MOBILE APPLICATION DEVELOPMENT

TYBSC(CS) 2017

COMPILED BY : SAURABH GAIKWAD (App Developer)

302 PARANJPE UDYOG BHAVAN, NEAR KHANDELWAL SWEETS, NEAR THANE STATION , THANE (WEST)

PHONE NO: 8097071144 / 8097071155

INDEX

UNIT 1

- **INTRODUCTION TO MOBILE COMPUTING**

- **Mobile**:-"able to move frelly or " or "able or willing to move freely or easily between ocuupations ,place of residence and social classes"

- **Definition for mobile computing**
- **Mobile computing** is human–computer interaction by which a computer is expected to be transported during normal usage, which allows for transmission of data, voice and video.
- In mobile computing, a set of distributed computing systems or service provider servers participate, connect, and synchronies through mobile communication protocols.
- "mobile computing" as a generic term describing ability to use the technology to wirelessly connect to and use centrally located information and/or application software through the application of small, portable, and wireless computing and communication devices

- **Mobile computing**
- Provides decentralized (distributed) computations on diversified devices, systems, and networks, which are mobile, synchronized, and interconnected via mobile communication standards and protocols.
- Mobile device does not restrict itself to just one application, such as, voice communication.
- Offers mobility with computing power.
- Facilitates a large number of applications on a single device.

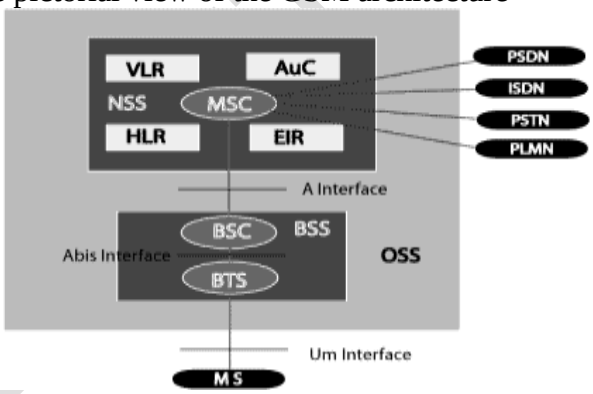
- **Technologies**
- Cellular technology is what mobile phone networks are based on, and it's the technology that gave mobile phones the name "cell phones". Cellular technology basically refers to having many small interconnected transmitters as opposed to one big one.
- The other main concept of cellular technology was that they were "multiple access", meaning that they placed multiple voice or data connections into a single radio channel.

- **GSM**

- **What is GSM?**
- CDMA (Code-Division Multiple Access) refers to any of several protocols used in second-generation (2G) and third-generation (3G) wireless communications. As the term implies, CDMA is a form of multiplexing, which allows numerous signals to occupy a single transmission channel, optimizing the use of available bandwidth. The technology is used in ultra-high-frequency (UHF) cellular telephone systems in the 800-MHz and 1.9-GHz bands.
- GSM stands for **G**lobal **S**ystem for **M**obile Communication. It is a digital cellular technology used for transmitting mobile voice and data services.
- The concept of GSM emerged from a cell-based mobile radio system at Bell Laboratories in the early 1970s.
- GSM is the name of a standardization group established in 1982 to create a common European mobile telephone standard.
- GSM is the most widely accepted standard in telecommunications and it is implemented globally.

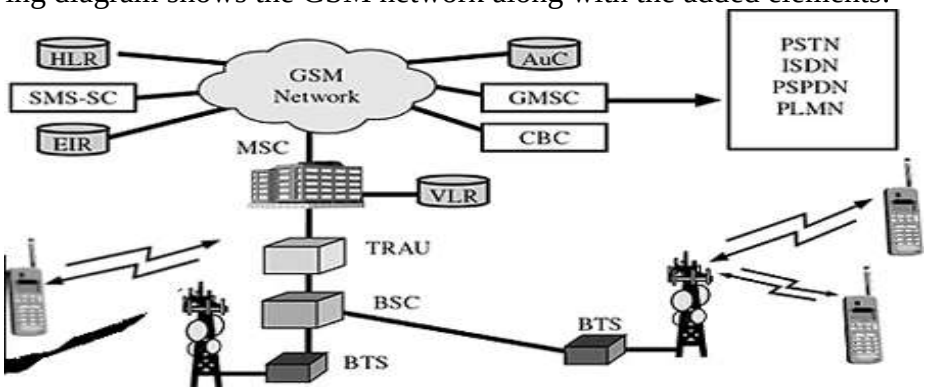
- GSM is a circuit-switched system that divides each 200 kHz channel into eight 25 kHz time-slots. GSM operates on the mobile communication bands 900 MHz and 1800 MHz in most parts of the world. In the US, GSM operates in the bands 850 MHz and 1900 MHz.
- GSM owns a market share of more than 70 percent of the world's digital cellular subscribers.
- GSM makes use of narrowband Time Division Multiple Access (TDMA) technique for transmitting signals.
- GSM was developed using digital technology. It has an ability to carry 64 kbps to 120 Mbps of data rates.
- Presently GSM supports more than one billion mobile subscribers in more than 210 countries throughout the world.
- GSM provides basic to advanced voice and data services including roaming service. Roaming is the ability to use your GSM phone number in another GSM network.
- GSM digitizes and compresses data, then sends it down through a channel with two other streams of user data, each in its own timeslot.

- **GSM Architecture**
- A GSM network comprises of many functional units. These functions and interfaces are explained in this chapter. The GSM network can be broadly divided into:
- The Mobile Station (MS)
- The Base Station Subsystem (BSS)
- The Network Switching Subsystem (NSS)
- The Operation Support Subsystem (OSS)
- Given below is a simple pictorial view of the GSM architecture



- The additional components of the GSM architecture comprise of databases and messaging systems functions:
- Home Location Register (HLR)
- Visitor Location Register (VLR)
- Base Transceiver Station (BTS)
- Equipment Identity Register (EIR)
- Authentication Center (AuC)
- SMS Serving Center (SMS SC)
- Gateway MSC (GMSC)
- Chargeback Center (CBC)
- Transcoder and Adaptation Unit (TRAU)
- A public switched data network (PSDN)
- Integrated Services Digital Network (ISDN)
- public switched telephone network (PSTN)
- public land mobile network (PLMN)

- The following diagram shows the GSM network along with the added elements:



- The MS and the BSS communicate across the Um interface. It is also known as the *air interface* or the *radio link*. The BSS communicates with the Network Service Switching (NSS) center across the *A* interface.

- **CDMA (Code Division Multiple Access)**

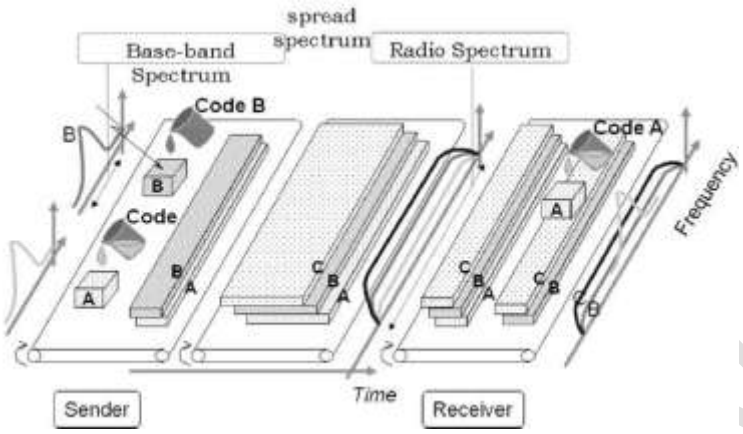
- **What is CDMA?**
- **C**ode **D**ivision **M**ultiple **A**ccess (CDMA) is a digital cellular technology used for mobile communication. Code Division Multiple Access (CDMA) is a sort of multiplexing that facilitates various signals to occupy a single transmission channel. It optimizes the use of available bandwidth. The technology is commonly used in ultra-high-frequency (UHF) cellular telephone systems, bands ranging between the 800-MHz and 1.9-GHz.

- **Salient Features of CDMA**
- CDMA, which is based on the spread spectrum technique has following salient features −
- In CDMA, every channel uses the full available spectrum.
- Individual conversations are encoded with a pseudo-random digital sequence and then transmitted using a wide frequency range.
- CDMA consistently provides better capacity for voice and data communications, allowing more subscribers to connect at any given time.
- CDMA is the common platform on which 3G technologies are built. For 3G, CDMA uses 1x EV-DO and EV-DV.

- **How Does CDMA Work?**
- CDMA allows up to 61 concurrent users in a 1.2288 MHz channel by processing each voice packet with two PN codes. There are 64 Walsh codes available to differentiate between calls and theoretical limits. Operational limits and quality issues will reduce the maximum number of calls somewhat lower than this value.

- In fact, many different "signals" baseband with different spreading codes can be modulated on the same carrier to allow many different users to be supported. Using different orthogonal codes, interference between the signals is minimal. Conversely, when signals are received from several mobile stations, the base station is capable of isolating each as they have different orthogonal spreading codes.

- The following figure shows the technicality of the CDMA system. During the propagation, we mixed the signals of all users, but by that you use the same code as the code that was used at the time of sending the receiving side. You can take out only the signal of each user.



1G

- **GSM** (**Global System for Mobile Communications**, originally ***Groupe Spécial Mobile***) is a standard developed by the European Telecommunications Standards Institute (ETSI) to describe the protocols for second-generation digital cellular networks used by mobile devices such as mobile phones, first deployed in Finland in December 1991.
- As of 2014, it has become the global standard for mobile communications – with over 90% market share, operating in over 219 countries and territories.

2G

- 2G networks developed as a replacement for first generation (1G) analog cellular networks, and the GSM standard originally described as a digital, circuit-switched network optimized for full duplex voice telephony. This expanded over time to include data communications, first by circuit-switched transport, then by packet data transport via GPRS (General Packet Radio Services) and EDGE (Enhanced Data rates for GSM Evolution, or EGPRS).
- Subsequently, the 3GPP developed third-generation (3G) UMTS standards, followed by fourth-generation (4G) LTE Advanced standards, which do not form part of the ETSI GSM standard.
- 2g was digital (rather than analog)
- 2g introduced encryption
- There were GSM and CDMA versions of 2g

3G

- stands for **third generation**, is the third generation of wireless mobile telecommunications technology. This is based on a set of standards used for mobile devices and mobile telecommunications use services and networks that comply with the International Mobile Telecommunications-2000 (IMT-2000) specifications by the International Telecommunication Union. 3G finds application in wireless voice telephony, mobile Internet access, fixed wireless Internet access, video calls and mobile TV.
- 3G telecommunication networks support services that provide an information transfer rate of at least 2 Mbit/s. Later 3G releases, often denoted 3.5G and 3.75G, also provide mobile broadband access of several Mbit/s to smartphones and mobile modems in laptop computers.

