btnSequential = (Button) findViewById(R.id.btnSequential);

  btnTogether = (Button) findViewById(R.id.btnTogether);

  txtFadeIn=(TextView)findViewById(R.id.txt_fade_in);

  txtFadeOut=(TextView)findViewById(R.id.txt_fade_out);
```

```
txtBlink=(TextView)findViewById(R.id.txt_blink);
txtZoomIn=(TextView)findViewById(R.id.txt_zoom_in);
txtZoomOut=(TextView)findViewById(R.id.txt_zoom_out);
txtRotate=(TextView)findViewById(R.id.txt_rotate);
txtMove=(TextView)findViewById(R.id.txt_move);
txtSlideUp=(TextView)findViewById(R.id.txt_slide_up);
txtSlideDown=(TextView)findViewById(R.id.txt_slide_down);
txtBounce=(TextView)findViewById(R.id.txt_bounce);
txtSeq=(TextView)findViewById(R.id.txt_seq);
txtTog=(TextView)findViewById(R.id.txt_tog);
txtIn=(TextView)findViewById(R.id.txt_in);
txtOut=(TextView)findViewById(R.id.txt_out);
animFadeIn = AnimationUtils.loadAnimation(getApplicationContext(),
      R.anim.fade_in);

animFadeIn = AnimationUtils.loadAnimation(getApplicationContext(),
      R.anim.fade_in);
// fade in
btnFadeIn.setOnClickListener(new View.OnClickListener() {

   @Override
   public void onClick(View v) {
      txtFadeIn.setVisibility(View.VISIBLE);
      txtFadeIn.startAnimation(animFadeIn);
   }
});
```

```
animFadeOut = AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.fade_out);

// fade out
btnFadeOut.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        txtFadeOut.setVisibility(View.VISIBLE);
        txtFadeOut.startAnimation(animFadeOut);
    }
});
animCrossFadeIn = AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.fade_in);
animCrossFadeOut = AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.fade_out);
// cross fade
btnCrossFade.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        txtOut.setVisibility(View.VISIBLE);
        // start fade in animation
        txtOut.startAnimation(animCrossFadeIn);

        // start fade out animation
        txtIn.startAnimation(animCrossFadeOut);
    }
});
```

```
animBlink = AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.blink);
// blink
btnBlink.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        txtBlink.setVisibility(View.VISIBLE);
        txtBlink.startAnimation(animBlink);
    }
});

animZoomIn = AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.zoom_in);
// Zoom In
btnZoomIn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        txtZoomIn.setVisibility(View.VISIBLE);
        txtZoomIn.startAnimation(animZoomIn);
    }
});
animZoomOut = AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.zoom_out);
// Zoom Out
btnZoomOut.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
```

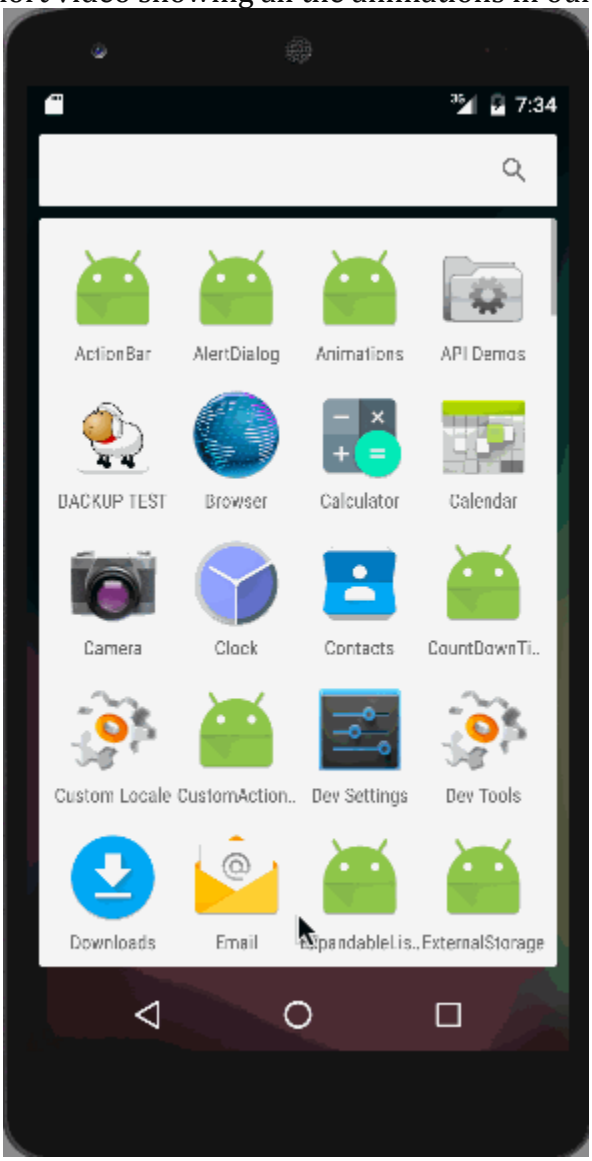
```
        txtZoomOut.setVisibility(View.VISIBLE);
        txtZoomOut.startAnimation(animZoomOut);
    }
});
animRotate = AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.rotate);

// Rotate
btnRotate.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        txtRotate.startAnimation(animRotate);
    }
});
animMove = AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.move);
// Move
btnMove.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        txtMove.startAnimation(animMove);
    }
});
animSlideUp = AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.slide_up);
// Slide Up
btnSlideUp.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
    public void onClick(View v) {
        txtSlideUp.startAnimation(animSlideUp);
    }
});
animSlideDown = AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.slide_down);
// Slide Down
btnSlideDown.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        txtSlideDown.startAnimation(animSlideDown);
    }
});
animBounce = AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.bounce);
// Slide Down
btnBounce.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        txtBounce.startAnimation(animBounce);
    }
});
animSequential = AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.sequential);
// Sequential
btnSequential.setOnClickListener(new View.OnClickListener() {
```

```
        @Override
        public void onClick(View v) {


                txtSeq.startAnimation(animSequential);
        }
      });
      animTogether = AnimationUtils.loadAnimation(getApplicationContext(),
          R.anim.together);


      // Together
      btnTogether.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
          txtTog.startAnimation(animTogether);
        }
      });


  }
}
```

As discussed before each textView animation is started by invoking the respective animation object in which the animation logic is loaded by **AnimationUtils.loadAnimation()** method. The crossFade animation consists of two TextViews in which one fades out and the other fades in.

Below is a short video showing all the animations in our application.



The together animation is seen in the image above. Note that these animation when run on an emulator wont be smooth, so it's recommended to run the application on a normal device.

This brings an end to android animation example tutorial.

Android SQLite Database

Android SQLite is the mostly preferred way to store data for android applications. For many applications, SQLite is the apps backbone whether it's used directly or via some third-party wrapper. Below is the final app we will create today using Android SQLite database.

Android SQLite

Android SQLite is a very lightweight database which comes with Android OS. Android SQLite combines a clean SQL interface with a very small memory footprint and decent speed. For Android, SQLite is "baked into" the Android runtime, so every Android application can create its own SQLite databases.

Android SQLite native API is not JDBC, as JDBC might be too much overhead for a memory-limited smartphone. Once a database is created successfully its located in **data/data//databases/** accessible from Android Device Monitor.

SQLite is a typical **relational database**, containing tables (which consists of rows and columns), indexes etc. We can create our own tables to hold the data accordingly. This structure is referred to as a **schema**.

Android SQLite SQLiteOpenHelper

Android has features available to handle changing database schemas, which mostly depend on using the SQLiteOpenHelper class.

SQLiteOpenHelper is designed to get rid of two very common problems.

1. When the application runs the first time – At this point, we do not yet have a database. So we will have to create the tables, indexes, starter data, and so on.
2. When the application is upgraded to a newer schema – Our database will still be on the old schema from the older edition of the app. We will have option to alter the database schema to match the needs of the rest of the app.

SQLiteOpenHelper wraps up these logic to create and upgrade a database as per our specifications. For that we'll need to create a custom subclass of SQLiteOpenHelper implementing at least the following three methods.

1. **Constructor** : This takes the Context (e.g., an Activity), the name of the database, an optional cursor factory (we'll discuss this later), and an integer representing the version of the database schema you are using (typically starting from 1 and increment later).
2.

```
public DatabaseHelper(Context context) {

    super(context, DB_NAME, null, DB_VERSION);

  }
```

3. **onCreate(SQLiteDatabase db)** : It's called when there is no database and the app needs one. It passes us a SQLiteDatabase object, pointing to a newly-created database, that we can populate with tables and initial data.

4. **onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)** : It's called when the schema version we need does not match the schema version of the database, It passes us a **SQLiteDatabase** object and the old and new version numbers. Hence we can figure out the best way to convert the database from the old schema to the new one.

We define a DBManager class to perform all database CRUD(Create, Read, Update and Delete) operations.

Opening and Closing Android SQLite Database Connection

Before performing any database operations like insert, update, delete records in a table, first open the database connection by calling **getWritableDatabase()** method as shown below:

```
public DBManager open() throws SQLException {
    dbHelper = new DatabaseHelper(context);
    database = dbHelper.getWritableDatabase();
    return this;
  }
```

The **dbHelper** is an instance of the subclass of SQLiteOpenHelper.

To close a database connection the following method is invoked.

```
public void close() {
    dbHelper.close();
  }
```

Inserting new Record into Android SQLite database table

The following code snippet shows how to insert a new record in the android SQLite database.

```
public void insert(String name, String desc) {
    ContentValues contentValue = new ContentValues();
    contentValue.put(DatabaseHelper.SUBJECT, name);
    contentValue.put(DatabaseHelper.DESC, desc);
```

```
    database.insert(DatabaseHelper.TABLE_NAME, null, contentValue);
  }
```

Content Values creates an empty set of values using the given initial size. We'll discuss the other instance values when we jump into the coding part.

Updating Record in Android SQLite database table

The following snippet shows how to update a single record.

```
public int update(long _id, String name, String desc) {
    ContentValues contentValues = new ContentValues();
    contentValues.put(DatabaseHelper.SUBJECT, name);
    contentValues.put(DatabaseHelper.DESC, desc);
    int i = database.update(DatabaseHelper.TABLE_NAME, contentValues,
DatabaseHelper._ID + " = " + _id, null);
    return i;
  }
```

Android SQLite – Deleting a Record

We just need to pass the id of the record to be deleted as shown below.

```
public void delete(long _id) {
    database.delete(DatabaseHelper.TABLE_NAME, DatabaseHelper._ID + "=" + _id, null);
  }
```

Android SQLite Cursor

A Cursor represents the entire result set of the query. Once the query is fetched a call to **cursor.moveToFirst()** is made. Calling moveToFirst() does two things:

- It allows us to test whether the query returned an empty set (by testing the return value)
- It moves the cursor to the first result (when the set is not empty)

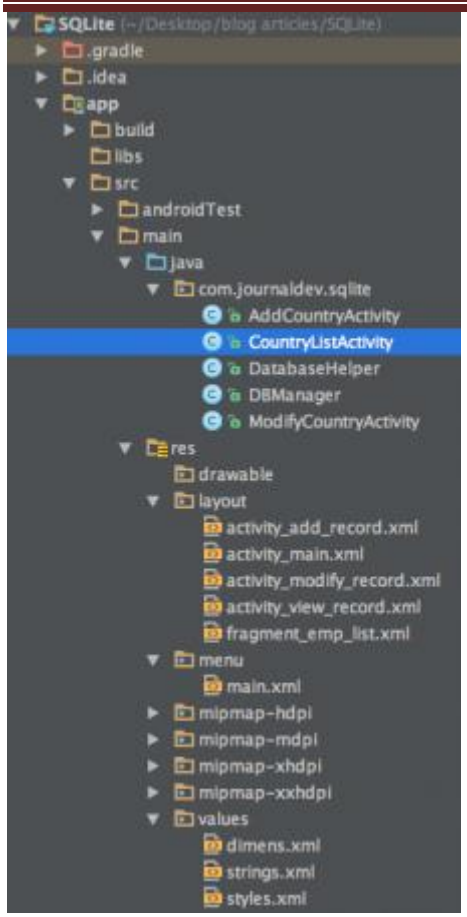
The following code is used to fetch all records:

```
public Cursor fetch() {
    String[] columns = new String[] { DatabaseHelper._ID, DatabaseHelper.SUBJECT,
DatabaseHelper.DESC };
    Cursor cursor = database.query(DatabaseHelper.TABLE_NAME, columns, null, null,
null, null, null);
    if (cursor != null) {
       cursor.moveToFirst();
    }
    return cursor;
  }
```

Another way to use a Cursor is to wrap it in a CursorAdapter. Just as ArrayAdapter adapts arrays, CursorAdapter adapts Cursor objects, making their data available to an AdapterView like a ListView.

Let's jump to our project that uses SQLite to store some meaningful data.

Android SQLite Example Project Structure



In this application we wish to create records that store Country names and their respective currencies in the form of a **ListView**. We cover all the features discusses above.

Android SQLite Project Code

The application consists of 5 classes. We begin with defining with **DatabaseHelper**, which is a subclass of SQLiteOpenHelper as follows:

DatabaseHelper.java

```
package com.journaldev.sqlite;

import android.content.Context;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

public class DatabaseHelper extends SQLiteOpenHelper {
```

```
// Table Name
public static final String TABLE_NAME = "COUNTRIES";

// Table columns
public static final String _ID = "_id";
public static final String SUBJECT = "subject";
public static final String DESC = "description";

// Database Information
static final String DB_NAME = "JOURNALDEV_COUNTRIES.DB";

// database version
static final int DB_VERSION = 1;

// Creating table query
private static final String CREATE_TABLE = "create table " + TABLE_NAME + "(" + _ID
      + " INTEGER PRIMARY KEY AUTOINCREMENT, " + SUBJECT + " TEXT NOT NULL, "
+ DESC + " TEXT);";

public DatabaseHelper(Context context) {
   super(context, DB_NAME, null, DB_VERSION);
}

@Override
public void onCreate(SQLiteDatabase db) {
   db.execSQL(CREATE_TABLE);
}
```