This ensures it can be applied to wireless voice telephony, mobile Internet access, fixed wireless Internet access, video calls and mobile TV technologies.

- A new generation of cellular standards has appeared approximately every tenth year since 1G systems were introduced in 1981/1982. Each generation is characterized by new frequency bands, higher data rates and non–backward-compatible transmission technology. The first 3G networks were introduced in 1998 and fourth generation 4G networks in 2008.

4G

- **4G** is the fourth generation of mobile telecommunications technology, succeeding 3G. Potential and current applications include amended mobile web access, IP telephony, gaming services, high-definition mobile TV, video conferencing, and 3D television.
- The first-release Long Term Evolution (LTE) standard (a 4G candidate system) has been commercially deployed in Oslo, Norway, and Stockholm, Sweden since 2009. It has, however, been debated whether first-release versions should be considered 4G.
- In March 2008, the International Telecommunications Union-Radio communications sector (ITU-R) specified a set of requirements for 4G standards, named the International Mobile Telecommunications Advanced (IMT-Advanced) specification, setting peak speed requirements for 4G service at 100 megabits per second (Mbit/s) for high mobility communication (such as from trains and cars) and 1 gigabit per second (Gbit/s) for low mobility communication (such as pedestrians and stationary users). Since the first-release versions of Mobile WiMAX and LTE support much less than 1 Gbit/s peak bit rate.

- **TYPES OF MOBILE COMPUTING DEVICES:**

- **Personal Digital Assistant (PDA)**
- A **personal digital assistant** (**PDA**), also known as a **handheld PC,** is a mobile device that functions as a personal information manager. PDAs were largely discontinued in the early 2010s after the widespread adoption of highly capable smartphones, in particular those based on iOS and Android.
- Nearly all PDAs have the ability to connect to the Internet. A PDA has an electronic visual display, letting it include a web browser. All models also have audio capabilities, allowing usage as a portable media player, and also enabling most of them to be used as mobile phones. Most PDAs can access the Internet, intranets or extranets via Wi-Fi or Wireless Wide Area Networks. Most PDAs employ touchscreen technology.
- The first PDA, the Organizer, was released in 1984 by Psion, followed by Psion's Series 3, in 1991. The latter began to resemble the more familiar PDA style, including a full keyboard. The term *PDA* was first used on January 7, 1992 by Apple Computer CEO John Sculley at the Consumer Electronics Show in Las Vegas, Nevada, referring to the Apple Newton. In 1994, IBM introduced the first PDA with full mobile phone functionality, the IBM Simon, which can also be considered the first smartphone. Then in 1996, Nokia introduced a PDA with full mobile phone functionality, the 9000 Communicator, which became the world's best-selling PDA. The Communicator spawned a new category of PDAs: the "PDA phone", now called "smartphone"
- Another early entrant in this market was Palm, with a line of PDA products which began in March 1996. The terms "personal digital assistant" and "PDA" apply to smartphones but are not used in marketing, media, or general conversation to refer to devices such as the BlackBerry, iPad, iPhone or Android devices.

- **SMARTPHONES**

- Mobile phone (also known as a wireless phone, cell phone, or cellular telephone) is a small portable radio telephone
- This kind of phone combines the features of a PDA with that of a mobile phone or camera phone. It has a superior edge over other kinds of mobile phones.
- Smartphones have the capability to run multiple programs concurrently. These phones include high-resolution touch screens, web browsers that can access and properly display standard web pages rather than just mobile-optimized sites, and high-speed data access via wi-fi and high speed cellular broadband.
- The most common mobile operating systems (OS) used by modern smartphones include google's android, apple's ios, nokia's symbian, rim's blackberry OS, samsung's bada, microsoft's windows phone, and embedded linux distributions such as maemo and meego. Such operating systems can be installed on different phone models, and typically each device can receive multiple OS software updates over its lifetime.

- **PAGER**

Pager (also known as a **beeper**) is a wireless telecommunications device that receives and displays alphanumeric messages and/or receives and announces voice messages. **One-way pagers** can only receive messages, while **response pagers** and **two-way pagers** can also acknowledge, reply to, and originate messages using an internal transmitter.

- Pagers operate as part of a paging system which includes one or more fixed transmitters (or in the case of response pagers and two-way pagers, one or more base stations), as well as a number of pagers carried by mobile users. These systems can range from a restaurant system with a single low-power transmitter, to a nationwide system with thousands of high-power base stations.
- Pagers were developed in the 1950s and 1960s, and became widely used by the 1980s. In the 2000s, the widespread availability of cellphones and smartphones has greatly diminished the pager industry. Nevertheless, pagers continue to be used by some emergency services and public safety personnel, because modern pager systems' coverage overlap, combined with use of satellite communications, can make paging systems more reliable than terrestrial based cellular networks in some cases, including

during natural and man-made disaster.[2] This resilience has led public safety agencies to adopt pagers over cellular and other commercial services for critical messaging.

- Benefits of pager
- Pagers are reliable
- Wide coverage
- Low cost
- Rapid response
- Helpful during critical incidents
- Group pages
- Discretion
- Integration
- Evaluation
- Mature technology

- **HISTORY OF MOBILE PLATFORM**

- **J2ME**
- Java Platform, Micro Edition or Java ME is a computing platform for development and deployment of portable code for embedded and mobile devices (micro-controllers, sensors, gateways, mobile phones, personal digital assistants, TV set-top boxes, printers).Java ME was formerly known as Java 2 Platform, Micro Edition or J2ME.
- The platform uses the object-oriented Java programming language. It is part of the Java software-platform family. Java ME was designed by Sun Microsystems, acquired by Oracle Corporation in 2010; the platform replaced a similar technology, PersonalJava. Originally developed under the Java Community Process as JSR 68, the different flavors of Java ME have evolved in separate JSRs. Sun provides a reference implementation of the specification, but has tended not to provide free binary implementations of its Java ME runtime environment for mobile devices, rather relying on third parties to provide their own.
- As of 22 December 2006, the Java ME source code is licensed under the GNU General Public License, and is released under the project name phoneME.
- As of 2008, all Java ME platforms are currently restricted to JRE 1.3 features and use that version of the class file format (internally known as version 47.0). Should Oracle ever declare a new round of Java ME configuration versions that support the later class file formats and language features, such as those corresponding to JRE 1.5 or 1.6 (notably, generics), it will entail extra work on the part of all platform vendors to update their JREs.
- Java ME devices implement a profile. The most common of these are the Mobile Information Device Profile aimed at mobile devices, such as cell phones, and the Personal Profile aimed at consumer products and embedded devices like set-top boxes and PDAs. Profiles are subsets of configurations, of which there are currently two: the Connected Limited Device Configuration (CLDC) and the Connected Device Configuration (CDC).

- **BB**
- The BlackBerry OS is a proprietary mobile operating system developed by Research In Motion for use on the company's popular BlackBerry handheld devices. The BlackBerry platform is popular with corporate users as it offers synchronization with Microsoft Exchange, Lotus Domino, Novell GroupWise email and other business software, when used with the BlackBerry Enterprise Server.

- **Android**
- Android is a mobile operating system developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. Android's user interface is mainly based on direct manipulation, using touch gestures that loosely correspond to real-world actions, such as swiping, tapping and pinching, to manipulate on-screen objects, along with a virtual keyboard for text input. In addition to touchscreen devices, Google has further developed Android TV for televisions, Android Auto for cars, and Android Wear for wrist watches, each with a specialized user interface. Variants of Android are also used on notebooks, game consoles, digital cameras, and other electronics.

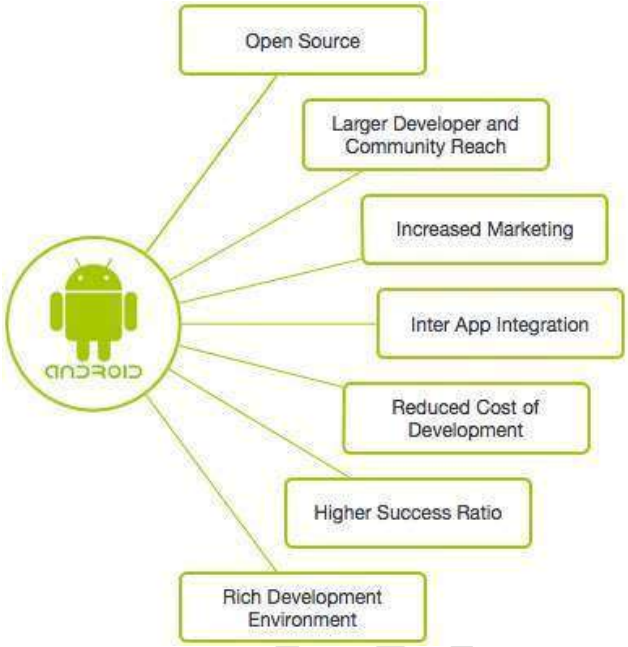
- **Windows Mobile**
- Windows Mobile is Microsoft's mobile operating system used in smartphones and mobile devices – with or without touch screens. The Mobile OS is based on the Windows CE 5.2 kernel. In 2010 Microsoft announced a new smartphone platform called Windows Phone 7.

- **THE ANDROID PLATFORM**

- **What is Android?**
- **Android** is a mobile operating system developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. Android's user interface is mainly based on direct manipulation, using touch gestures that loosely correspond to real-world actions, such as swiping, tapping and pinching, to manipulate on-screen objects, along with a virtual keyboard for text input. In addition to touchscreen devices, Google has further developed Android TV for televisions, Android Auto for cars, and Android Wear for wrist watches, each with a specialized user interface. Variants of Android are also used on game consoles, digital cameras, PCs and other electronics.
- Initially developed by Android Inc., which Google bought in 2005, Android was unveiled in 2007, along with the founding of the Open Handset Alliance – a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. Beginning with the first commercial Android device in September 2008, the operating system has gone through multiple major releases, with the current version being 8.0 "Oreo", released in August 2017. Android applications ("apps") can be downloaded from the Google Play store, which features over 2.7 million apps as of February 2017. Android has been the best-selling OS on tablets since 2013, and runs on the vast majority[a] of smartphones. As of May 2017, Android has two billion monthly active users, and it has the largest installed base of any operating system.
- Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.
- The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.
- On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 Jelly Bean. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance.