```
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}
```

As discussed above we have overridden the onCreate() and onUpgrade() methods besides the constructor. We've assigned the names to the database and the table as JOURNALDEV_COUNTRIES.DB and COUNTRIES respectively. The index column is auto incremented whenever a new row is inserted. The column names for country and currency are "subject" and "description".

The DBManager classes is where the DatabaseHelper is initialized and the CRUD Operations are defined. Below is the code for this class:

DBManager.java

```
package com.journaldev.sqlite;

import android.content.ContentValues;

import android.content.Context;

import android.database.Cursor;

import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;


public class DBManager {

    private DatabaseHelper dbHelper;

    private Context context;
```

```
private SQLiteDatabase database;

public DBManager(Context c) {
    context = c;
}

public DBManager open() throws SQLException {
    dbHelper = new DatabaseHelper(context);
    database = dbHelper.getWritableDatabase();
    return this;
}

public void close() {
    dbHelper.close();
}

public void insert(String name, String desc) {
    ContentValues contentValue = new ContentValues();
    contentValue.put(DatabaseHelper.SUBJECT, name);
    contentValue.put(DatabaseHelper.DESC, desc);
    database.insert(DatabaseHelper.TABLE_NAME, null, contentValue);
}

public Cursor fetch() {
    String[] columns = new String[] { DatabaseHelper._ID, DatabaseHelper.SUBJECT,
DatabaseHelper.DESC };
    Cursor cursor = database.query(DatabaseHelper.TABLE_NAME, columns, null, null,
null, null, null);
```

```
    if (cursor != null) {
        cursor.moveToFirst();
    }
    return cursor;
  }

  public int update(long _id, String name, String desc) {
    ContentValues contentValues = new ContentValues();
    contentValues.put(DatabaseHelper.SUBJECT, name);
    contentValues.put(DatabaseHelper.DESC, desc);
    int i = database.update(DatabaseHelper.TABLE_NAME, contentValues,
DatabaseHelper._ID + " = " + _id, null);
    return i;
  }

  public void delete(long _id) {
    database.delete(DatabaseHelper.TABLE_NAME, DatabaseHelper._ID + "=" + _id, null);
  }

}
```

The CountryListActivity.java class is the activity which is launched when the application starts. Below is layout defined for it:
fragment_emp_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="https://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
```

```
<ListView
    android:id="@+id/list_view"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:dividerHeight="1dp"
    android:padding="10dp" >
</ListView>

<TextView
    android:id="@+id/empty"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:text="@string/empty_list_text" />

</RelativeLayout>
```

Here a ListView component is defined to included the records stored in the database. Initially the ListView would be empty hence a TextView is used to display the same.

CountryListActivity.java

```
package com.journaldev.sqlite;

import android.content.Intent;
import android.database.Cursor;
import android.os.Bundle;
import android.support.v4.widget.SimpleCursorAdapter;
import android.support.v7.app.ActionBarActivity;
```

```
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.TextView;

public class CountryListActivity extends ActionBarActivity {

   private DBManager dbManager;

   private ListView listView;

   private SimpleCursorAdapter adapter;

   final String[] from = new String[] { DatabaseHelper._ID,
         DatabaseHelper.SUBJECT, DatabaseHelper.DESC };

   final int[] to = new int[] { R.id.id, R.id.title, R.id.desc };

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

      setContentView(R.layout.fragment_emp_list);

      dbManager = new DBManager(this);
```

```
dbManager.open();

Cursor cursor = dbManager.fetch();


listView = (ListView) findViewById(R.id.list_view);

listView.setEmptyView(findViewById(R.id.empty));


adapter = new SimpleCursorAdapter(this, R.layout.activity_view_record, cursor, from,
to, 0);

adapter.notifyDataSetChanged();


listView.setAdapter(adapter);


// OnCLickListiner For List Items

listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

   @Override

   public void onItemClick(AdapterView<?> parent, View view, int position, long
viewId) {

      TextView idTextView = (TextView) view.findViewById(R.id.id);

      TextView titleTextView = (TextView) view.findViewById(R.id.title);

      TextView descTextView = (TextView) view.findViewById(R.id.desc);


      String id = idTextView.getText().toString();

      String title = titleTextView.getText().toString();

      String desc = descTextView.getText().toString();


      Intent modify_intent = new Intent(getApplicationContext(),
ModifyCountryActivity.class);

      modify_intent.putExtra("title", title);
```

```
            modify_intent.putExtra("desc", desc);
            modify_intent.putExtra("id", id);

            startActivity(modify_intent);
        }
    });
}

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

@Override
public boolean onOptionsItemSelected(MenuItem item) {

    int id = item.getItemId();
    if (id == R.id.add_record) {

        Intent add_mem = new Intent(this, AddCountryActivity.class);
        startActivity(add_mem);

    }
    return super.onOptionsItemSelected(item);
}
```

}

In this activity the DBManager object is invoked to perform the CRUD Operations.

A SimpleCursorAdapter is defined to add elements to the list from the query results that are returned in an Cursor Object.
On list item click an intent is performed to open the ModifyCountryActivity class.

The menu contains an item to add a new record from the ActionBar. Here again an intent is performed to open the AddCountryActivity class. Below is menu.xml code.
menu.xml

```
<menu xmlns:android="https://schemas.android.com/apk/res/android"
   xmlns:app="https://schemas.android.com/apk/res-auto"
   xmlns:tools="https://schemas.android.com/tools"
   tools:context="com.example.sqlitesample.MainActivity" >

   <item
      android:id="@+id/add_record"
      android:icon="@android:drawable/ic_menu_add"
      android:orderInCategory="100"
      android:title="@string/add_record"
      app:showAsAction="always"/>

</menu>
```

The xml layout and code of AddCountryActivity.java file are defined below:
activity_add_record.xml

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

```
android:orientation="vertical"
android:padding="20dp" >

<EditText
   android:id="@+id/subject_edittext"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:ems="10"
   android:hint="@string/enter_title" >

   <requestFocus />
</EditText>

<EditText
   android:id="@+id/description_edittext"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:ems="10"
   android:hint="@string/enter_desc"
   android:inputType="textMultiLine"
   android:minLines="5" >
</EditText>

<Button
   android:id="@+id/add_record"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
```

```
    android:layout_gravity="center"
    android:text="@string/add_record" />

</LinearLayout>
```

Two EditText components that take the inputs for country and currency along with a button to add the values to the database and display it in the ListView are defined.

AddCountryActivity.java

```
package com.journaldev.sqlite;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class AddCountryActivity extends Activity implements OnClickListener {

  private Button addTodoBtn;
  private EditText subjectEditText;
  private EditText descEditText;

  private DBManager dbManager;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);

    setTitle("Add Record");

    setContentView(R.layout.activity_add_record);

    subjectEditText = (EditText) findViewById(R.id.subject_edittext);
    descEditText = (EditText) findViewById(R.id.description_edittext);

    addTodoBtn = (Button) findViewById(R.id.add_record);

    dbManager = new DBManager(this);
    dbManager.open();
    addTodoBtn.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.add_record:

            final String name = subjectEditText.getText().toString();
            final String desc = descEditText.getText().toString();

            dbManager.insert(name, desc);

            Intent main = new Intent(AddCountryActivity.this, CountryListActivity.class)
```

```
                .setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);


            startActivity(main);
            break;
        }
    }

}
```

The CRUD operation performed here is adding a new record to the database.

The xml layout and code of ModifyCountryActivity.java file are defined below:

activity_modify_record.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="https://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp" >

    <EditText
        android:id="@+id/subject_edittext"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="10dp"
        android:ems="10"
        android:hint="@string/enter_title" />
```

```
<EditText
   android:id="@+id/description_edittext"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:ems="10"
   android:hint="@string/enter_desc"
   android:inputType="textMultiLine"
   android:minLines="5" >
</EditText>


<LinearLayout
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:weightSum="2"
   android:gravity="center_horizontal"
   android:orientation="horizontal" >

   <Button
      android:id="@+id/btn_update"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="@string/btn_update" />

   <Button
      android:id="@+id/btn_delete"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/btn_delete" />
    </LinearLayout>

</LinearLayout>
```

It's similar to the previous layout except that modify and delete buttons are added.

ModifyCountryActivity.java

```
package com.journaldev.sqlite;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class ModifyCountryActivity extends Activity implements OnClickListener {

    private EditText titleText;
    private Button updateBtn, deleteBtn;
    private EditText descText;

    private long _id;
```

```
private DBManager dbManager;

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

    setTitle("Modify Record");

    setContentView(R.layout.activity_modify_record);

    dbManager = new DBManager(this);
    dbManager.open();

    titleText = (EditText) findViewById(R.id.subject_edittext);
    descText = (EditText) findViewById(R.id.description_edittext);

    updateBtn = (Button) findViewById(R.id.btn_update);
    deleteBtn = (Button) findViewById(R.id.btn_delete);

    Intent intent = getIntent();
    String id = intent.getStringExtra("id");
    String name = intent.getStringExtra("title");
    String desc = intent.getStringExtra("desc");

    _id = Long.parseLong(id);
```

```
    titleText.setText(name);
    descText.setText(desc);

    updateBtn.setOnClickListener(this);
    deleteBtn.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.btn_update:
            String title = titleText.getText().toString();
            String desc = descText.getText().toString();

            dbManager.update(_id, title, desc);
            this.returnHome();
            break;

        case R.id.btn_delete:
            dbManager.delete(_id);
            this.returnHome();
            break;
    }
}

public void returnHome() {
    Intent home_intent = new Intent(getApplicationContext(), CountryListActivity.class)
```

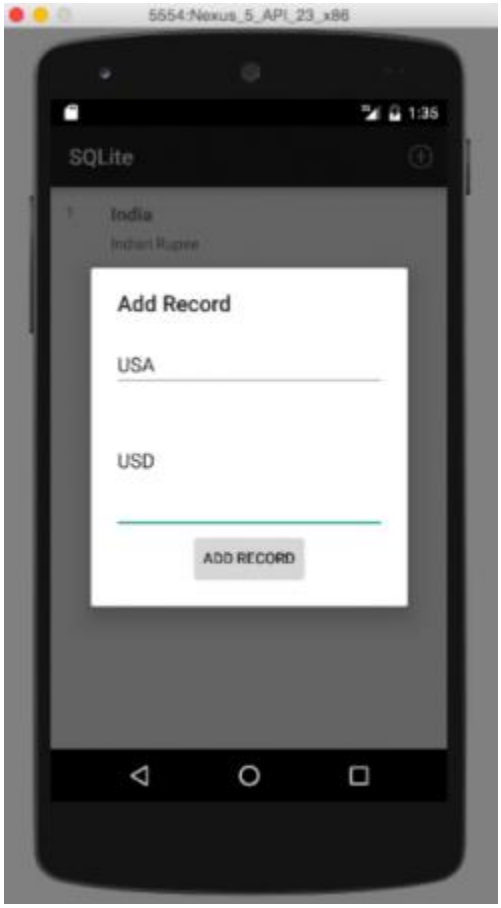
```
            .setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);

        startActivity(home_intent);

    }

}
```

The CRUD operations performed here are updating and deleting a record.

The below images are the screenshots of the final output of our project.

The first image is the output seen when the application is launched for the first time.

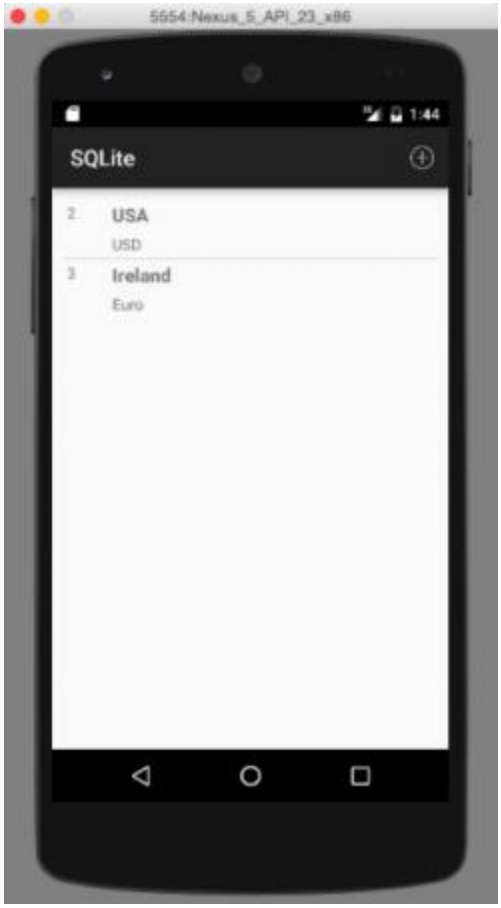
The second image is the result of clicking the menu option from the ActionBar to add a new record as shown below.



The third image shows an output when 3 records are added :

The fourth image shows the output when any list item is clicked to modify or delete a record :

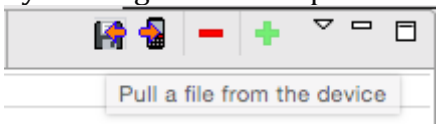
The final image is the output when a record is deleted. In this example we delete the first record :



Opening the Android SQLite Database file

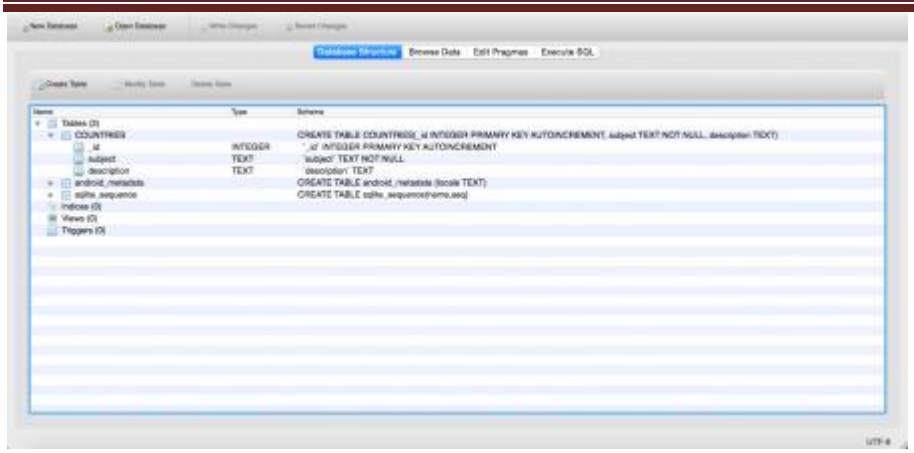
As we've discussed earlier in this tutorial, the database file is stored in the internal storage that is accessible from the Android Device Monitor as visible in the pic below.

To view this database we need to pull this file from the device to our desktop. This is done by clicking the menu option in the top right as seen in the image below :



To open this file download the SQLiteBrowser from this link.

The snippets below show the schema and tables in the browser.



To view the table go to the Browse Data tab on top. The following image is seen:

	_id	subject	description
	Filter	Filter	Filter
1	2	USA	USD
2	3	Ireland	Euro

This brings an end to Android SQLite tutorial.

SQLiteDatabase

In Android, the SQLiteDatabase namespace defines the functionality to connect and manage a database. It provides functionality to create, delete, manage and display database content.

Simple steps to create a database and handle are as follows.

1. Create "SQLiteDatabase" object.
2. Open or Create a database and create a connection.
3. Perform insert, update or delete operation.
4. Create a Cursor to display data from the table of the database.
5. Close the database connectivity.

Following tutorial helps you to create a database and insert records in it.

Step 1: Instantiate "SQLiteDatabase" object

```
SQLiteDatabase db;
```

Before you can use the above object, you must import the android.database.sqlite.SQLiteDatabase namespace in your application.