- The source code for Android is available under free and open source software licenses. Google publishes most of the code under the Apache License version 2.0 and the rest, Linux kernel changes, under the GNU General Public License version 2.

- **Why Android?**

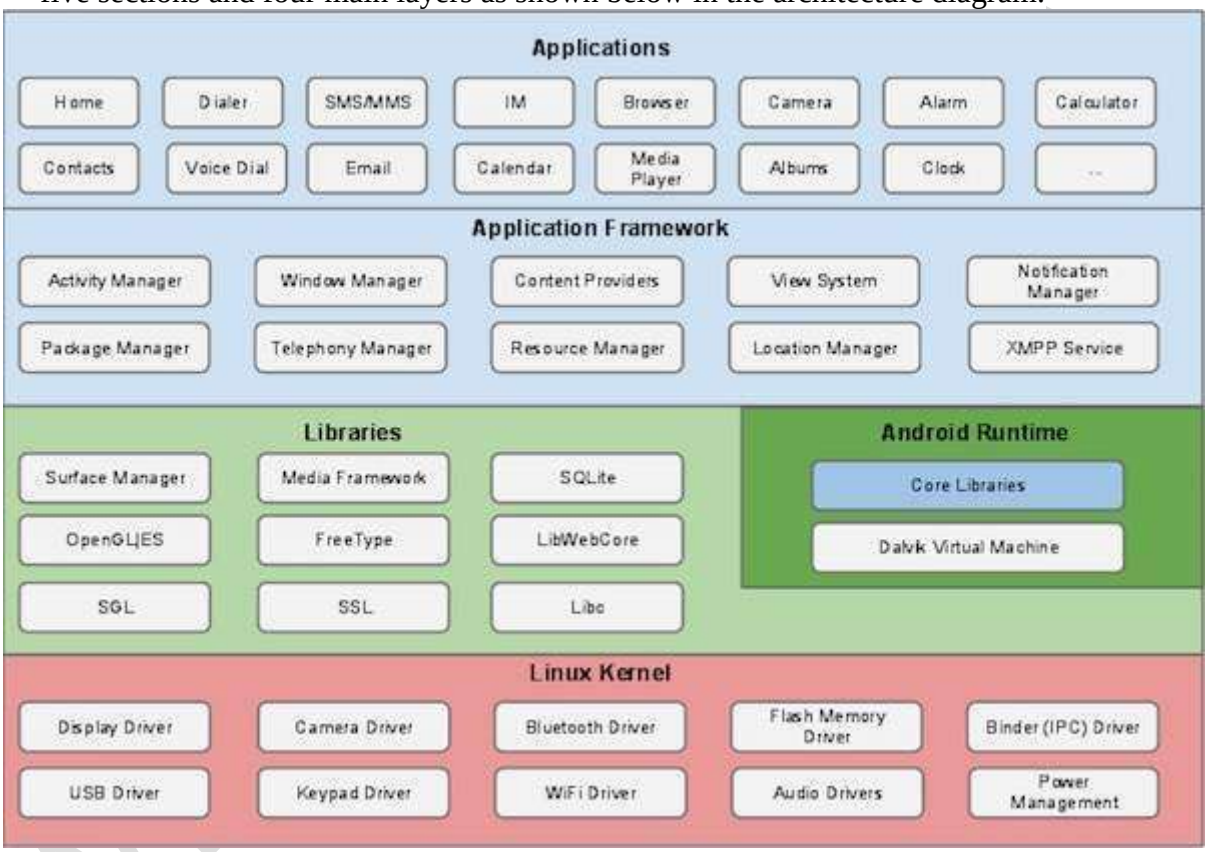


- **Features of Android**
- Android is a powerful operating system competing with Apple 4GS and supports great features. Few of them are listed below −

Sr.No.	Feature & Description
1	Beautiful UI Android OS basic screen provides a beautiful and intuitive user interface.
2	Connectivity GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.
3	Storage SQLite, a lightweight relational database, is used for data storage purposes.
4	Media support H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP.
5	Messaging SMS and MMS
6	Web browser Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.
7	Multi-touch Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.
8	Multi-tasking User can jump from one task to another and same time various application can run simultaneously.
9	Resizable widgets Widgets are resizable, so users can expand them to show more content or shrink them to save space.
10	Multi-Language

	Supports single direction and bi-directional text.
11	GCM Google Cloud Messaging (GCM) is a service that lets developers send short message data to their users on Android devices, without needing a proprietary sync solution.
12	Wi-Fi Direct A technology that lets apps discover and pair directly, over a high-bandwidth peer-to-peer connection.
13	Android Beam A popular NFC-based technology that lets users instantly share, just by touching two NFC-enabled phones together.

- **ANDROID ARCHITECTURE**
- Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.



- **Linux kernel**
- At the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

- **Libraries**
- On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage

and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

- **Android Libraries**
- This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access. A summary of some key core Android libraries available to the Android developer is as follows −
- android.app − Provides access to the application model and is the cornerstone of all Android applications.
- android.content − Facilitates content access, publishing and messaging between applications and application components.
- android.database − Used to access data published by content providers and includes SQLite database management classes.
- android.opengl − A Java interface to the OpenGL ES 3D graphics rendering API.
- android.os − Provides applications with access to standard operating system services including messages, system services and inter-process communication.
- android.text − Used to render and manipulate text on a device display.
- android.view − The fundamental building blocks of application user interfaces.
- android.widget − A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
- android.webkit − A set of classes intended to allow web-browsing capabilities to be built into applications.
- Having covered the Java-based core libraries in the Android runtime, it is now time to turn our attention to the C/C++ based libraries contained in this layer of the Android software stack.

- **Android Runtime**
- This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called Dalvik Virtual Machine which is a kind of Java Virtual Machine specially designed and optimized for Android.
- The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.
- The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

- **Application Framework**
- The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.
- The Android framework includes the following key services −
- Activity Manager − Controls all aspects of the application lifecycle and activity stack.
- Content Providers − Allows applications to publish and share data with other applications.
- Resource Manager − Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
- Notifications Manager − Allows applications to display alerts and notifications to the user.
- View System − An extensible set of views used to create application user interfaces.

- **Applications**
- You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games etc.
- Android application component

- **ANDROID APPLICATION COMPONENTS**
- Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file *AndroidManifest.xml* that describes each component of the application and how they interact.
- There are following four main components that can be used within an Android application –

Sr.No	Components & Description
1	Activities They dictate the UI and handle the user interaction to the smart phone screen.
2	Services They handle background processing associated with an application.
3	Broadcast Receivers They handle communication between Android OS and applications.
4	Content Providers They handle data and database management issues.

- **Activities**
- An activity represents a single screen with a user interface, in-short Activity performs actions on the screen. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.
- An activity is implemented as a subclass of Activity class as follows −

```
public class MainActivity extends Activity {
}
```

- **Services**
- A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.
- A service is implemented as a subclass of Service class as follows −

```
public class MyService extends Service {
}
```

- **Broadcast Receivers**
- Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

- A broadcast receiver is implemented as a subclass of BroadcastReceiver class and each message is broadcaster as an Intent object.

```
public class MyReceiver  extends  BroadcastReceiver {
public void onReceive(context,intent){}
}
```

- **Content Providers**
  - A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the *ContentResolver* class. The data may be stored in the file system, the database or somewhere else entirely.
  - A content provider is implemented as a subclass of ContentProvider class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider extends  ContentProvider {
public void onCreate(){}
}
```

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

S.No	Components & Description
1	**Fragments** Represents a portion of user interface in an Activity.
2	**Views** UI elements that are drawn on-screen including buttons, lists forms etc.
3	**Layouts** View hierarchies that control screen format and appearance of the views.
4	**Intents** Messages wiring components together.
5	**Resources** External elements, such as strings, constants and drawable pictures.
6	**Manifest** Configuration files for the application.

- **DEVELOPMENT TOOLS**
- **ANDROID SDK FEATURES**
  - The true appeal of Android as a development environment lies in its APIs. As an application-neutral platform, Android gives you the opportunity to create applications that are as much a part of the phone as anything provided out-of-the-box. The following list highlights some of the most noteworthy Android features:
  - GSM, EDGE, 3G, 4G, and LTE networks for telephony or data transfer, enabling you to make or receive calls or SMS messages, or to send and retrieve data across mobile networks
  - Comprehensive APIs for location-based services such as GPS and network-based location detection
  - Full support for applications that integrate map controls as part of their user interfaces
  - Wi-Fi hardware access and peer-to-peer connections
  - Full multimedia hardware control, including playback and recording with the camera and microphone
  - Media libraries for playing and recording a variety of audio/video or still-image formats

- APIs for using sensor hardware, including accelerometers, compasses, and barometers
- Libraries for using Bluetooth and NFC hardware for peer-to-peer data transfer
- IPC message passing
- Shared data stores and APIs for contacts, social networking, calendar, and multi-media
- Background Services, applications, and processes
- Home-screen Widgets and Live Wallpaper
- The ability to integrate application search results into the system searches
- An integrated open-source HTML5 WebKit-based browser
- Mobile-optimized, hardware-accelerated graphics, including a path-based 2D graphics library and support for 3D graphics using OpenGL ES 2.0
- Localization through a dynamic resource framework
- An application framework that encourages the reuse of application components and the
- replacement of native applications

- **ANDROID DEBUG BRIDGE (ADB)**
- The Android Debug Bridge (ADB) is a client-service application that lets you connect with an Android device (virtual or actual). It's made up of three components:
- A daemon running on the device or Emulator
- A service that runs on your development computer
- Client applications (such as the DDMS) that communicate with the daemon through the service
- As a communications conduit between your development hardware and the Android device/ Emulator, the ADB lets you install applications, push and pull files, and run shell commands on the target device. Using the device shell, you can change logging settings and query or modify SQLite databases available on the device.
- The ADT tool automates and simplifies a lot of the usual interaction with the ADB, including application installation and updating, file logging, and file transfer

- **GRADLE**
- **Features of Gradle**
- Following is the list of features that Gradle provides.
- **Declarative builds and build-by-convention** − Gradle is available with separate Domain Specific Language (DSL) based on Groovy language. Gradle provides declarative language elements. The elements also provide build-by-convention support for Java, Groovy, OSGi, Web and Scala.
- **Language for dependency based programming** − The declarative language lies on top of a general purpose task graph, which you can fully leverage in your build.
- **Structure your build** − Gradle allows you to apply common design principles to your build. It gives you a perfect structure for build, so that you can design well-structured and easily maintained, comprehensible build.
- **Deep API** − Using this API, you can monitor and customize its configuration and execution behavior to its core.
- **Gradle scales** − Gradle can easily increase productivity, from simple and single project builds to huge enterprise multi-project builds.
- **Multi-project builds** − Gradle supports multi-project builds and also partial builds. If you build a subproject, Gradle takes care of building all the subprojects that it depends on.
- **Different ways to manage your builds** − Gradle supports different strategies to manage your dependencies.

- **First build integration tool** − Gradle completely supports ANT tasks, Maven and lvy repository infrastructure for publishing and retrieving dependencies. It also provides a converter for turning a Maven pom.xml to Gradle script.
- **Ease of migration** − Gradle can easily adapt to any structure you have. Therefore, you can always develop your Gradle build in the same branch where you can build
- **Gradle Wrapper** − Gradle Wrapper allows you to execute Gradle builds on machines where Gradle is not installed. This is useful for continuous integration of servers.
- **Free open source** − Gradle is an open source project, and licensed under the Apache Software License (ASL).
- **Groovy** − Gradle's build script is written in Groovy. The whole design of Gradle is oriented towards being used as a language, not as a rigid framework. Groovy allows you to write your own script with some abstractions. The entire Gradle API is designed in Groovy language.

- **ANDROID ENVIRONMENT SETUP**
- We can start your Android application development on either of the following operating systems −
- Microsoft Windows XP or later version.
- Mac OS X 10.5.8 or later version with Intel chip.
- Linux including GNU C Library 2.7 or later.
- Second point is that all the required tools to develop Android applications are freely available and can be downloaded from the Web. Following is the list of software's you will need before you start your Android application programming.
- Java JDK5 or later version
- Android Studio
- Here last two components are optional and if you are working on Windows machine then these components make your life easy while doing Java based application development. So let us have a look how to proceed to set required environment.

- **Set-up Java Development Kit (JDK)**
- You can download the latest version of Java JDK from Oracle's Java site − Java SE Downloads. You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup. Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains java and javac, typically java_install_dir/bin and java_install_dir respectively.
- If you are running Windows and installed the JDK in C:\jdk1.8.0_102, you would have to put the following line in your C:\autoexec.bat file.