```
db=openOrCreateDatabase(String path, int mode, SQLiteDatabase.CursorFactory fact
ory)
```

This method is used to create/open database. As the name suggests, it will open a database connection if it is already there, otherwise, it will create a new one.

Example,

```
db=openOrCreateDatabase("XYZ_Database",SQLiteDatabase.CREATE_IF_NECESSARY,
null);
```

Arguments:

String path	Name of the database
Int mode	operating mode. Use 0 or "MODE_PRIVATE" for the default operation, or "CREATE_IF_NECESSARY" if you like to give an option that "if a database is not there, create it"
CursorFactory factory	An optional factory class that is called to instantiate a cursor when a query is called

Step 2: Execute DDL command

```
db.execSQL(String sql) throws SQLException
```

This command is used to execute a single SQL statement that doesn't return any data means other than SELECT or any other.

```
db.execSQL("Create Table Temp (id Integer, name Text)");
```

In the above example, it takes "CREATE TABLE" statement of SQL. This will create a table of "Integer" & "Text" fields.

Try and Catch block is required while performing this operation. An exception that indicates there was an error with SQL parsing or execution.

Step 3: Create an object of "ContentValues" and Initiate it.

```
ContentValues values=new ContentValues();
```

This class is used to store a set of values. We can also say, it will map ColumnName and relevant ColumnValue.

```
values.put("id", eid.getText().toString());

values.put("name", ename.getText().toString());
```

String Key	Name of the field as in table. Ex. "id", "name"
String Value	Value to be inserted.

Step 4: Perform Insert Statement.

```
insert(String table, String nullColumnHack, ContentValues values)
```

String table	Name of table related to the database.
String nullColumnHack	If not set to null, the nullColumnHack parameter provides the name of nullable column name to explicitly insert a NULL into in the case where your *values* is empty.
ContentValues values	This map contains the initial column values for the row.

This method returns a long. The row ID of the newly inserted row, or -1 if an error occurred.

Example,

```
db.insert("temp", null, values);
```

Step 5: Create Cursor

This interface provides random read-write access to the result set returned by a database query.

```
Cursor c=db.rawQuery(String sql, String[] selectionArgs)
```

Strign sql	The SQL query
String []selectionArgs	You may include ?s in where clause in the query, which will be replaced by the values from selectionArgs. The values will be bound as Strings.

Example,

```
Cursor c=db.rawQuery("SELECT * FROM temp",null);
```

Methods

moveToFirst	Moves cursor pointer at a first position of a result set
moveToNext	Moves cursor pointer next to the current position.
isAfterLast	Returns false, if the cursor pointer is not atlast position of a result set.

Example,
c.moveToFirst();

```
while(!c.isAfterLast())

{

   //statementâ€¦

c.moveToNext();

}
```

Step 6: Close Cursor and Close Database connectivity

It is very important to release our connections before closing our activity. It is advisable to release the Database connectivity in "onStop" method. And Cursor connectivity after use it.

DatabaseDemoActivity.java

```
package com.DataBaseDemo;
import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class DataBaseDemoActivity extends Activity {
   /** Called when the activity is first created. */
    SQLiteDatabase db;
Button btnInsert;
    @Override
    public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    btnInsert=(Button)findViewById(R.id.button1);
    try{
    db=openOrCreateDatabase("StudentDB",SQLiteDatabase.CREATE_IF_NECESSARY,null
);
    db.execSQL("Create Table Temp(id integer,name text)");
    }catch(SQLException e)
    {
    }
    btnInsert.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
        // TODO Auto-generated method stub
        EditText eid=(EditText) findViewById(R.id.editText1);
        EditText ename=(EditText)findViewById(R.id.editText2);
        ContentValues values=new ContentValues();
```

```
        values.put("id", eid.getText().toString());
        values.put("name", ename.getText().toString());
        if((db.insert("temp", null, values))!=-1)
        {
        Toast.makeText(DataBaseDemoActivity.this, "Record Successfully Inserted", 2000).
show();
        }
        else
        {
        Toast.makeText(DataBaseDemoActivity.this, "Insert Error", 2000).show();
        }
        eid.setText("");
        ename.setText("");
        Cursor c=db.rawQuery("SELECT * FROM temp",null);
        c.moveToFirst();
        while(!c.isAfterLast())
        {
        Toast.makeText(DataBaseDemoActivity.this,c.getString(0)+ " "+c.getString(1),1000
).show();
        c.moveToNext();
        }
        c.close();
        }
      });
   }
   @Override
   protected void onStop() {
    // TODO Auto-generated method stub
    db.close();
    super.onStop();
   }
}
```

Note:

If you want to see where your database stored? Follow below instruction.

1. Start Your Emulator (It is necessary to start Emulator to see File Explorer content)
2. Open "File Explorer"
3. Data -> Data -> find your "package" -> databases -> "database"

SQLite Transaction

Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

A transaction is the propagation of one or more changes to the database. For example, if you are creating, updating, or deleting a record from the table, then you

are performing transaction on the table. It is important to control transactions to ensure data integrity and to handle database errors.

SQLite & ACID

SQLite is a transactional database that all changes and queries are atomic, consistent, isolated, and durable (ACID).

SQLite guarantees all the transactions are ACID compliant even if the transaction is interrupted by a program crash, operation system dump, or power failure to the computer.

- **A**tomic: a transaction should be atomic. It means that a change cannot be broken down into smaller ones. When you commit a transaction, either the entire transaction is applied or not.
- **C**onsistent: a transaction must ensure to change the database from one valid state to another. When a transaction starts and executes a statement to modify data, the database becomes inconsistent. However, when the transaction is committed or rolled back, it is important that the transaction must keep the database consistent.
- **I**solation: a pending transaction performed by a session must be isolated from other sessions. When a session starts a transaction and executes the INSERT or UPDATE statement to change the data, these changes are only visible to the current session, not others. On the other hand, the changes committed by other sessions after the transaction started should not be visible to the current session.
- Durable: if a transaction is successfully committed, the changes must be permanent in the database regardless of the condition such as power failure or program crash. On the contrary, if the program crashes before the transaction is committed, the change should not persist.

SQLite transaction statements

By default, SQLite operates in auto-commit mode. It means that for each command, SQLite starts, processes, and commits the transaction automatically.

To start a transaction explicitly, you use the following steps:

First, open a transaction by issuing the BEGIN TRANSACTION command.

```
BEGIN TRANSACTION;
```

After executing the statement BEGIN TRANSACTION, the transaction is open until it is explicitly committed or rolled back.
Second, issue SQL statements to select or update data in the database. Note that the change is only visible to the current session (or client).

Third, commit the changes to the database by using the COMMIT or COMMIT TRANSACTION statement.

```
COMMIT;
```

If you do not want to save the changes, you can roll back using the ROLLBACK or ROLLBACK TRANSACTION statement:

```
ROLLBACK;
```

SQLite transaction example

We will create two new tables: accounts and account_changes for the demonstration.
The accounts table stores data about the account numbers and their balances.
The account_changes table stores the changes of the accounts.
First, create the accounts and account_changes tables by using the following CREATE TABLE statements:

```
CREATE TABLE accounts (
   account_no INTEGER NOT NULL,
   balance DECIMAL NOT NULL DEFAULT 0,
   PRIMARY KEY(account_no),
      CHECK(balance >= 0)
);

CREATE TABLE account_changes (
   change_no INT NOT NULL PRIMARY KEY,
   account_no INTEGER NOT NULL,
   flag TEXT NOT NULL,
   amount DECIMAL NOT NULL,
   changed_at TEXT NOT NULL
);
```

Second, insert some sample data into the accounts table.

```
INSERT INTO accounts (account_no,balance)
VALUES (100,20100);

INSERT INTO accounts (account_no,balance)
VALUES (200,10100);
```

Third, query data from the accounts table:

```
SELECT * FROM accounts;
```

account_no	balance
100	20,100
200	10,100

Fourth, transfer 1000 from account 100 to 200, and log the changes to the table account_changes in a single transaction.

```
BEGIN TRANSACTION;
UPDATE accounts
   SET balance = balance - 1000
 WHERE account_no = 100;

UPDATE accounts
   SET balance = balance + 1000
 WHERE account_no = 200;

INSERT INTO account_changes(account_no,flag,amount,changed_at)
VALUES(100,'-',1000,datetime('now'));

INSERT INTO account_changes(account_no,flag,amount,changed_at)
VALUES(200,'+',1000,datetime('now'));
COMMIT;
```

Fifth, query data from the accounts table:

```
SELECT * FROM accounts;
```

account_no	balance
100	19,100
200	11,100

As you can see, balances have been updated successfully.

Sixth, query the contents of the account_changes table:

```
SELECT * FROM account_changes;
```

change_no	account_no	flag	amount	changed_at
1	100	-	1,000	2019-08-19 10:33:01
2	200	+	1,000	2019-08-19 10:33:04

Let's take another example of rolling back a transaction.
First, attempt to deduct 20,000 from account 100:

```
BEGIN TRANSACTION;

UPDATE accounts
   SET balance = balance - 20000
 WHERE account_no = 100;

INSERT INTO account_changes(account_no,flag,amount,changed_at)
VALUES(100,'-',20000,datetime('now'));
```

SQLite issued an error due to not enough balance:

```
[SQLITE_CONSTRAINT]  Abort due to constraint violation (CHECK constraint failed: accounts)
```

However, the log has been saved to the account_changes table:

```
SELECT * FROM account_changes;
```

change_no	account_no	flag	amount	changed_at
1	100	-	1,000	2019-08-19 10:48:38
2	200	+	1,000	2019-08-19 10:48:40
3	100	-	20,000	2019-08-19 10:54:07

Second, roll back the transaction by using the ROLLBACK statement:

```
ROLLBACK;
```

Finally, query data from the account_changes table, you will see that the change no #3 is not there anymore:

```
SELECT * FROM account_changes;
```

change_no	account_no	flag	amount	changed_at
1	100	-	1,000	2019-08-19 10:48:38
2	200	+	1,000	2019-08-19 10:48:40

Unit-VI Security and Application Deployment

Course Outcome:

Publish Android applications.

Unit Outcomes:

6a. Explain the given location based service.
6b. Write the steps to customize the given permissions for users.
6c. Explain features of the given android security service.
6d. Write the steps to publish the given android App.

Contents:

6.1 SMS Telephony
6.2 Location Based Services: Creating the project, Getting the maps API key, Displaying the map, Displaying the zoom control, Navigating to a specific location, Adding markers, Getting location, Geocoding and reverse Geocoding, Getting Location data, Monitoring Location.
6.3 Android Security Model, Declaring and Using Permissions, Using Custom Permission.
6.4 Application Deployment: Creating Small Application, Signing of application, Deploying app on Google Play Store, Become a Publisher, Developer Console

---

6.1 SMS Telephony

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 a built-in SMS app to send SMS from our application.

Classes of SMS Telephony

1. Telephony.Sms.Conversations
   Contains a view of SMS conversations (also referred to as threads).

2. Telephony.Sms.Draft
   Contains all draft text-based SMS messages in the SMS app.

3. Telephony.Sms.Inbox
   Contains all text-based SMS messages in the SMS app inbox.

4. Telephony.Sms.Intents
   Contains constants for SMS related Intents that are broadcast.

5. Telephony.Sms.Outbox
   Contains all pending outgoing text-based SMS messages.

6. Telephony.Sms.Sent
   Contains all sent text-based SMS messages in the SMS app.

Android Send 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

In android, Intent is a messaging object which is used to request an action from another app component such as activities, services, broadcast receivers, and content providers. To know more about an Intent object in android check this Android Intents with Examples.

To send SMS using the 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.

Android Send SMS Example

Create a new android application using android studio and give names as SendSMSExample. 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: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.tutlane.sendsmsexample path and write the code like as shown below

MainActivity.java

```
package com.tutlane.sendsmsexample;
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();
            }
         }
      });
```

```
    }
}
```

If you observe above code, we are sending SMS using SMSManager api on button click. As discussed, we need to add a SEND_SMS permission in our android manifest.

Now open android manifest file (AndroidManifest.xml) and write the code like as shown below

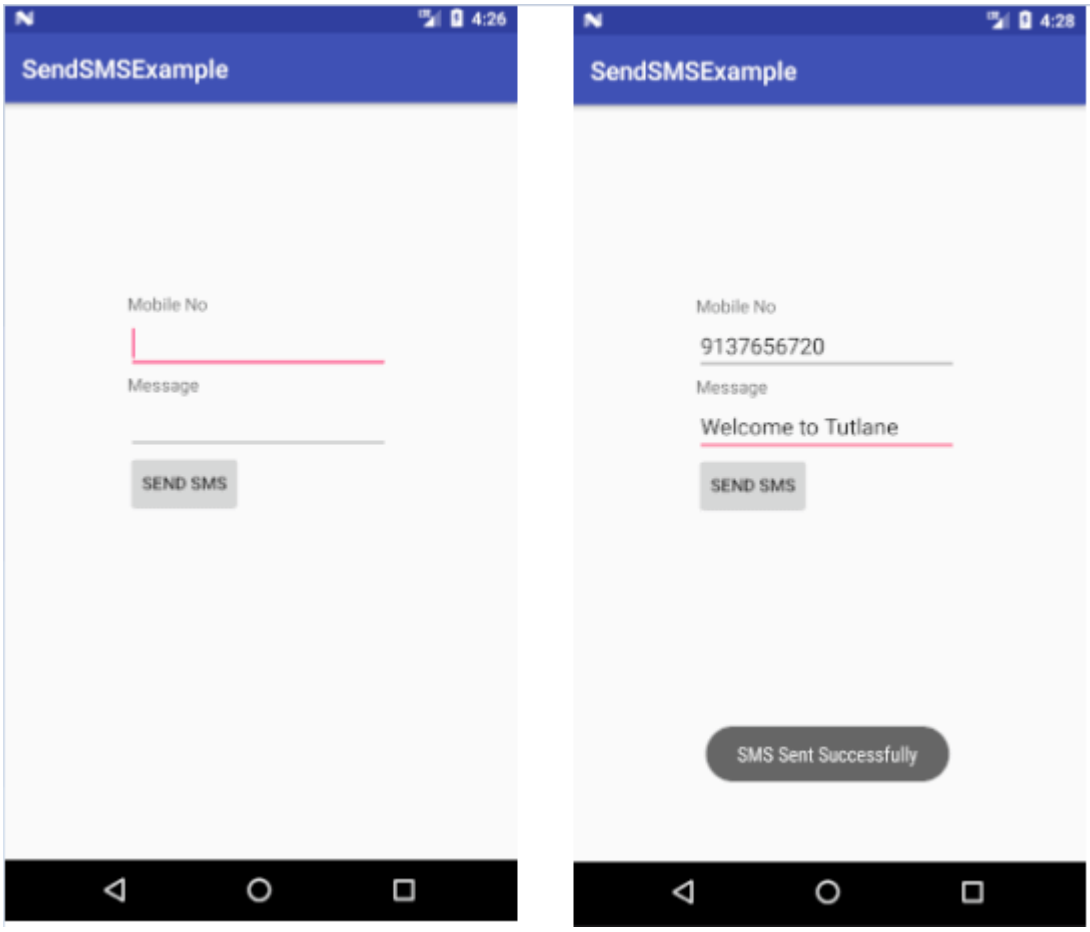
AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.tutlane.sendsmsexample">
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <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>
```

If you observe above AndroidManifest.xml file, we added a SEND_SMS permissions in manifest file.

Output of Android Send SMS Example

When we run above program in android studio we will get the result like as shown below.



Once you enter all details and click on Send SMS button it will send SMS and show the alert message like as mentioned in above image. The above example we implemented using SMSManager API. In case if we want to use Intents to send SMS to 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();
      }
    }
});
```

This is how we can send SMS using either SMSManager API or Intent objects in android applications based on our requirements.

6.2 Location Based Services

Location Based Services are provided by Android through its location framework. The framework provides a location API which consists of certain classes and interface. These classes and interface are the key components which allow us to develop Location Based Application in Android.

Classes and Interfaces of Location Based Services:

LocationManager – This class helps to get access to the location service of the system.
LocationListener – This interface acts as the listener which receives notification from the location manager when the location changes or the location provider is disabled or enabled.
Location – This is the class which represents the geographic location returned at a particular time.

Android Google Map

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:

1. **Normal:** This type of map displays typical road map, natural features like river and some features build by humans.
2. **Hybrid:** This type of map displays satellite photograph data with typical road maps. It also displays road and feature labels.
3. **Satellite:** Satellite type displays satellite photograph data, but doesn't display road and feature labels.
4. **Terrain:** This type displays photographic data. This includes colors, contour lines and labels and perspective shading.
5. **None:** This type displays an empty grid with no tiles loaded.

Syntax of different types of map

1. googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
2. googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
3. googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
4. 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.

Google Map - Layout file

Now you have to add the map fragment into xml layout file. Its syntax is given below –

```
<fragment
  android:id="@+id/map"
  android:name="com.google.android.gms.maps.MapFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"/>
```

Google Map - AndroidManifest file

The next thing you need to do is to add some permissions along with the Google Map API key in the AndroidManifest.XML file. Its syntax is given below –

```
<!--Permissions-->

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="com.google.android.providers.gsf.permission.
  READ_GSERVICES" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<!--Google MAP API key-->

<meta-data
  android:name="com.google.android.maps.v2.API_KEY"
  android:value="AIzaSyDKymeBXNeiFWY5jRUejv6zItpmr2MVyQ0" />
```

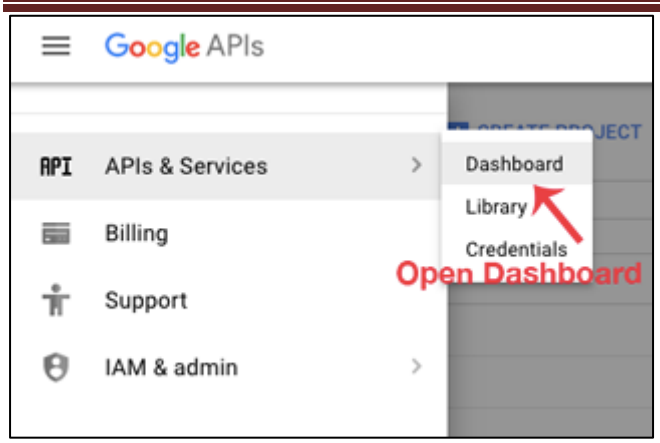
Getting the maps API key

An API key is needed to access the Google Maps servers. This key is free and you can use it with any of your applications. If you haven't created project, you can follow the below steps to get started:

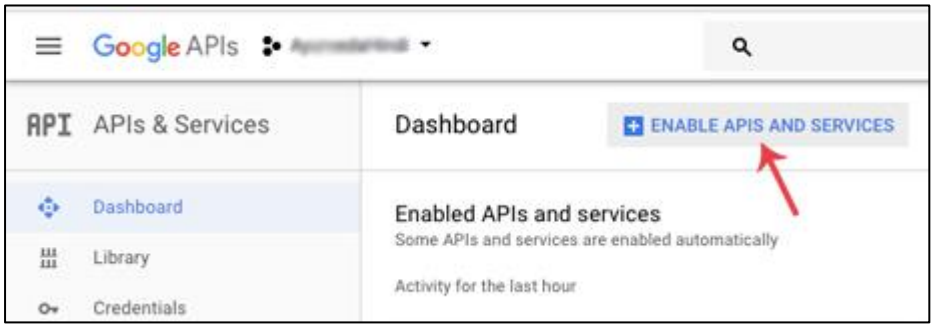
Step 1: Open Google developer console and signin with your gmail account: https://console.developers.google.com/project

Step 2: Now create new project. You can create new project by clicking on the **Create Project** button and give name to your project.

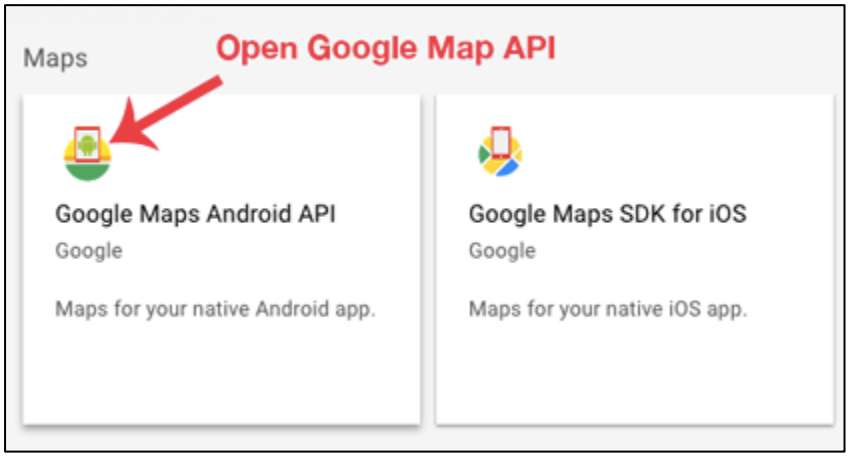
Step 3: Now click on APIs & Services and open **Dashboard** from it.



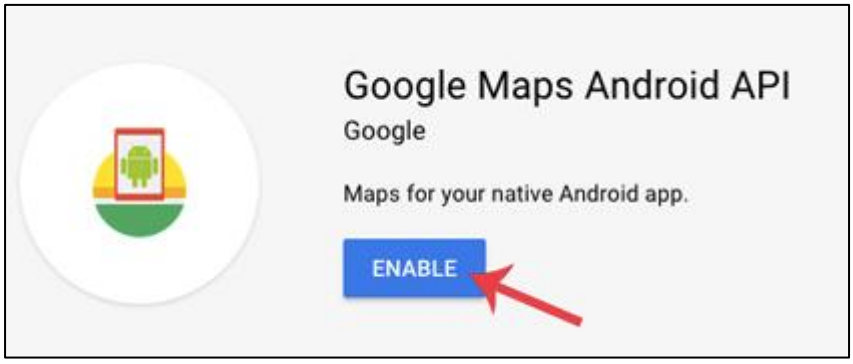
Step 4: In this open **Enable APIS AND SERICES**.



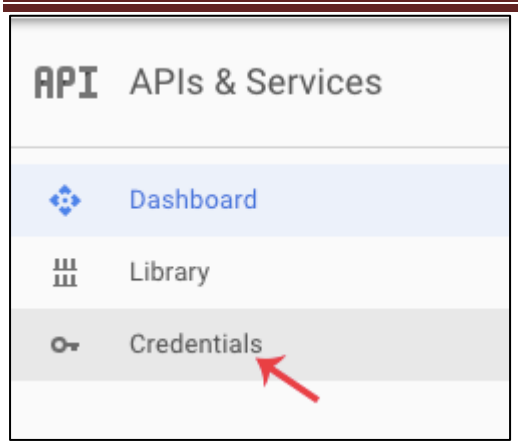
Step 5: Now open Google Map Android API.



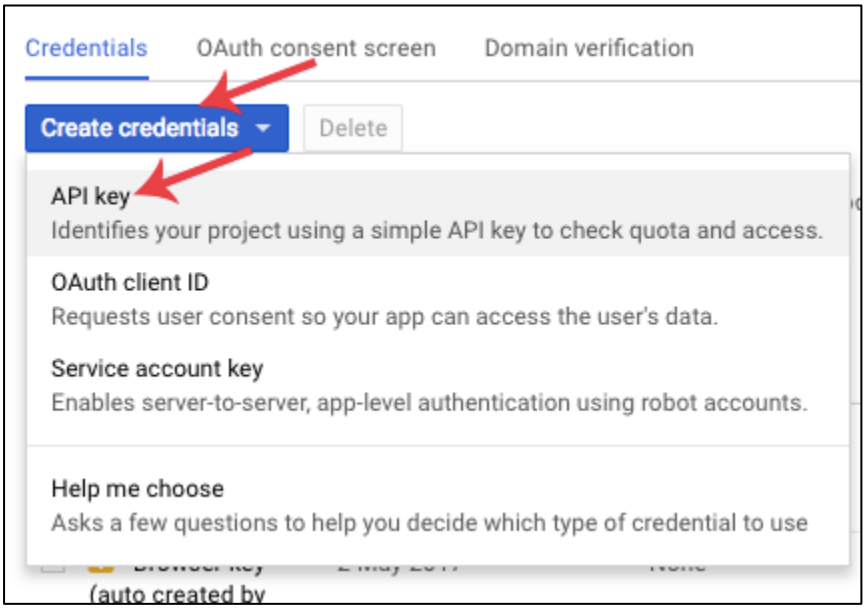
Step 6: Now enable the Google Maps Android API.



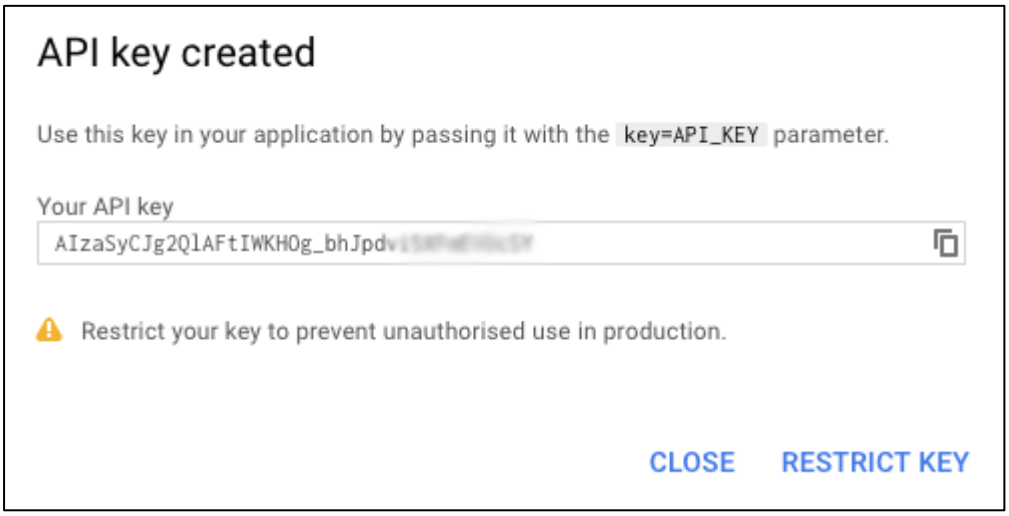
Step 6: Now go to **Credentials**



Step 7: Here click on Create credentials and choose API key



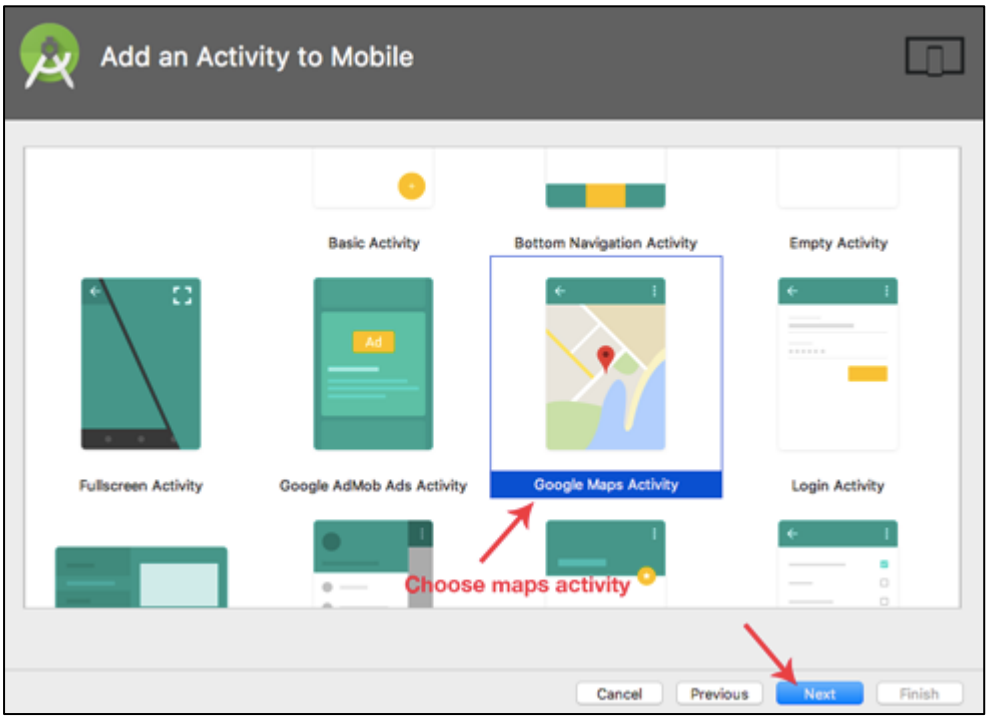
Step 8: Now API your API key will be generated. Copy it and save it somewhere as we will need it when implementing Google Map in our Android project.



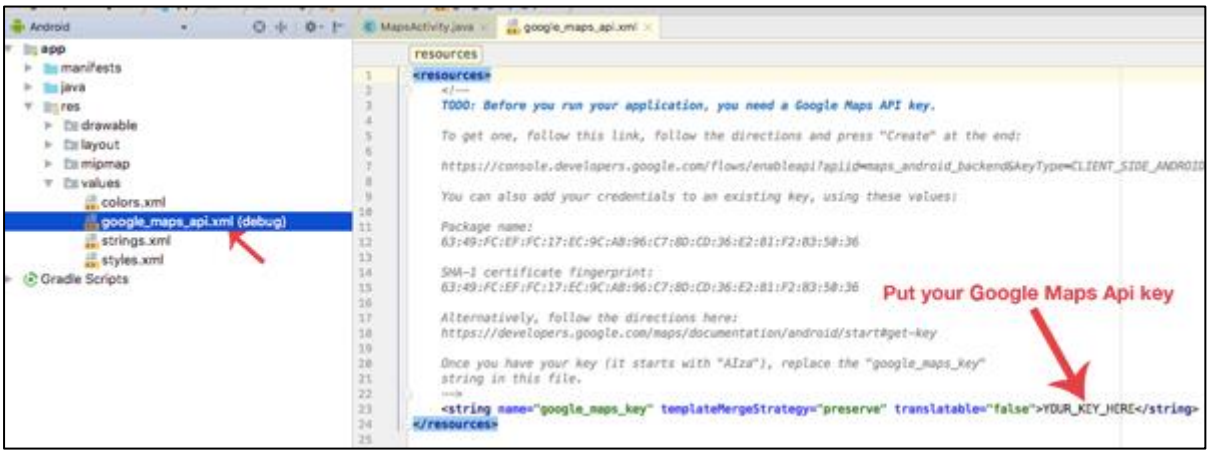
Creating the project, displaying the map, displaying the zoom control

Step 1: Create a New Android Project and name it GoogleMaps.

Step 2: Now select Google Maps Activity and then click Next and finish.



Step 3: Now open **google_maps_api.xml** (debug) in values folder



Step 4: Here enter your Google Maps API key in place of YOUR_KEY_HERE.

Step 5: Now open build.gradle and add compile 'com.google.android.gms:play-services:8.4.0' in dependencies

build.gradle code