```
set PATH=C:\jdk1.8.0_102\bin;%PATH%
set JAVA_HOME=C:\jdk1.8.0_102
```

- Alternatively, you could also right-click on *My Computer*, select *Properties*, then *Advanced*, then *Environment Variables*. Then, you would update the PATH value and press the OK button.
- On Linux, if the SDK is installed in /usr/local/jdk1.8.0_102 and you use the C shell, you would put the following code into your .cshrc file.

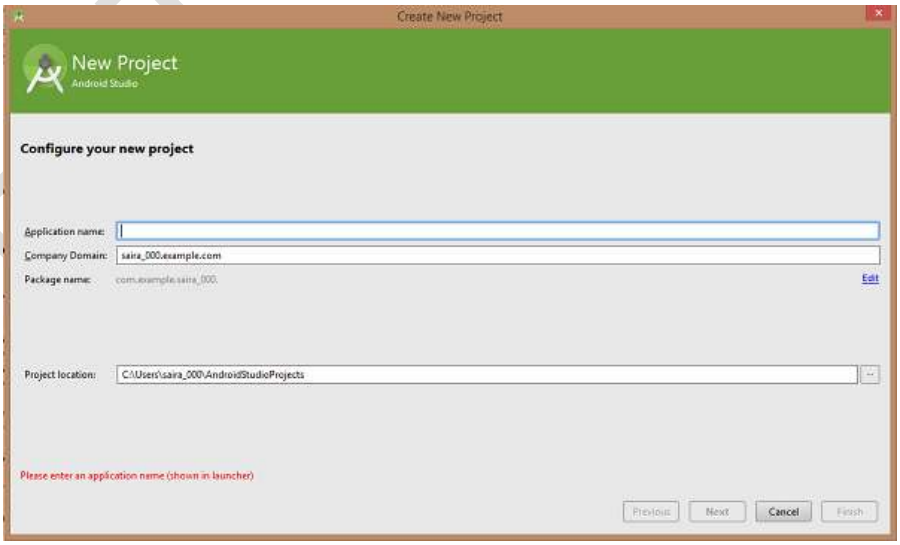
```
setenv PATH /usr/local/jdk1.8.0_102/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.8.0_102
```

- Alternatively, if you use Android studio, then it will know automatically where you have installed your Java.

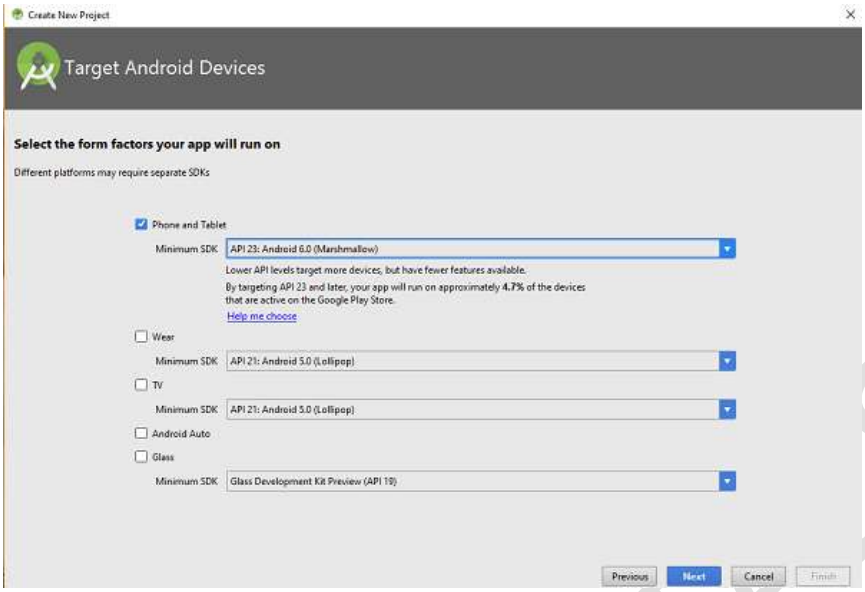
- **Android IDEs**
- There are so many sophisticated Technologies are available to develop android applications, the familiar technologies, which are predominantly using tools as follows
- Android Studio
- Eclipse IDE(Deprecated)

- **CREATE ANDROID APPLICATION**
- The first step is to create a simple Android Application using Android studio. When you click on Android studio icon, it will show screen as shown below

- You can start your application development by calling start a new android studio project in a new installation frame should ask Application name, package information and location of the project.−

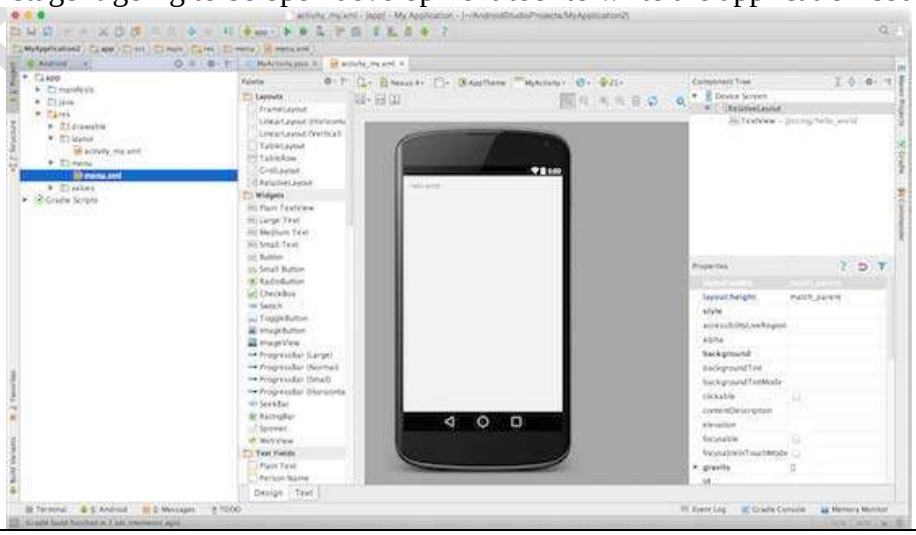


- After entered application name, it going to be called select the form factors your application runs on, here need to specify Minimum SDK, in our tutorial, I have declared as API23: Android 6.0(Mashmallow) –

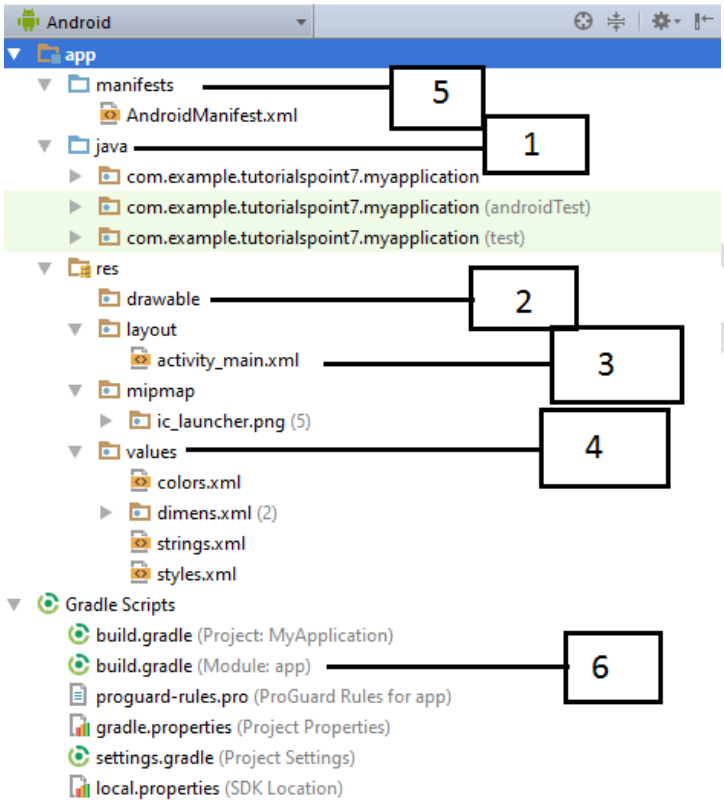


- The next level of installation should contain selecting the activity to mobile, it specifies the default layout for Applications.

- At the final stage it going to be open development tool to write the application code.



- **ANATOMY OF ANDROID APPLICATION**
- Before you run your app, you should be aware of a few directories and files in the Android project −



Sr.No.	Folder, File & Description
1	**Java** This contains the .java source files for your project. By default, it includes an *MainActivity.java* source file having an activity class that runs when your app is launched using the app icon.
2	**res/drawable-hdpi** This is a directory for drawable objects that are designed for high-density screens.
3	**res/layout** This is a directory for files that define your app's user interface.
4	**res/values** This is a directory for other various XML files that contain a collection of resources, such as strings and colours definitions.
5	**AndroidManifest.xml** This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.
6	**Build.gradle** This is an auto generated file which contains compileSdkVersion, buildToolsVersion, applicationId, minSdkVersion, targetSdkVersion, versionCode and versionName

-
-

- **The Main Activity File**
- The main activity code is a Java file MainActivity.java. This is the actual application file which ultimately gets converted to a Dalvik executable and runs your application. Following is the default code generated by the application wizard for *Hello World!* application −