```
apply plugin: 'com.android.application'

android {
   compileSdkVersion 26
   buildToolsVersion "26.0.2"
   defaultConfig {
```

```
        applicationId "com.abhiandroid.GoogleMaps.googlemaps"
        minSdkVersion 15
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:26.+'
    compile 'com.google.android.gms:play-services:8.4.0'
    testCompile 'junit:junit:4.12'
}
```

Step 6: Now open activity_maps.xml and add a fragment code in it

Here add a fragment element to the activity's layout file to define a Fragment object. In this element, set the android:name attribute to "com.google.android.gms.maps.MapFragment". This automatically attaches a MapFragment to the activity. The following layout file contains a fragment element:

activity_maps.xml code

```
<fragment android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```
  xmlns:map="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="com.abhiandroid.GoogleMaps.googlemaps.MapsActivity"/>
```

Step 6: Now define internet and location permissions in Android Manifest

INTERNET – To determine if we are connected to Internet or not. **ACCESS_FINE_LOCATION** – To determine user's location using GPS. It will give us precise location.

AndroidManifest.xml code:

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

Step 7: Now we will code MapsActivity.java file for inserting callbacks in Google Maps:

-OnMapReadyCallback: This callback is called when the map is ready to be used

```
@Override
public void onMapReady(GoogleMap googleMap) {}
```

-GoogleApiClient.ConnectionCallbacks: This callback is called whenever device is connected and disconnected and implement onConnected() and onConnectionSuspended() functions.

```
//When the connect request has successfully completed
@Override
public void onConnected(Bundle bundle) {}
//Called when the client is temporarily in a disconnected state.
@Override
public void onConnectionSuspended(int i) {
}
```

-GoogleApiClient.OnConnectionFailedListener: Provides callbacks for scenarios that result in a failed attempt to connect the client to the service. Whenever connection is failed onConnectionFailed() will be called.

```
@Override
public void onConnectionFailed(ConnectionResult connectionResult) {
}
```

-LocationListener: This callback have function onLocationChanged() that will be called whenever there is change in location of device.

```
@Override
public void onLocationChanged(Location location) {}
```

-onMapReady(): This function is called when the map is ready to be used.

-buildGoogleApiClient(): This method is used to initialize the Google Play Services.

```
@Override
public void onMapReady(GoogleMap googleMap) {
   mMap = googleMap;
   mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
```

```
    mMap.getUiSettings().setZoomControlsEnabled(true);
    mMap.getUiSettings().setZoomGesturesEnabled(true);
    mMap.getUiSettings().setCompassEnabled(true);

    //Initialize Google Play Services
    if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        if (ContextCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION)
                == PackageManager.PERMISSION_GRANTED) {
            buildGoogleApiClient();
            mMap.setMyLocationEnabled(true);
        }
    } else {
        buildGoogleApiClient();
        mMap.setMyLocationEnabled(true);
    }
}
```

-addConnectionCallbacks(): You need to call registers a listener to receive connection events from this GoogleApiClient.

-addOnConnectionFailedListener(): This methods adds a listener to register to receive connection failed events from this GoogleApiClient.

-GoogleApiClient.Builder: Builder is used to help construct the GoogleApiClient object and addApi () specify which Apis are requested by your app.

-mGoogleApiClient.connect(): A client must be connected before executing any operation.

```
protected synchronized void buildGoogleApiClient() {
 mGoogleApiClient = new GoogleApiClient.Builder(this)
 .addConnectionCallbacks(this)
 .addOnConnectionFailedListener(this)
 .addApi(LocationServices.API)
 .build();
 mGoogleApiClient.connect();
}
```

-Zoom Controls: The Maps API provides built-in zoom controls that appear in the bottom right hand corner of the map. These can be enabled by calling:

```
mMap.getUiSettings().setZoomControlsEnabled(true);
```

-Zoom Gestures:

ZoomIn: Double tap to increase the zoom level by 1.

Zoom Out: Two finger tap to decrease the zoom level by 1.

```
mMap.getUiSettings().setZoomGesturesEnabled(true);
```

-Compass: You can set compass by calling below method:

```
mMap.getUiSettings().setCompassEnabled(true);
```

-Changing the Map Type:

The Android Maps API provides normal, satellite, terrain and hybrid map types to help you out:

```
mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
mMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
```

MAP_TYPE_NORMAL : Represents a typical road map with street names and labels.

MAP_TYPE_SATELLITE: Represents a Satellite View Area without street names and labels.

MAP_TYPE_TERRAIN: Topographic data. The map includes colors, contour lines and labels, and perspective shading. Some roads and labels are also visible.

MAP_TYPE_HYBRID : Combines a satellite View Area and Normal mode displaying satellite images of an area with all labels.

Map_TYPE_NONE : No tiles. It is similar to a normal map, but doesn't display any labels or coloration for the type of environment in an area.

Add the following inside setUpMap() just below the setMyLocationEnabled() call:

The location of the user is updated at the regular intervals. We have used FusedLocationProvider. We have used requestLocationUpdates() method to get regular updates about a device's location. Do this in the onConnected() callback provided by Google API Client, which is called when the client is ready.

LocationRequest mLocationRequest is used to get quality of service for location updates from the FusedLocationProviderApi using requestLocationUpdates.

```
@Override
```

```
public void onConnected(Bundle bundle) {
   mLocationRequest = new LocationRequest();
   mLocationRequest.setInterval(1000);
   mLocationRequest.setFastestInterval(1000);
   mLocationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCUR
ACY);
   if (ContextCompat.checkSelfPermission(this,
         Manifest.permission.ACCESS_FINE_LOCATION)
         == PackageManager.PERMISSION_GRANTED) {
    LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
            mLocationRequest, this);
   }
}
```

Whenever user's location is changed. For that Google has predefined function onLocationChanged that will be called as soon as user's location change. Here we are getting the coordinates of current location using getLatitude() and getLongitude() and we are also adding Marker.

Complete code of MapsActivity.java class:

```
package com.abhiandroid.GoogleMaps.googlemaps;
import android.Manifest;
import android.content.Context;
import android.content.pm.PackageManager;
import android.location.Address;
import android.location.Criteria;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationManager;
import android.os.Build;
import android.os.Bundle;
import android.support.v4.app.ActivityCompat;
import android.support.v4.app.FragmentActivity;
import android.support.v4.content.ContextCompat;
import android.widget.Toast;
import com.google.android.gms.common.ConnectionResult;
```

```
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.location.LocationListener;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationServices;
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.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;
import com.abhiandroid.GoogleMaps.googlemaps.R;

import java.io.IOException;
import java.util.List;
import java.util.Locale;
public class MapsActivity extends FragmentActivity implements OnMapReadyCallback,
      GoogleApiClient.ConnectionCallbacks,
      GoogleApiClient.OnConnectionFailedListener,
      LocationListener {
   public static final int MY_PERMISSIONS_REQUEST_LOCATION = 99;
   GoogleApiClient mGoogleApiClient;
   Location mLastLocation;
   Marker mCurrLocationMarker;
   LocationRequest mLocationRequest;
   private GoogleMap mMap;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_maps);

      if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
```

```
        checkLocationPermission();
    }
    SupportMapFragment mapFragment = (SupportMapFragment)
          getSupportFragmentManager()
                .findFragmentById(R.id.map);

    mapFragment.getMapAsync(this);
  }
  @Override
  public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
    mMap.getUiSettings().setZoomControlsEnabled(true);
    mMap.getUiSettings().setZoomGesturesEnabled(true);
    mMap.getUiSettings().setCompassEnabled(true);
    //Initialize Google Play Services
    if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
      if (ContextCompat.checkSelfPermission(this,
          Manifest.permission.ACCESS_FINE_LOCATION)
          == PackageManager.PERMISSION_GRANTED) {
        buildGoogleApiClient();
        mMap.setMyLocationEnabled(true);
      }
    } else {
      buildGoogleApiClient();
      mMap.setMyLocationEnabled(true);
    }
  }
  protected synchronized void buildGoogleApiClient() {
    mGoogleApiClient = new GoogleApiClient.Builder(this)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .addApi(LocationServices.API)
```

```
            .build();
        mGoogleApiClient.connect();
    }
    @Override
    public void onConnected(Bundle bundle) {
        mLocationRequest = new LocationRequest();
        mLocationRequest.setInterval(1000);
        mLocationRequest.setFastestInterval(1000);
        mLocationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCU
RACY);
        if (ContextCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION)
                == PackageManager.PERMISSION_GRANTED) {
            LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
                    mLocationRequest, this);
        }
    }
    @Override
    public void onConnectionSuspended(int i) {
    }
    @Override
    public void onLocationChanged(Location location) {
        mLastLocation = location;
        if (mCurrLocationMarker != null) {
            mCurrLocationMarker.remove();
        }
//Showing Current Location Marker on Map
        LatLng latLng = new LatLng(location.getLatitude(), location.getLongitude());
        MarkerOptions markerOptions = new MarkerOptions();
        markerOptions.position(latLng);
        LocationManager locationManager = (LocationManager)
                getSystemService(Context.LOCATION_SERVICE);
        String provider = locationManager.getBestProvider(new Criteria(), true);
```

```
if (ActivityCompat.checkSelfPermission(this,
    Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_
GRANTED &&
    ActivityCompat.checkSelfPermission(this,  Manifest.permission.ACCESS_COARSE_
LOCATION)
        != PackageManager.PERMISSION_GRANTED) {
  return;
}
Location locations = locationManager.getLastKnownLocation(provider);
List<String> providerList = locationManager.getAllProviders();
if (null != locations && null != providerList && providerList.size() > 0) {
  double longitude = locations.getLongitude();
  double latitude = locations.getLatitude();
  Geocoder geocoder = new Geocoder(getApplicationContext(),
      Locale.getDefault());
  try {
    List<Address> listAddresses = geocoder.getFromLocation(latitude,
        longitude, 1);
    if (null != listAddresses && listAddresses.size() > 0) {
      String state = listAddresses.get(0).getAdminArea();
      String country = listAddresses.get(0).getCountryName();
      String subLocality = listAddresses.get(0).getSubLocality();
      markerOptions.title("" + latLng + "," + subLocality + "," + state
          + "," + country);
    }
  } catch (IOException e) {
    e.printStackTrace();
  }
}
markerOptions.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFacto
ry.HUE_BLUE));
mCurrLocationMarker = mMap.addMarker(markerOptions);
mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));
mMap.animateCamera(CameraUpdateFactory.zoomTo(11));
```