```
package com.example.helloworld;
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);
}
}
```

- Here, *R.layout.activity_main* refers to the *activity_main.xml* file located in the *res/layout* folder. The *onCreate()* method is one of many methods that are figured when an activity is loaded.

- **The Manifest File**
- Whatever component you develop as a part of your application, you must declare all its components in a *manifest.xml* which resides at the root of the application project directory. This file works as an interface between Android OS and your application, so if you do not declare your component in this file, then it will not be considered by the OS. For example, a default manifest file will look like as following file −

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

- Here <application>...</application> tags enclosed the components related to the application. Attribute *android:icon* will point to the application icon available under *res/drawable-hdpi*. The application uses the image named ic_launcher.png located in the drawable folders

- The <activity> tag is used to specify an activity and *android:name* attribute specifies the fully qualified class name of the *Activity* subclass and the *android:label* attributes specifies a string to use as the label for the activity. You can specify multiple activities using <activity> tags.
- The action for the intent filter is named *android.intent.action.MAIN* to indicate that this activity serves as the entry point for the application. The category for the intent-filter is named *android.intent.category.LAUNCHER* to indicate that the application can be launched from the device's launcher icon.
- The *@string* refers to the *strings.xml* file explained below. Hence, *@string/app_name* refers to the *app_name* string defined in the strings.xml file, which is "HelloWorld". Similar way, other strings get populated in the application.
- Following is the list of tags which you will use in your manifest file to specify different Android application components −
- <activity>elements for activities
- <service> elements for services
- <receiver> elements for broadcast receivers
- <provider> elements for content providers

- **The Strings File**
- The strings.xml file is located in the *res/values* folder and it contains all the text that your application uses. For example, the names of buttons, labels, default text, and similar types of strings go into this file. This file is responsible for their textual content. For example, a default strings file will look like as following file −

```
<resources>
<string name="app_name">HelloWorld</string>
<string name="hello_world">Hello world!</string>
<string name="menu_settings">Settings</string>
<string name="title_activity_main">MainActivity</string>
</resources>
```

- **The Layout File**
- The activity_main.xml is a layout file available in *res/layout* directory that is referenced by your application when building its interface. You will modify this file very frequently to change the layout of your application. For your "Hello World!" application, this file will have following content related to default 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" >

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:layout_centerVertical="true"
android:padding="@dimen/padding_medium"
android:text="@string/hello_world"
```

```
tools:context=".MainActivity" />

</RelativeLayout>
```

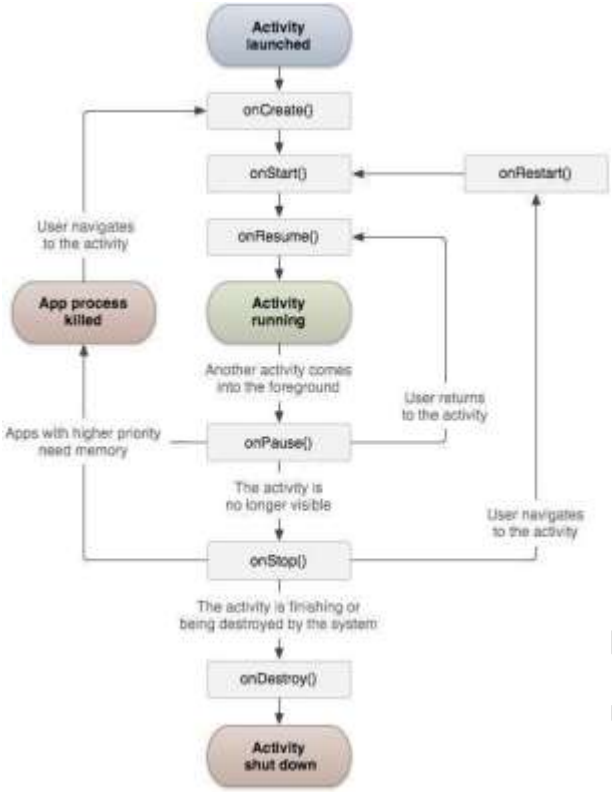
- The *TextView* is an Android control used to build the GUI and it have various attributes like *android:layout_width*, *android:layout_height* etc which are being used to set its width and height etc.. The *@string* refers to the strings.xml file located in the res/values folder. Hence, @string/hello_world refers to the hello string defined in the strings.xml file, which is "Hello World!".

- **Running the Application**
- Let's try to run our Hello World! application we just created. I assume you had created your AVD while doing environment set-up. To run the app from Android studio, open one of your project's activity files and click Run icon from the tool bar. Android studio installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display following Emulator window −

- **ANDROID ACTIVITY**
- Android system initiates its program within an Activity starting with a call on *onCreate()* callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity life cycle diagram:

- The Activity class defines the following call backs i.e. events. You don't need to implement all the callbacks methods.

Sr.No	Callback & Description
1	onCreate() This is the first callback and called when the activity is first created.
2	onStart() This callback is called when the activity becomes visible to the user.
3	onResume() This is called when the user starts interacting with the application.
4	onPause() The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
5	onStop() This callback is called when the activity is no longer visible.
6	onDestroy() This callback is called before the activity is destroyed by the system.
7	onRestart() This callback is called when the activity restarts after stopping it.



- **Example**
- This example will take through simple steps to show Android application activity life cycle. Follow the following steps to modify the Android application we created in *Hello World Example* −

Step	Description
1	You will use Android studio to create an Android application and name it as *HelloWorld* under a package *com.example.helloworld* as explained in the *Hello World Example*.
2	Modify main activity file *MainActivity.java* as explained below. Keep rest of the files unchanged.
3	Run the application to launch Android emulator and verify the result of the changes done in the application.

- Following is the content of the modified main activity file src/com.example.helloworld/MainActivity.java. This file includes each of the fundamental life cycle methods. The Log.d() method has been used to generate log messages –

```
package com.example.helloworld;
import android.os.Bundle;
import android.app.Activity;
import android.util.Log;

public class MainActivity extends Activity {
String msg = "Android : ";

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
```

```
Log.d(msg, "The onCreate() event");
}

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

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

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

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

- An activity class loads all UI components using the XML file available in *res/layout* folder of the project. Following statement loads UI components from *res/layout/activity_main.xml file*:

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

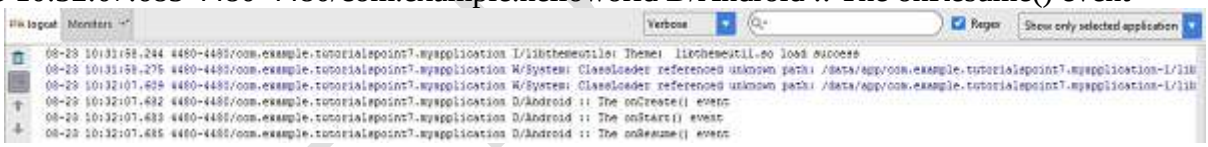
- An application can have one or more activities without any restrictions. Every activity you define for your application must be declared in your *AndroidManifest.xml* file and the main activity for your app must be declared in the manifest with an <intent-filter> that includes the MAIN action and LAUNCHER category as follows:

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

  <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>
</application>

</manifest>
```

- If either the MAIN action or LAUNCHER category is not declared for one of our activities, then our app icon will not appear in the Home screen's list of apps.
- Let's try to run our modified Hello World! application we just modified. I assume you had created your AVD while doing environment setup. To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display Emulator window and you should see following log messages in LogCat window in Android studio −

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

- 
- Let us try to click lock screen button on the Android emulator and it will generate following events messages in LogCat window in android studio:

08-23 10:32:53.230 4480-4480/com.example.helloworld D/Android :: The onPause() event
08-23 10:32:53.294 4480-4480/com.example.helloworld D/Android :: The onStop() event

- Let us again try to unlock your screen on the Android emulator and it will generate following events messages in LogCat window in Android studio:

08-23 10:34:41.390 4480-4480/com.example.helloworld D/Android :: The onStart() event
08-23 10:34:41.392 4480-4480/com.example.helloworld D/Android :: The onResume() event

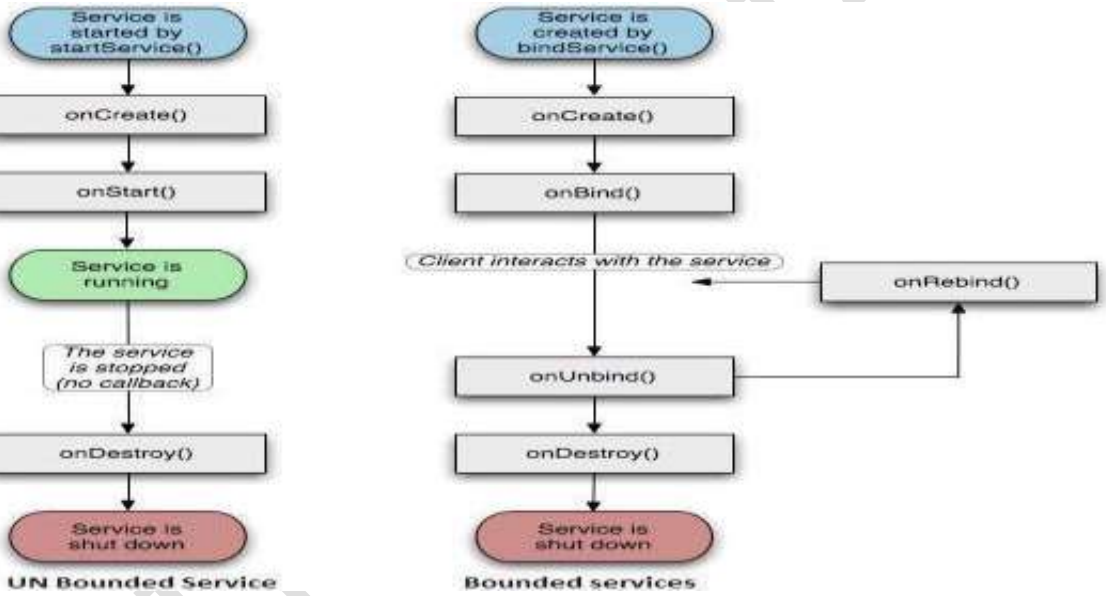
- Next, let us again try to click Back button on the Android emulator and it will generate following events messages in LogCat window in Android studio and this completes the Activity Life Cycle for an Android Application.

08-23 10:37:24.806 4480-4480/com.example.helloworld D/Android :: The onPause() event
08-23 10:37:25.668 4480-4480/com.example.helloworld D/Android :: The onStop() event
08-23 10:37:25.669 4480-4480/com.example.helloworld D/Android :: The onDestroy() event

- **SERVICE**
  - A service is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed. A service can essentially take two states −

Sr.No.	State & Description
1	Started A service is started when an application component, such as an activity, starts it by calling *startService()*. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
2	Bound A service is bound when an application component binds to it by calling *bindService()*. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

- A service has life cycle callback methods that you can implement to monitor changes in the service's state and you can perform work at the appropriate stage. The following diagram on the left shows the life cycle when the service is created with startService() and the diagram on the right shows the life cycle when the service is created with bindService(): *(image courtesy : android.com)*



- To create a service, you create a Java class that extends the Service base class or one of its existing subclasses. The Service base class defines various callback methods and the most important are given below.

Sr.No.	Callback & Description
1	onStartCommand() The system calls this method when another component, such as an activity, requests that the service be started, by calling *startService()*. If you implement this method, it is your responsibility to stop the service when its work is done, by calling *stopSelf()* or *stopService()* methods.
2	onBind() The system calls this method when another component wants to bind with the service by calling *bindService()*. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an *IBinder* object. You must always implement this method, but if you don't want to allow binding, then you should return *null*.
3	onUnbind() The system calls this method when all clients have disconnected from a particular

	interface published by the service.
4	onRebind() The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its *onUnbind(Intent)*.
5	onCreate() The system calls this method when the service is first created using *onStartCommand()* or *onBind()*. This call is required to perform one-time set-up.
6	onDestroy() The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

- The following skeleton service demonstrates each of the life cycle methods −

```
package com.tutorial;
import android.app.Service;
import android.os.IBinder;
import android.content.Intent;
import android.os.Bundle;
public class HelloService extends Service {

/** indicates how to behave if the service is killed */
int mStartMode;

/** interface for clients that bind */
IBinder mBinder;

/** indicates whether onRebind should be used */
boolean mAllowRebind;

/** Called when the service is being created. */
@Override
public void onCreate() {

}

/** The service is starting, due to a call to startService() */
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
return mStartMode;
}

/** A client is binding to the service with bindService() */
@Override
public IBinder onBind(Intent intent) {
return mBinder;
}

/** Called when all clients have unbound with unbindService() */
@Override
public boolean onUnbind(Intent intent) {
return mAllowRebind;
}
```

```
/** Called when a client is binding to the service with bindService()*/
@Override
public void onRebind(Intent intent) {

}

/** Called when The service is no longer used and is being destroyed */
@Override
public void onDestroy() {

}
}
```

- Example
- This example will take us through simple steps to show how to create your own Android Service. Follow the following steps to modify the Androidapplication we created in *Hello World Example*

Step	Description
1	You will use Android StudioIDE to create an Android application and name it as *My Application* under a package *com.example.demo.myapplication* as explained in the *Hello World Example* .
2	Modify main activity file *MainActivity.java* to add *startService()* and *stopService()* methods.
3	Create a new java file *MyService.java* under the package *com.example.My Application*. This file will have implementation of Android service related methods.
4	Define your service in *AndroidManifest.xml* file using <service.../> tag. An application can have one or more services without any restrictions.
5	Modify the default content of *res/layout/activity_main.xml* file to include two buttons in linear layout.
6	No need to change any constants in *res/values/strings.xml* file. Androidstudio take care of string values
7	Run the application to launch Android emulator and verify the result of the changes done in the application.

- Following is the content of the modified main activity file MainActivity.java. This file can include each of the fundamental life cycle methods. We have added *startService()* and *stopService()* methods to start and stop the service.

```
package com.example.Demo.myapplication;

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

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;
import android.view.View;

public class MainActivity extends Activity {
String msg = "Android : ";

/** Called when the activity is first created. */
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
Log.d(msg, "The onCreate() event");
}

public void startService(View view) {
startService(new Intent(getBaseContext(), MyService.class));
}

// Method to stop the service
public void stopService(View view) {
stopService(new Intent(getBaseContext(), MyService.class));
}
}
```

Following is the content of MyService.java. This file can have implementation of one or more methods associated with Service based on requirements. For now we are going to implement only two methods *onStartCommand()* and *onDestroy()* −

```
package com.example.Demo.myapplication;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.support.annotation.Nullable;
import android.widget.Toast;
public class MyService extends Service {
@Nullable
@Override
public IBinder onBind(Intent intent) {
return null;
}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
// Let it continue running until it is stopped.
Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
return START_STICKY;
}

@Override
public void onDestroy() {
super.onDestroy();
Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();
}
}
```

Following will the modified content of *AndroidManifest.xml* file. Here we have added <service.../> tag to include our service −

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

```
package="com.example.Demo.myapplication">

<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>

<service android:name=".MyService" />
</application>

</manifest>
```

Following will be the content of res/layout/activity_main.xml file to include two buttons −

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

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Example of services"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="30dp" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials point "
    android:textColor="#ff87ff09"
    android:textSize="30dp"
    android:layout_above="@+id/imageButton"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="40dp" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
```

```
        android:src="@drawable/abc"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button2"
        android:text="Start Services"
        android:onClick="startService"
        android:layout_below="@+id/imageButton"
        android:layout_centerHorizontal="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Stop Services"
        android:id="@+id/button"
        android:onClick="stopService"
        android:layout_below="@+id/button2"
        android:layout_alignLeft="@+id/button2"
        android:layout_alignStart="@+id/button2"
        android:layout_alignRight="@+id/button2"
        android:layout_alignEnd="@+id/button2" />
    </RelativeLayout>
```

- Let's try to run our modified Hello World! application we just modified. I assume you had created your AVD while doing environment setup. To run the app from Android studio, open one of your project's activity files and click Run icon from the tool bar. Android Studio installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display following Emulator window −
-

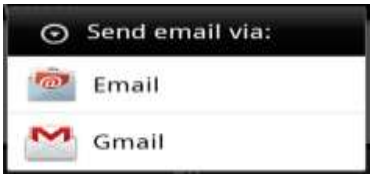
- Now to start your service, let's click on Start Service button, this will start the service and as per our programming in *onStartCommand()* method, a message *Service Started* will appear on the bottom of the simulator as follows –

- To stop the service, you can click the Stop Service button.

- **INTENT-**
- An Android **Intent** is an abstract description of an operation to be performed. It can be used with **startActivity** to launch an Activity, **broadcastIntent** to send it to any interested BroadcastReceiver components, and **startService(Intent)** or **bindService(Intent, ServiceConnection, int)** to communicate with a background Service.
- **The intent itself, an Intent object, is a passive data structure holding an abstract description of an operation to be performed.**
- For example, let's assume that you have an Activity that needs to launch an email client and sends an email using your Android device. For this purpose, your Activity would send an ACTION_SEND along with appropriate **chooser**, to the Android Intent Resolver. The specified chooser gives the proper interface for the user to pick how to send your email data.

```
Intent email = new Intent(Intent.ACTION_SEND, Uri.parse("mailto:"));
email.putExtra(Intent.EXTRA_EMAIL, recipients);
email.putExtra(Intent.EXTRA_SUBJECT, subject.getText().toString());
email.putExtra(Intent.EXTRA_TEXT, body.getText().toString());
startActivity(Intent.createChooser(email, "Choose an email client from..."));
```

Above syntax is calling startActivity method to start an email activity and result should be as shown below −



- For example, assume that you have an Activity that needs to open URL in a web browser on your Android device. For this purpose, your Activity will send ACTION_WEB_SEARCH Intent to the Android Intent Resolver to open given URL in the web browser. The Intent Resolver parses through a list of Activities and chooses the one that would best match your Intent, in this case, the Web Browser Activity. The Intent Resolver then passes your web page to the web browser and starts the Web Browser Activity.