```
    if (mGoogleApiClient != null) {
        LocationServices.FusedLocationApi.removeLocationUpdates(mGoogleApiClient,
                this);
    }
}
@Override
public void onConnectionFailed(ConnectionResult connectionResult) {
}
public boolean checkLocationPermission() {
    if (ContextCompat.checkSelfPermission(this,
            Manifest.permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED) {

        if (ActivityCompat.shouldShowRequestPermissionRationale(this,
                Manifest.permission.ACCESS_FINE_LOCATION)) {
            ActivityCompat.requestPermissions(this,
                    new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                    MY_PERMISSIONS_REQUEST_LOCATION);
        } else {
            ActivityCompat.requestPermissions(this,
                    new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                    MY_PERMISSIONS_REQUEST_LOCATION);
        }
        return false;
    } else {
        return true;
    }
}
@Override
public void onRequestPermissionsResult(int requestCode,
                                       String permissions[], int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_LOCATION: {
```

```
            if (grantResults.length > 0
                && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
              if (ContextCompat.checkSelfPermission(this,
                  Manifest.permission.ACCESS_FINE_LOCATION)
                  == PackageManager.PERMISSION_GRANTED) {
                if (mGoogleApiClient == null) {
                  buildGoogleApiClient();
                }
                mMap.setMyLocationEnabled(true);
              }
            } else {
              Toast.makeText(this, "permission denied",
                  Toast.LENGTH_LONG).show();
            }
            return;
          }
        }
      }
    }
```

Output:

Now run the App. If you are connected to internet and provide access to your location then in Map you will see your current location.

Navigating to a specific location,

Navigating to a destination is done using a NavController, an object that manages app navigation within a NavHost. Each NavHost has its own corresponding NavController. NavController provides a few different ways to navigate to a destination, which are further described in the sections below.

To retrieve the NavController for a fragment, activity, or view, use one of the following methods:

- NavHostFragment.findNavController(Fragment)
- Navigation.findNavController(Activity, @IdRes int viewId)
- Navigation.findNavController(View)

After you've retrieved a NavController, you can call one of the overloads of navigate() to navigate between destinations. Each overload provides support for various navigation scenarios, as described in the following sections.

Adding markers,

Customizing Google Map

You can easily customize google map from its default view, and change it according to your demand.

Adding Marker

You can place a maker with some text over it displaying your location on the map. It can be done by via **addMarker()** method. Its syntax is given below –

```
final LatLng TutorialsPoint = new LatLng(21 , 57);
Marker TP = googleMap.addMarker(new MarkerOptions()
  .position(TutorialsPoint).title("TutorialsPoint"));
```

Changing Map Type

You can also change the type of the MAP. There are four different types of map and each give a different view of the map. These types are Normal,Hybrid,Satellite and terrain. You can use them as below

```
googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
```

Enable/Disable zoom

You can also enable or disable the zoom gestures in the map by calling the **setZoomControlsEnabled(boolean)** method. Its syntax is given below –

```
googleMap.getUiSettings().setZoomGesturesEnabled(true);
```

Apart from these customization, there are other methods available in the GoogleMap class , that helps you more customize the map. They are listed below –

Sr.No	Method & description
1	**addCircle(CircleOptions options)** This method add a circle to the map
2	**addPolygon(PolygonOptions options)** This method add a polygon to the map
3	**addTileOverlay(TileOverlayOptions options)** This method add tile overlay to the map
4	**animateCamera(CameraUpdate update)** This method Moves the map according to the update with an animation
5	**clear()** This method removes everything from the map.

6	**getMyLocation()** This method returns the currently displayed user location.
7	**moveCamera(CameraUpdate update)** This method repositions the camera according to the instructions defined in the update
8	**setTrafficEnabled(boolean enabled)** This method Toggles the traffic layer on or off.
9	**snapshot(GoogleMap.SnapshotReadyCallback callback)** This method Takes a snapshot of the map
10	**stopAnimation()** This method stops the camera animation if there is one in progress

Getting location

Appropriate use of location information can be beneficial to users of your app. For example, if your app helps the user find their way while walking or driving, or if your app tracks the location of assets, it needs to get the location of the device at regular intervals. As well as the geographical location (latitude and longitude), you may want to give the user further information such as the bearing (horizontal direction of travel), altitude, or velocity of the device. This information, and more, is available in the Location object that your app can retrieve from the fused location provider. In response, the API updates your app periodically with the best available location, based on the currently-available location providers such as WiFi and GPS (Global Positioning System). The accuracy of the location is determined by the providers, the location permissions you've requested, and the options you set in the location request.

Declare permissions

Apps that use location services must request location permissions. In most cases, you can request the coarse location permission and still get reasonably accurate location information from the available location providers.

The following snippet demonstrates how to request the coarse location permission:

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

Geocoding and reverse Geocoding

Geocoding is the process of converting addresses (like a street address) into geographic coordinates (like latitude and longitude), which you can use to place markers on a map, or position the map.

Reverse geocoding is the process of converting geographic coordinates into a human-readable address.

Geocoding is the process of finding the geographical coordinates (latitude and longitude) of a given address or location. Reverse Geocoding is the opposite of geocoding where a pair of latitude and longitude is converted into an address or location.

For achieving Geocode or Reverse Geocode you must first import the proper package.

```
import android.location.Geocoder;
```

The geocoding or reverse geocoding operation needs to be done on a separate thread and should never be used on the UI thread as it will cause the system to display an Application Not Responding (ANR) dialog to the user.

To Achieve Geocode, use the below code

```
Geocoder gc = new Geocoder(context);
if(gc.isPresent()){
List<Address> list = gc.getFromLocationName("155 Park Theater, Palo Alto, CA", 1);
Address address = list.get(0);
double lat = address.getLatitude();
double lng = address.getLongitude();
}
```

To Achieve Reverse Geocode, use the below code

```
Geocoder gc = new Geocoder(context);
if(gc.isPresent()){
List<address> list = gc.getFromLocation(37.42279, -122.08506,1);
Address address = list.get(0);
StringBuffer str = new StringBuffer();
str.append("Name: " + address.getLocality() + "\n");
str.append("Sub-Admin Ares: " + address.getSubAdminArea() + "\n");
str.append("Admin Area: " + address.getAdminArea() + "\n");
str.append("Country: " + address.getCountryName() + "\n");
str.append("Country Code: " + address.getCountryCode() + "\n");
String strAddress = str.toString();
}
```

Getting Location data

There are two types of location providers,

1. GPS Location Provider
2. Network Location Provider

Any one of the above providers is enough to get current location of the user or user's device. But, it is recommended to use both providers as they both have different advantages. Because, GPS provider will take time to get location at indoor area. And, the Network Location Provider will not get location when the network connectivity is poor.

Network Location Provider vs GPS Location Provider

- Network Location provider is comparatively faster than the GPS provider in providing the location co-ordinates.
- GPS provider may be very very slow in in-door locations and will drain the mobile battery.
- Network location provider depends on the cell tower and will return our nearest tower location.
- GPS location provider, will give our location accurately.

Steps to get location in Android

1. Provide permissions in manifest file for receiving location update
2. Create LocationManager instance as reference to the location service
3. Request location from LocationManager
4. Receive location update from LocationListener on change of location

Provide permissions for receiving location update

To access current location information through location providers, we need to set permissions with android manifest file.

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

ACCESS_COARSE_LOCATION is used when we use network location provider for our Android app. But, ACCESS_FINE_LOCATION is providing permission for both providers. INTERNET permission is must for the use of network provider.

Create LocationManager instance as reference to the location service

For any background Android Service, we need to get reference for using it. Similarly, location service reference will be created using getSystemService() method. This reference will be added with the newly created LocationManager instance as follows.

```
locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

Request current location from LocationManager

After creating the location service reference, location updates are requested using requestLocationUpdates() method of LocationManager. For this function, we need to send the type of location provider, number of seconds, distance and the LocationListener object over which the location to be updated.

```
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, this);
```

Receive location update from LocationListener on change of location

LocationListener will be notified based on the distance interval specified or the number seconds.

Monitoring Location.

6.3 Android Security Model

The Android security model is primarily based on a sandbox and permission mechanism. Each application is running in a specific Dalvik virtual machine with a unique user ID assigned to it, which means the application code runs in isolation from the code of all others applications. As a consequence, one application has not granted access to other applications' files.

Android application has been signed with a certificate with a private key Know the owner of the application is unique. This allows the author of The application will be identified if needed. When an application is installed in The phone is assigned a user ID, thus avoiding it from affecting it Other applications by creating a sandbox for it. This user ID is permanent on which devices and applications with the same user ID are allowed to run in a single process. This is a way to ensure that a malicious application has Can not access / compromise the data of the genuine application.

It is mandatory for an application to list all the resources it will Access during installation. Terms are required of an application, in The installation process should be user-based or interactive Check with the signature of the application.

Declaring and Using Permissions

The purpose of a permission is to protect the privacy of an Android user. Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet). Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request. (2)

Permissions are divided into several protection levels. The protection level affects whether runtime permission requests are required. There are three protection levels that affect third-party apps: *normal*, *signature*, and *dangerous* permissions.

Normal permissions cover areas where your app needs to access data or resources outside the app's sandbox, but where there's very little risk to the user's privacy or the operation of other apps. For example, permission to set the time zone is a normal permission. If an app declares in its manifest that it needs a normal permission, the system automatically grants the app that permission at install time. The system doesn't prompt the user to grant normal permissions, and users cannot revoke these permissions.

Signature permissions: The system grants these app permissions at install time, but only when the app that attempts to use permission is signed by the same certificate as the app that defines the permission.

Dangerous permissions cover areas where the app wants data or resources that involve the user's private information, or could potentially affect the user's stored data or the operation of other apps. For example, the ability to read the user's contacts is a dangerous permission. If an app declares that it needs a dangerous permission, the user has to explicitly grant the permission to the app. Until the user approves the permission, your app cannot provide functionality that depends on that permission. To use a dangerous permission, your app must prompt the user to grant permission at runtime. For more details about how the user is prompted, see Request prompt for dangerous permission.

Using Custom Permission

Apps can define their own custom permissions and request custom permissions from other apps by defining `<uses-permission>` elements. To enforce your own permissions, you must first declare them in your `AndroidManifest.xml` using one or more `<permission>` elements.

For example, an app that wants to control who can start one of its activities could declare a permission for this operation as follows:

```
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.myapp" >

    <permission
      android:name="com.example.myapp.permission.DEADLY_ACTIVITY"
      android:label="@string/permlab_deadlyActivity"
```

```
      android:description="@string/permdesc_deadlyActivity"
      android:permissionGroup="android.permission-group.COST_MONEY"
      android:protectionLevel="dangerous" />
    ...
</manifest>
```

The `protectionLevel` attribute is required, telling the system how the user is to be informed of apps requiring the permission, or who is allowed to hold that permission, as described in the linked documentation.

The `android:permissionGroup` attribute is optional, and only used to help the system display permissions to the user.

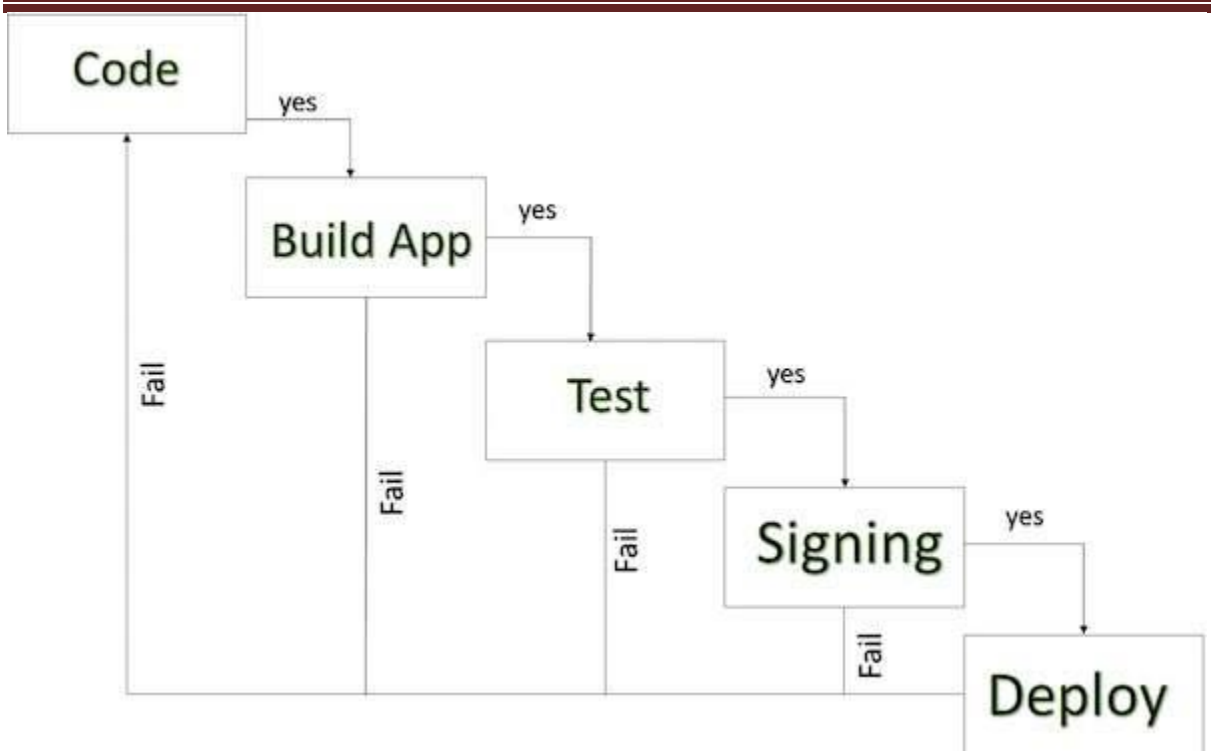
You need to supply both a label and description for the permission. These are string resources that the user can see when they are viewing a list of permissions (`android:label`) or details on a single permission (`android:description`). The label should be short; a few words describing the key piece of functionality the permission is protecting. The description should be a couple of sentences describing what the permission allows a holder to do. Our convention is a two-sentence description: the first sentence describes the permission, and the second sentence warns the user of the type of things that can go wrong if an app is granted the permission.