```
String q = "tutorial";
Intent intent = new Intent(Intent.ACTION_WEB_SEARCH );
intent.putExtra(SearchManager.QUERY, q);
startActivity(intent);
```

- There are separate mechanisms for delivering intents to each type of component − activities, services, and broadcast receivers.

Sr.No	Method & Description
1	**Context.startActivity()** The Intent object is passed to this method to launch a new activity or get an existing activity to do something new.
2	**Context.startService()** The Intent object is passed to this method to initiate a service or deliver new instructions to an ongoing service.
3	**Context.sendBroadcast()** The Intent object is passed to this method to deliver the message to all interested broadcast receivers.

- **INTENT OBJECTS**
- An Intent object is a bundle of information which is used by the component that receives the intent as well as information used by the Android system.
- An Intent object can contain the following components based on what it is communicating or going to perform −

- **Action**
- This is mandatory part of the Intent object and is a string naming the action to be performed — or, in the case of broadcast intents, the action that took place and is being reported. The action largely determines how the rest of the intent object is structured. The Intent class defines a number of action constants corresponding to different intents. Here is a list of Android Intent Standard Actions
- The action in an Intent object can be set by the setAction() method and read by getAction().

- **Data**
- Adds a data specification to an intent filter. The specification can be just a data type (the mimeType attribute), just a URI, or both a data type and a URI. A URI is specified by separate attributes for each of its parts −
- These attributes that specify the URL format are optional, but also mutually dependent −
- If a scheme is not specified for the intent filter, all the other URI attributes are ignored.
- If a host is not specified for the filter, the port attribute and all the path attributes are ignored.
- The setData() method specifies data only as a URI, setType() specifies it only as a MIME type, and setDataAndType() specifies it as both a URI and a MIME type. The URI is read by getData() and the type by getType().
- Some examples of action/data pairs are −

Sr.No.	Action/Data Pair & Description
1	**ACTION_VIEW content://contacts/people/1** Display information about the person whose identifier is "1".
2	**ACTION_DIAL content://contacts/people/1** Display the phone dialer with the person filled in.
3	**ACTION_VIEW tel:123** Display the phone dialer with the given number filled in.

4	**ACTION_DIAL tel:123** Display the phone dialer with the given number filled in.
5	**ACTION_EDIT content://contacts/people/1** Edit information about the person whose identifier is "1".
6	**ACTION_VIEW content://contacts/people/** Display a list of people, which the user can browse through.
7	**ACTION_SET_WALLPAPER** Show settings for choosing wallpaper
8	**ACTION_SYNC** It going to be synchronous the data,Constant Value is **android.intent.action.SYNC**
9	**ACTION_SYSTEM_TUTORIAL** It will start the platform-defined tutorial(Default tutorial or start up tutorial)
10	**ACTION_TIMEZONE_CHANGED** It intimates when time zone has changed
11	**ACTION_UNINSTALL_PACKAGE** It is used to run default uninstaller

- **Category**
- The category is an optional part of Intent object and it's a string containing additional information about the kind of component that should handle the intent. The addCategory() method places a category in an Intent object, removeCategory() deletes a category previously added, and getCategories() gets the set of all categories currently in the object. Here is a list of Android Intent Standard Categories.
- You can check detail on Intent Filters in below section to understand how do we use categories to choose appropriate activity corresponding to an Intent.

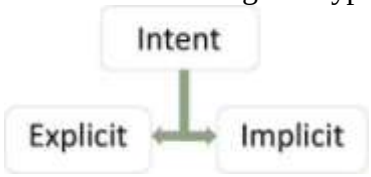
- **Extras**
- This will be in key-value pairs for additional information that should be delivered to the component handling the intent. The extras can be set and read using the putExtras() and getExtras() methods respectively. Here is a list of Android Intent Standard Extra Data

- **Flags**
- These flags are optional part of Intent object and instruct the Android system how to launch an activity, and how to treat it after it's launched etc.

Sr.No	Flags & Description
1	**FLAG_ACTIVITY_CLEAR_TASK** If set in an Intent passed to Context.startActivity(), this flag will cause any existing task that would be associated with the activity to be cleared before the activity is started. That is, the activity becomes the new root of an otherwise empty task, and any old activities are finished. This can only be used in conjunction with FLAG_ACTIVITY_NEW_TASK.
2	**FLAG_ACTIVITY_CLEAR_TOP** If set, and the activity being launched is already running in the current task, then instead of launching a new instance of that activity, all of the other activities on top of it will be closed and this Intent will be delivered to the (now on top) old activity as a new Intent.
3	**FLAG_ACTIVITY_NEW_TASK** This flag is generally used by activities that want to present a "launcher" style behavior: they give the user a list of separate things that can be done, which otherwise run completely independently of the activity launching them.

- **Component Name**
- This optional field is an android **ComponentName** object representing either Activity, Service or BroadcastReceiver class. If it is set, the Intent object is delivered to an instance of the designated class otherwise Android uses other information in the Intent object to locate a suitable target.
- The component name is set by setComponent(), setClass(), or setClassName() and read by getComponent().

- **TYPES OF INTENTS**
- There are following two types of intents supported by Android



- **Explicit Intents**
- Explicit intent going to be connected internal world of application,suppose if you wants to connect one activity to another activity, we can do this quote by explicit intent, below image is connecting first activity to second activity by clicking button.

- These intents designate the target component by its name and they are typically used for application-internal messages - such as an activity starting a subordinate service or launching a sister activity. For example −

```
// Explicit Intent by specifying its class name
Intent i = new Intent(FirstActivity.this, SecondActivity.class);

// Starts TargetActivity
startActivity(i);
```

- **Implicit Intents**
- These intents do not name a target and the field for the component name is left blank. Implicit intents are often used to activate components in other applications. For example −

```
Intent read1=new Intent();
read1.setAction(android.content.Intent.ACTION_VIEW);
read1.setData(ContactsContract.Contacts.CONTENT_URI);
startActivity(read1);
```

- Above code will give result as shown below

- The target component which receives the intent can use the **getExtras()** method to get the extra data sent by the source component. For example −

```
// Get bundle object at appropriate place in your code
Bundle extras = getIntent().getExtras();

// Extract data using passed keys
String value1 = extras.getString("Key1");
String value2 = extras.getString("Key2");
```

- Example
- Following example shows the functionality of a Android Intent to launch various Android built-in applications.
-

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *My Application* under a package *com.example.saira_000.myapplication*.
2	Modify *src/main/java/MainActivity.java* file and add the code to define two listeners corresponding two buttons ie. Start Browser and Start Phone.
3	Modify layout XML file *res/layout/activity_main.xml* to add three buttons in linear layout.
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

- Following is the content of the modified main activity file **src/com.example.My Application/MainActivity.java**.

```
package com.example.saira_000.myapplication;

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 b1,b2;

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

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

b1=(Button)findViewById(R.id.button);
b1.setOnClickListener(new View.OnClickListener() {

@Override
public void onClick(View v) {
Intent i = new Intent(android.content.Intent.ACTION_VIEW,
Uri.parse("http://www.example.com"));
startActivity(i);
}
});

b2=(Button)findViewById(R.id.button2);
b2.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent i = new Intent(android.content.Intent.ACTION_VIEW,
Uri.parse("tel:9510300000"));
startActivity(i);
}
});
}
}
```

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

```
<?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">

<TextView
android:id="@+id/textView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Intent Example"
android:layout_alignParentTop="true"
android:layout_centerHorizontal="true"
android:textSize="30dp" />

<TextView
android:id="@+id/textView2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Tutorials point"
android:textColor="#ff87ff09"
android:textSize="30dp"
```

```
android:layout_below="@+id/textView1"
android:layout_centerHorizontal="true" />

<ImageButton
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/imageButton"
android:src="@drawable/abc"
android:layout_below="@+id/textView2"
android:layout_centerHorizontal="true" />

<EditText
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/editText"
android:layout_below="@+id/imageButton"
android:layout_alignRight="@+id/imageButton"
android:layout_alignEnd="@+id/imageButton" />

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Start Browser"
android:id="@+id/button"
android:layout_alignTop="@+id/editText"
android:layout_alignRight="@+id/textView1"
android:layout_alignEnd="@+id/textView1"
android:layout_alignLeft="@+id/imageButton"
android:layout_alignStart="@+id/imageButton" />

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Start Phone"
android:id="@+id/button2"
android:layout_below="@+id/button"
android:layout_alignLeft="@+id/button"
android:layout_alignStart="@+id/button"
android:layout_alignRight="@+id/textView2"
android:layout_alignEnd="@+id/textView2" />
</RelativeLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">My Applicaiton</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.saira_000.myapplication">
```

```
<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>
</application>
</manifest>
```

- Let's try to run your **My Application** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar.Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −

- Now click on **Start Browser** button, which will start a browser configured and display http://www.example.com as shown below –

- Similar way you can launch phone interface using Start Phone button, which will allow you to dial already given phone number.

- **INTENT FILTERS**
- You have seen how Intent has been used to call another activity. Android OS uses filters to pinpoint the set of Activities, Services, and Broadcast receivers that can handle the Intent with help of specified set of action, categories, data scheme associated with Intent. You will

use **<intent-filter>** element in the manifest file to list down actions, categories and data types associated with any activity, service, or broadcast receiver.

- Following is an example of a part of **AndroidManifest.xml** file to specify an activity **com.example.My Application.CustomActivity** which can be invoked by either of the two mentioned actions, one category, and one data −

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

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

</activity>
```

- Once this activity is defined along with above mentioned filters, other activities will be able to invoke this activity using either the **android.intent.action.VIEW**, or using the **com.example.My Application.LAUNCH** action provided their category is **android.intent.category.DEFAULT**.
- The **<data>** element specifies the data type expected by the activity to be called and for above example our custom activity expects the data to start with the "http://"
- There may be a situation that intent can pass through the filters of more than one activity or service, the user may be asked which component to activate. An exception is raised if no target can be found.
- There are following test Android checks before invoking an activity −
- A filter <intent-filter> may list more than one action as shown above but this list cannot be empty; a filter must contain at least one <action> element, otherwise it will block all intents. If more than one actions are mentioned then Android tries to match one of the mentioned actions before invoking the activity.
- A filter <intent-filter> may list zero, one or more than one categories if there is no category mentioned then Android always pass this test but if more than one categories are mentioned then for an intent to pass the category test, every category in the Intent object must match a category in the filter.
- Each <data> element can specify a URI and a data type (MIME media type). There are separate attributes like **scheme, host, port**, and **path** for each part of the URI. An Intent object that contains both a URI and a data type passes the data type part of the test only if its type matches a type listed in the filter.

- Example
- Following example is a modification of the above example. Here we will see how Android resolves conflict if one intent is invoking two activities defined in , next how to invoke a custom activity using a filter and third one is an exception case if Android does not file appropriate activity defined for an intent.

Step	Description
1	You will use android studio to create an Android application and name it as *My Application* under a package *com.example.tutorial.myapplication;*.
2	Modify *src/Main/Java/MainActivity.java* file and add the code to define three listeners

	corresponding to three buttons defined in layout file.
3	Add a new *src/Main/Java/CustomActivity.java* file to have one custom activity which will be invoked by different intents.
4	Modify layout XML file *res/layout/activity_main.xml* to add three buttons in linear layout.
5	Add one layout XML file *res/layout/custom_view.xml* to add a simple <TextView> to show the passed data through intent.
6	Modify *AndroidManifest.xml* to add <intent-filter> to define rules for your intent to invoke custom activity.
7	Run the application to launch Android emulator and verify the result of the changes done in the application.