Here is an example of a label and description for the `CALL_PHONE` permission:

```
<string name="permlab_callPhone">directly call phone numbers</string>
<string name="permdesc_callPhone">Allows the app to call
    phone numbers without your intervention. Malicious apps may
    cause unexpected calls on your phone bill. Note that this does not
    allow the app to call emergency numbers.</string>
```

6.4 Application Deployment:

Android application publishing is a process that makes your Android applications available to users. Infect, publishing is the last phase of the Android application development process.



Android development life cycle

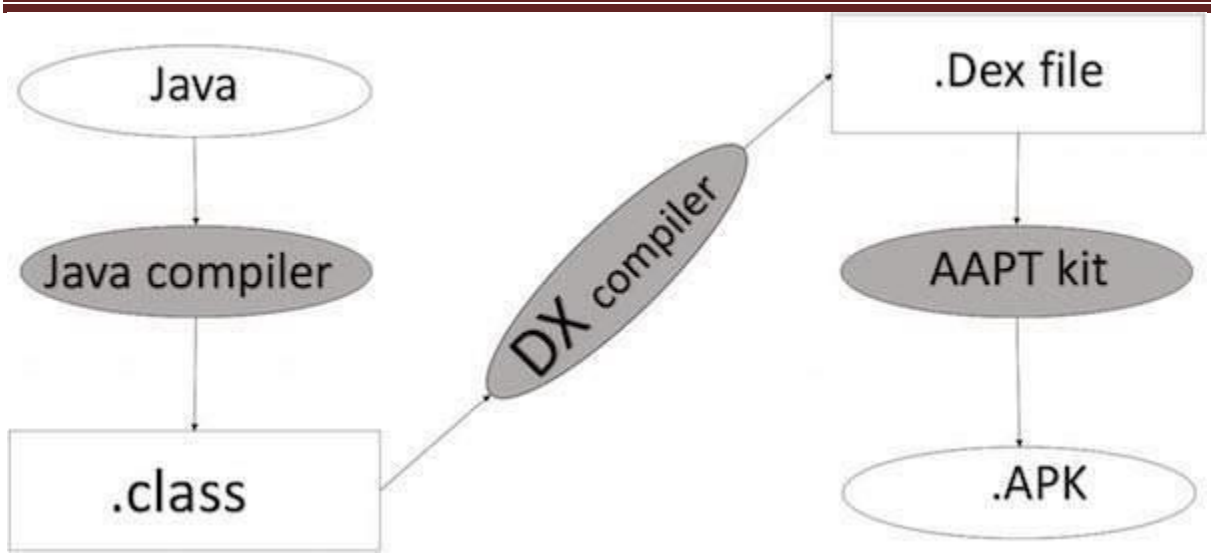
Once you developed and fully tested your Android Application, you can start selling or distributing free using Google Play (A famous Android marketplace). You can also release your applications by sending them directly to users or by letting users download them from your own website.

You can check a detailed publishing process at Android official website, but this tutorial will take you through simple steps to launch your application on Google Play. Here is a simplified check list which will help you in launching your Android application –

Step	Activity
1	**Regression Testing** Before you publish your application, you need to make sure that its meeting the basic quality expectations for all Android apps, on all of the devices that you are targeting. So perform all the required testing on different devices including phone and tablets.
2	**Application Rating** When you will publish your application at Google Play, you will have to specify a content rating for your app, which informs Google Play users of its maturity level. Currently available ratings are (a) Everyone (b) Low maturity (c) Medium maturity (d) High maturity.
3	**Targeted Regions** Google Play lets you control what countries and territories where your application will be sold. Accordingly you must take care of setting up time zone,

	localization or any other specific requirement as per the targeted region.
4	**Application Size** Currently, the maximum size for an APK published on Google Play is 50 MB. If your app exceeds that size, or if you want to offer a secondary download, you can use APK Expansion Files, which Google Play will host for free on its server infrastructure and automatically handle the download to devices.
5	**SDK and Screen Compatibility** It is important to make sure that your app is designed to run properly on the Android platform versions and device screen sizes that you want to target.
6	**Application Pricing** Deciding whether you app will be free or paid is important because, on Google Play, free app's must remain free. If you want to sell your application then you will have to specify its price in different currencies.
7	**Promotional Content** It is a good marketing practice to supply a variety of high-quality graphic assets to showcase your app or brand. After you publish, these appear on your product details page, in store listings and search results, and elsewhere.
8	**Build and Upload release-ready APK** The release-ready APK is what you you will upload to the Developer Console and distribute to users. You can check complete detail on how to create a release-ready version of your app: Preparing for Release.
9	**Finalize Application Detail** Google Play gives you a variety of ways to promote your app and engage with users on your product details page, from colourful graphics, screen shots, and videos to localized descriptions, release details, and links to your other apps. So you can decorate your application page and provide as much as clear crisp detail you can provide.

Export Android Application Process



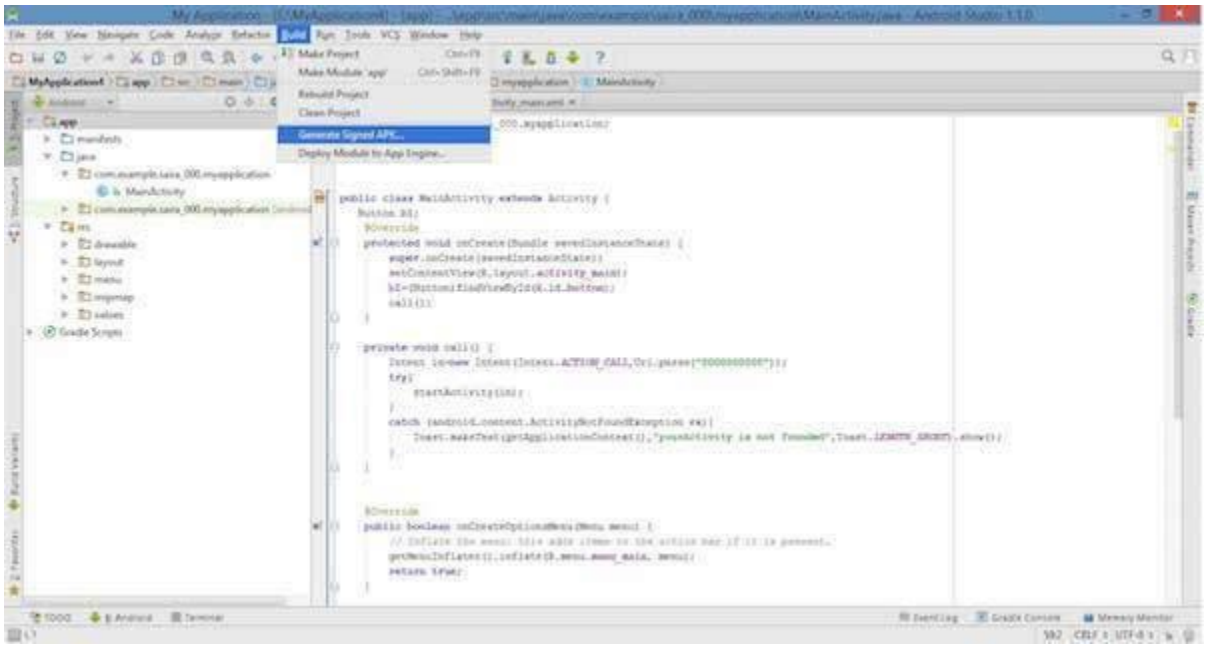
Apk development process

Before exporting the apps, you must some of tools

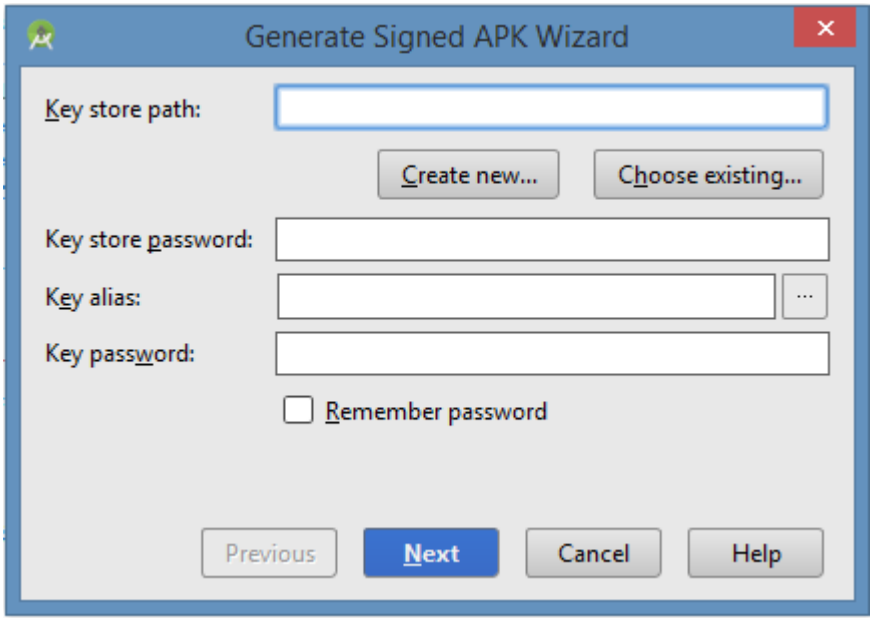
- **Dx tools**(Dalvik executable tools): It going to convert **.class file** to **.dex file**. it has useful for memory optimization and reduce the boot-up speed time
- **AAPT**(Android assistance packaging tool):it has useful to convert **.Dex file** to**.Apk**
- **APK**(Android packaging kit): The final stage of deployment process is called as .apk.

You will need to export your application as an APK (Android Package) file before you upload it Google Play marketplace.

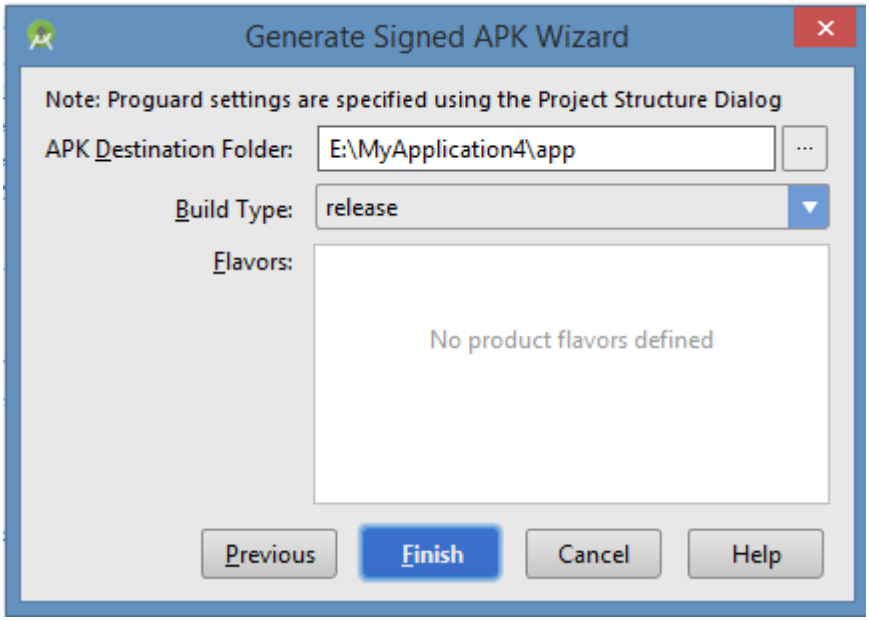
To export an application, just open that application project in Android studio and select **Build → Generate Signed APK** from your Android studio and follow the simple steps to export your application –



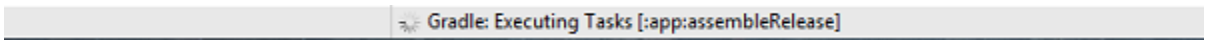
Next select, **Generate Signed APK** option as shown in the above screen shot and then click it so that you get following screen where you will choose **Create new keystore** to store your application.



Enter your key store path,key store password,key alias and key password to protect your application and click on **Next** button once again. It will display following screen to let you create an application –



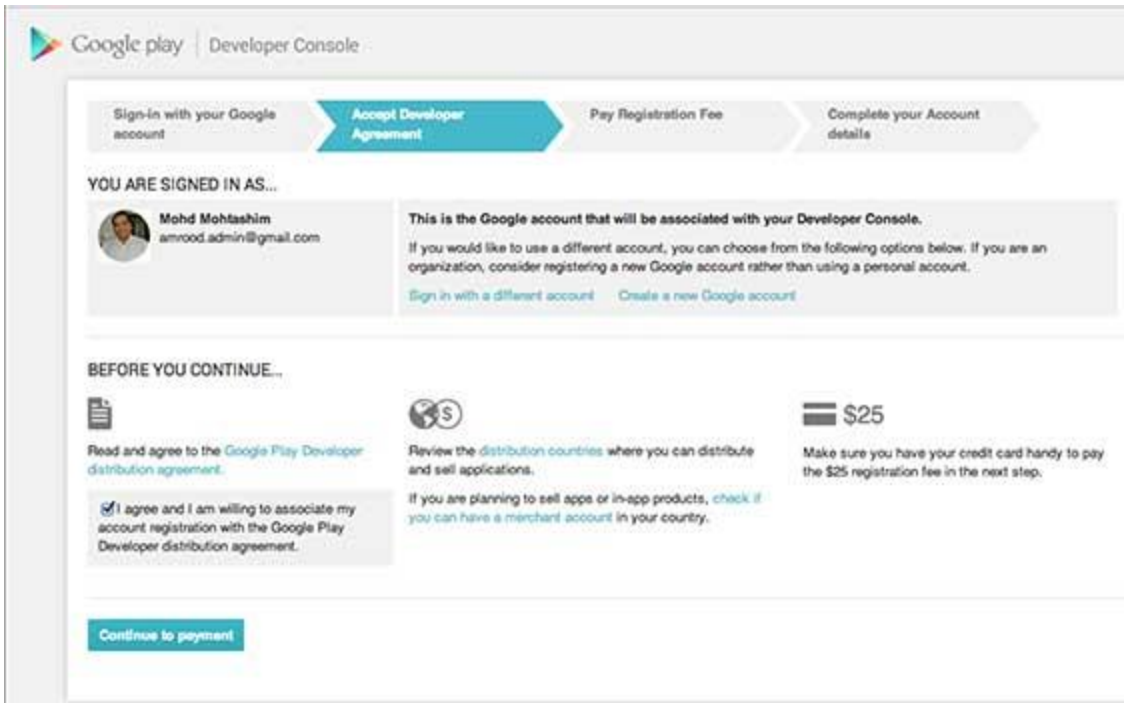
Once you filled up all the information,like app destination,build type and flavours click **finish** button While creating an application it will show as below



Finally, it will generate your Android Application as APK formate File which will be uploaded at Google Play marketplace.

Google Play Registration

The most important step is to register with Google Play using Google Play Marketplace. You can use your existing google ID if you have any otherwise you can create a new Google ID and then register with the marketplace. You will have following screen to accept terms and condition.



You can use **Continue to payment** button to proceed to make a payment of $25 as a registration fee and finally to complete your account detail.

Once you are a registered user at Google Play, you can upload **release-ready APK** for your application and finally you will complete application detail using application detail page as mentioned in step 9 of the above mentioned checklist.

Signing Your App Manually

You do not need Android Studio to sign your app. You can sign your app from the command line using standard tools from the Android SDK and the JDK. To sign an app in release mode from the command line –

- Generate a private key using keytool

```
$ keytool -genkey -v -keystore my-release-key.keystore
-alias alias_name -keyalg RSA -keysize 2048 -validity 10000
```

- Compile your app in release mode to obtain an unsigned APK
- Sign your app with your private key using jarsigner

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1
-keystore my-release-key.keystore my_application.apk alias_name
```

- Verify that your APK is signed. For example –

```
$ jarsigner -verify -verbose -certs my_application.apk
```

- Align the final APK package using zipalign.

```
$ zipalign -v 4 your_project_name-unaligned.apk your_project_name.apk
```

Steps steps to publish the Android application.

Step 1: Sign up. Sign up for an account on the **Android** Developer Console. ...

Step 2: Create a new **application**. ...

Step 3: Prepare multimedia. ...

Step 4: Prepare code for **release**. ...

Step 5: Build a **release**-ready APK. ...

Step 6: Upload APK. ...

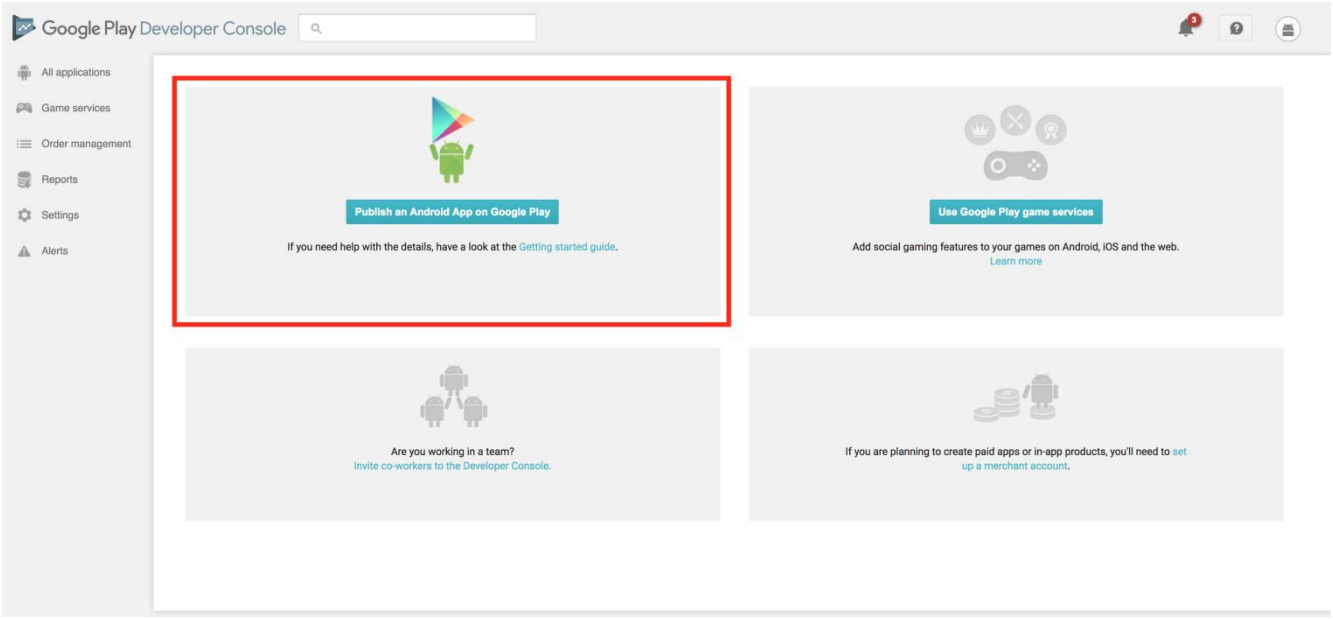
Complete the checklist on the left until all the items have a green checkmark.

Step 1: Sign up

Sign up for an account on the Android Developer Console. Creating an account costs $25.

Step 2: Create a new application

- On the Developer Console select the *Publish an Android Application* option.
- Fill out the details: Title, Short Description, Full Description.



Step 3: Prepare multimedia

- Screenshots: I used the android emulator to take screenshots of my app.
- Hi-res icon: I used the launcher icon. It was an SVG file, so I converted it to PNG using GIMP.

- Feature graphic: This is an image that shows up on the top of the app download page in Google Play on mobile phones.

Step 4: Prepare code for release

- Remove ***log*** statements.
- Remove the ***android:debuggable*** attribute from your manifest file. I didn't have to do this because Android Studio automatically sets this attribute based on the kind of APK its building. Neat!
- Set the ***android:versionCode*** attribute in the manifest tag in manifest.xml. Two important notes: (1) This must be an integer that increases with each release. (2) This number is not displayed to users.
  I chose "1".
- Set the ***android:versionName*** attribute in the manifest tag in manifest.xml. This string is shown to users and has no other purpose.
  I chose "1.0".

Step 5: Build a release-ready APK

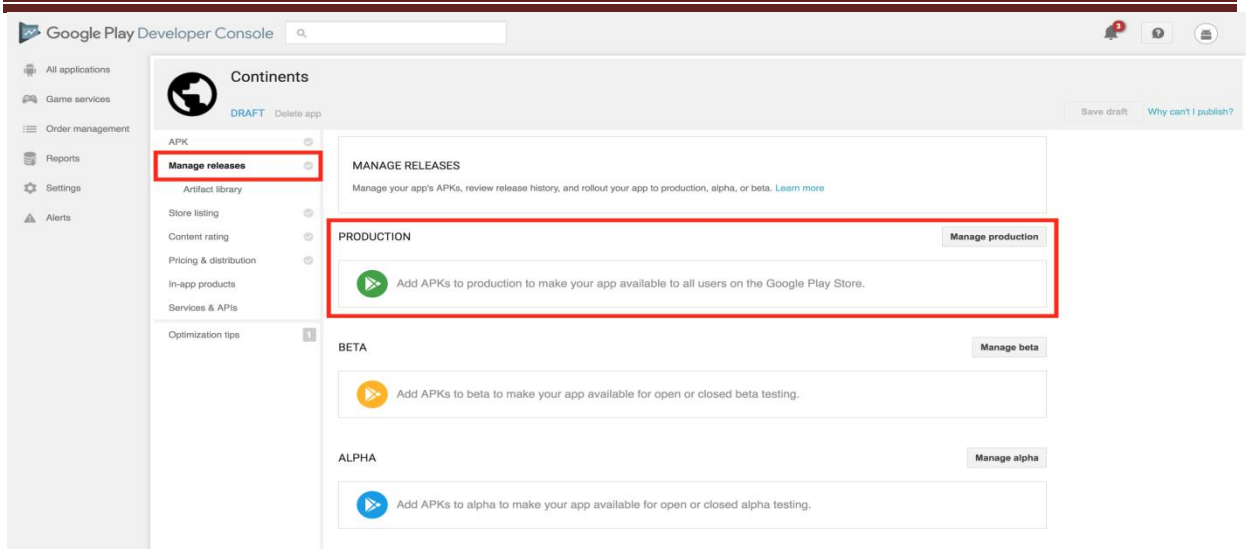
The release-ready APK is different from the debug APK in that it is signed with certificate that is owned by the developer. This is done to ensure that updates to the app come from a verified source, i.e. a developer with access to the private key.

I recommend you follow the instructions here to create a signed APK.

- Android Studio -> Build -> Generate Signed APK
- A Java Keystore (JKS) is a repository of public-private key pairs.
- You must sign all APKs with the same key pair.
- Losing a key-pair consequences that you will not be able to push updates to your app.

Step 6: Upload APK

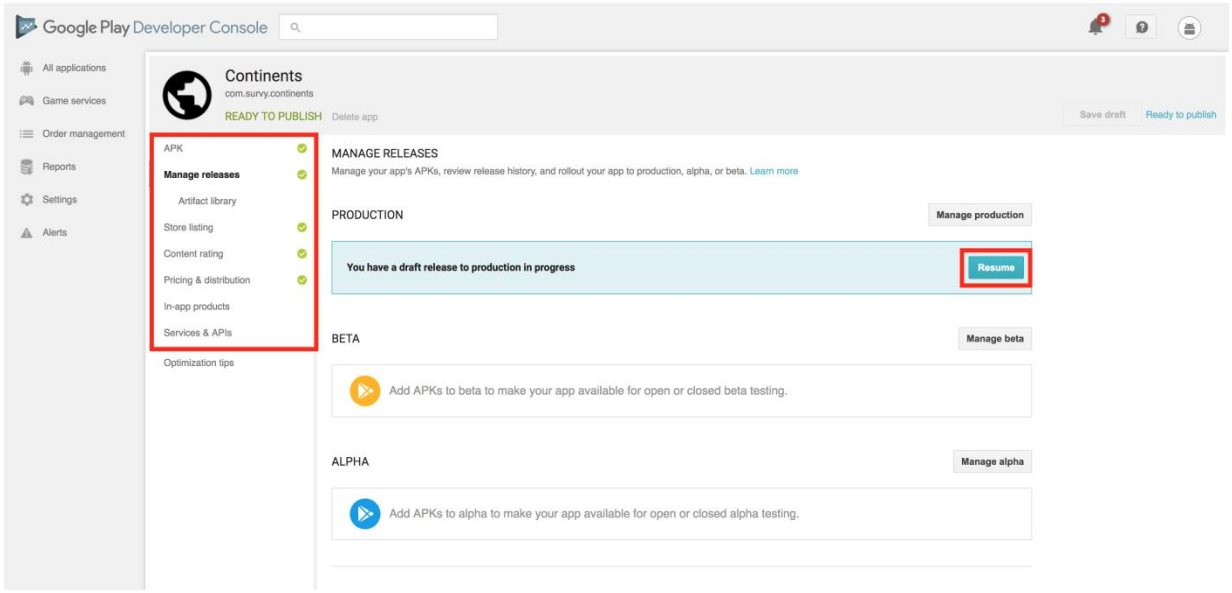
Go back to the Developer Console and click on *Manage Releases*. Then create a *Production Release* and upload your signed APK.



Google will perform a check on the APK. My app was using an SVG for the launcher icon, which is no-bueno. I had to change it to PNG and recreate the signed APK.

Step 7:

Complete the checklist on the left until all the items have a green checkmark. The console re-evaluates the checklist every time you click *Save Draft* in the top right.



You are now ready to publish

Developer Console

The Android Things Console provides easy and secure deployment of updates to your connected devices. Google provides the infrastructure to host and deliver system and app updates with the developer in final control.

Using the console, developers and device makers can:

- Download and install the latest Android Things system image
- Build factory images that contain OEM applications along with the system image
- Push over-the-air (OTA) seamless updates, including OEM applications and the system image, to devices
- Manage and share OEM applications across products and owners
- Monitor informative analytics to understand how well products are performing

Add developer account users & manage permissions

There are three different access levels on the Play Console: account owner, admins, and users. Your access type determines what actions you can take and what information you can view on the Play Console.

Account access levels

Type	Description
Account owner	• First registered the account on the Play Console • Has full access to the Play Console • Can add users, manage individual permissions, and remove user access • Is the only person who can have a linked payments profile to sell paid apps • Is the only person who can edit information on the Payments Settings page in the Play Console • Is the only person who can edit Developer Profile information in the Play Console
Admin	• Has the "Manage user permissions" permission • Can be given access to all or specific apps • Can add users, manage individual permissions, and remove user access
User	• Can have different levels of access to the Play Console • Can be given access to all or specific apps • Can't invite new users or edit user permissions • Doesn't need to pay the $25 registration fee

Give users access

Step 1: Decide whether your user needs global or per-app access

Before you set up permissions, you need to decide if your user needs global or per-app access.

- Global: Global access applies to all apps in your developer account.
- Per-app: Per-app access only applies to the selected app.

For details on how global and per-app access impacts a specific permission, select the permission below under "Permission definition & uses."

Step 2: Add users & turn permissions on or off

If you're an account owner or admin, you can add users to your Play Console account and manage permissions across all apps or for specific apps.

1. Sign in to your Play Console.
2. Click Settings > Users & permissions.
    - To add a user, select Invite new user and follow the onscreen instructions.
    - To update permissions for an existing user, hover over their email address and select the pencil icon .
    - Note: Users can only sign in to the Play Console with a Google account using the same email address that you invite.
3. Use the "Role" selector to choose a pre-defined role or use the checkboxes for individual permissions.
4. Choose whether each permission applies to all apps in your developer account ("Global") or specific apps.
    - To add an app to the permissions table, use the down arrow next to "Add an app."
    - To see details for each permission, review the permission definitions section.
5. Click Send Invitation.

Plan your MSBTE Journey with MSBTE NATION app

GET IT ON Google Play