- Following is the content of the modified main activity file **src/MainActivity.java**.

```
package com.example.tutorial.myapplication;

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 b1,b2,b3;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
b1=(Button)findViewById(R.id.button);
b1.setOnClickListener(new View.OnClickListener() {

@Override
public void onClick(View v) {
Intent i = new Intent(android.content.Intent.ACTION_VIEW,
Uri.parse("http://www.example.com"));
startActivity(i);
}
});

b2 = (Button)findViewById(R.id.button2);
b2.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent i = new Intent("com.example.
tutorial.myapplication.
LAUNCH",Uri.parse("http://www.example.com"));
startActivity(i);
}
});

b3 = (Button)findViewById(R.id.button3);
b3.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
```

```
Intent i = new Intent("com.example.
My Application.LAUNCH",
Uri.parse("https://www.example.com"));
startActivity(i);
}
});
}
}
```

- Following is the content of the modified main activity file **src/com.example.My Application/CustomActivity.java**.

```
package com.example.tutorial.myapplication;

import android.app.Activity;
import android.net.Uri;
import android.os.Bundle;
import android.widget.TextView;

/**
Created by Tutorial on 8/23/2016.
*/
public class CustomActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.custom_view);
TextView label = (TextView) findViewById(R.id.show_data);
Uri url = getIntent().getData();
label.setText(url.toString());
}
}
```

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

```
<?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"
tools:context="com.example.tutorial.myapplication.MainActivity">

<TextView
android:id="@+id/textView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Intent Example"
android:layout_alignParentTop="true"
android:layout_centerHorizontal="true"
```

```
android:textSize="30dp" />

<TextView
android:id="@+id/textView2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Tutorials point"
android:textColor="#ff87ff09"
android:textSize="30dp"
android:layout_below="@+id/textView1"
android:layout_centerHorizontal="true" />

<ImageButton
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/imageButton"
android:src="@drawable/abc"
android:layout_below="@+id/textView2"
android:layout_centerHorizontal="true" />

<EditText
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/editText"
android:layout_below="@+id/imageButton"
android:layout_alignRight="@+id/imageButton"
android:layout_alignEnd="@+id/imageButton" />

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Start Browser"
android:id="@+id/button"
android:layout_alignTop="@+id/editText"
android:layout_alignLeft="@+id/imageButton"
android:layout_alignStart="@+id/imageButton"
android:layout_alignEnd="@+id/imageButton" />

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Start browsing with launch action"
android:id="@+id/button2"
android:layout_below="@+id/button"
android:layout_alignLeft="@+id/button"
android:layout_alignStart="@+id/button"
android:layout_alignEnd="@+id/button" />
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Exceptional condition"
android:id="@+id/button3"
android:layout_below="@+id/button2"
android:layout_alignLeft="@+id/button2"
android:layout_alignStart="@+id/button2"
```

```
android:layout_toStartOf="@+id/editText"
android:layout_alignParentEnd="true" />
</RelativeLayout>
```

Following will be the content of **res/layout/custom_view.xml** file −

```
<?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/show_data"
android:layout_width="fill_parent"
android:layout_height="400dp"/>
</LinearLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">My Application</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

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

<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="com.example.tutorial.myapplication.CustomActivity">

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

</activity>
</application>

</manifest>
```

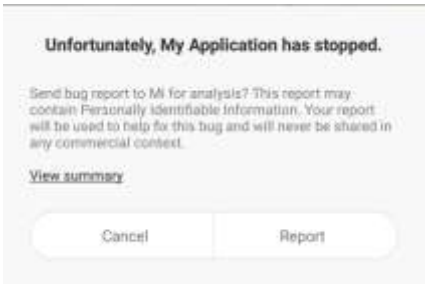
- Let's try to run your **My Application** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar. Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −

- Now let's start with first button "Start Browser with VIEW Action". Here we have defined our custom activity with a filter "android.intent.action.VIEW", and there is already one default activity against VIEW action defined by Android which is launching web browser, So android displays following two options to select the activity you want to launch.

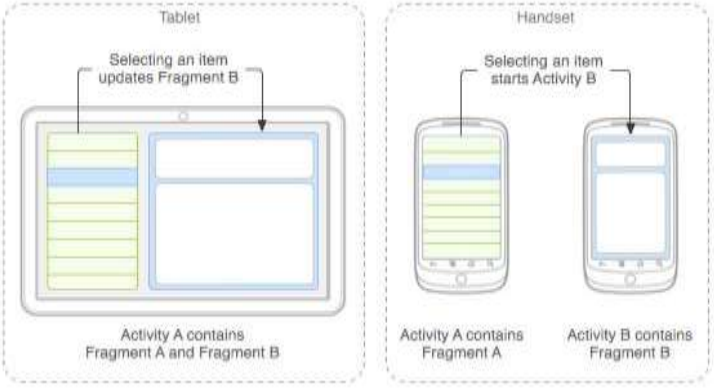
- Now if you select Browser, then Android will launch web browser and open example.com website but if you select IndentDemo option then Android will launch CustomActivity which does nothing but just capture passed data and displays in a text view as follows –

- Now go back using back button and click on "Start Browser with LAUNCH Action" button, here Android applies filter to choose define activity and it simply launch your custom activity
- Again, go back using back button and click on "Exception Condition" button, here Android tries to find out a valid filter for the given intent but it does not find a valid activity defined

because this time we have used data as **https** instead of **http** though we are giving a correct action, so Android raises an exception and shows following screen −



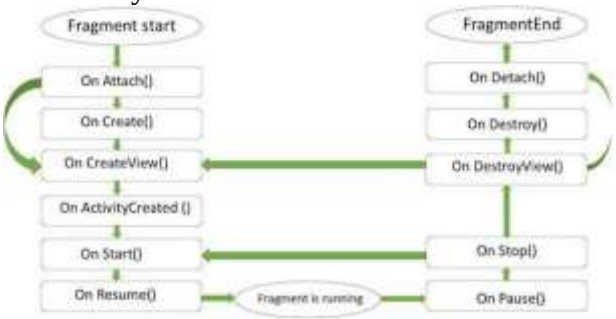
- **FRAGMENTS**
- A **Fragment** is a piece of an activity which enable more modular activity design. It will not be wrong if we say, a fragment is a kind of **sub-activity**.
- Following are important points about fragment −
- A fragment has its own layout and its own behaviour with its own life cycle callbacks.
- You can add or remove fragments in an activity while the activity is running.
- You can combine multiple fragments in a single activity to build a multi-plane UI.
- A fragment can be used in multiple activities.
- Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped.
- A fragment can implement a behaviour that has no user interface component.
- Fragments were added to the Android API in Honeycomb version of Android which API version 11.
- You create fragments by extending **Fragment** class and You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a **<fragment>** element.
- Prior to fragment introduction, we had a limitation because we can show only a single activity on the screen at one given point in time. So we were not able to divide device screen and control different parts separately. But with the introduction of fragment we got more flexibility and removed the limitation of having a single activity on the screen at a time. Now we can have a single activity but each activity can comprise of multiple fragments which will have their own layout, events and complete life cycle.
- Following is a typical example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.



- The application can embed two fragments in Activity A, when running on a tablet-sized device. However, on a handset-sized screen, there's not enough room for both fragments, so

Activity A includes only the fragment for the list of articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article.

- **FRAGMENT LIFE CYCLE**
- Android fragments have their own life cycle very similar to an android activity. This section briefs different stages of its life cycle.



- ***FRAGMENT LIFECYCLE***
- Here is the list of methods which you can to override in your fragment class –
- **onAttach()**The fragment instance is associated with an activity instance.The fragment and the activity is not fully initialized. Typically you get in this method a reference to the activity which uses the fragment for further initialization work.
- **onCreate()** The system calls this method when creating the fragment. You should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
- **onCreateView()** The system calls this callback when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a **View** component from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.
- **onActivityCreated()**The onActivityCreated() is called after the onCreateView() method when the host activity is created. Activity and fragment instance have been created as well as the view hierarchy of the activity. At this point, view can be accessed with the findViewById() method. example. In this method you can instantiate objects which require a Context object
- **onStart()**The onStart() method is called once the fragment gets visible.
- **onResume()**Fragment becomes active.
- **onPause()** The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session.
- **onStop()**Fragment going to be stopped by calling onStop()
- **onDestroyView()**Fragment view will destroy after call this method
- **onDestroy()**onDestroy() called to do final clean up of the fragment's state but Not guaranteed to be called by the Android platform.

- **How to use Fragments?**
- This involves number of simple steps to create Fragments.
- First of all decide how many fragments you want to use in an activity. For example let's we want to use two fragments to handle landscape and portrait modes of the device.
- Next based on number of fragments, create classes which will extend the *Fragment* class. The Fragment class has above mentioned callback functions. You can override any of the functions based on your requirements.

- Corresponding to each fragment, you will need to create layout files in XML file. These files will have layout for the defined fragments.
- Finally modify activity file to define the actual logic of replacing fragments based on your requirement.

- **Types of Fragments**
- Basically fragments are divided as three stages as shown below.
- Single frame fragments − Single frame fragments are using for hand hold devices like mobiles, here we can show only one fragment as a view.
- List fragments − fragments having special list view is called as list fragment
- Fragments transaction − Using with fragment transaction. we can move one fragment to another fragment.

- Intent Object to Invoke Built-in Application

  - Android provides Built-in applications for phone calls, in some occasions we may need to make a phone call through our application. This could easily be done by using implicit Intent with appropriate actions.
  - You can use Android Intent to make phone call by calling built-in Phone Call functionality of the Android. Following section explains different parts of our Intent object required to make a call.
  - You will use **ACTION_CALL** action to trigger built-in phone call functionality available in Android device. Following is simple syntax to create an intent with ACTION_CALL action

    ```
    Intent phoneIntent = new Intent(Intent.ACTION_CALL);
    ```

  - You can use **ACTION_DIAL** action instead of ACTION_CALL, in that case you will have option to modify hardcoded phone number before making a call instead of making a direct call.

  - Intent Object Data Type to make Phone Call
  - To make a phone call at a given number 91-000-000-0000, you need to specify **tel:** as URI using setData() method as follows −

    ```
    phoneIntent.setData(Uri.parse("tel:91-000-000-0000"));
    ```

  - The interesting point is that, to make a phone call, you do not need to specify any extra data or data type.

Example

- Following example shows you in practical how to use Android Intent to make phone call to the given mobile number.

Step	Description

1	You will use Android studio IDE to create an Android application and name it as *My Application* under a package *com.example.saira_000.myapplication*.
2	Modify *src/MainActivity.java* file and add required code to take care of making a call.
3	Modify layout XML file *res/layout/activity_main.xml* add any GUI component if required. I'm adding a simple button to Call 91-000-000-0000 number
4	No need to define default string constants.Android studio takes care of default constants.
5	Modify *AndroidManifest.xml* as shown below
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

- To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.
- Following is the content of the modified main activity file **src/MainActivity.java**.

```
package com.example.saira_000.myapplication;
import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {
  private Button button;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    button = (Button) findViewById(R.id.buttonCall);

    button.setOnClickListener(new View.OnClickListener() {
      public void onClick(View arg0) {
        Intent callIntent = new Intent(Intent.ACTION_CALL);
        callIntent.setData(Uri.parse("tel:0377778888"));

        if (ActivityCompat.checkSelfPermission(MainActivity.this,
        Manifest.permission.CALL_PHONE) !=
PackageManager.PERMISSION_GRANTED) {
          return;
        }
        startActivity(callIntent);
      }
    });
  }
}
```

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

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

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <Button
      android:id="@+id/buttonCall"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="call 0377778888" />
</LinearLayout>
```

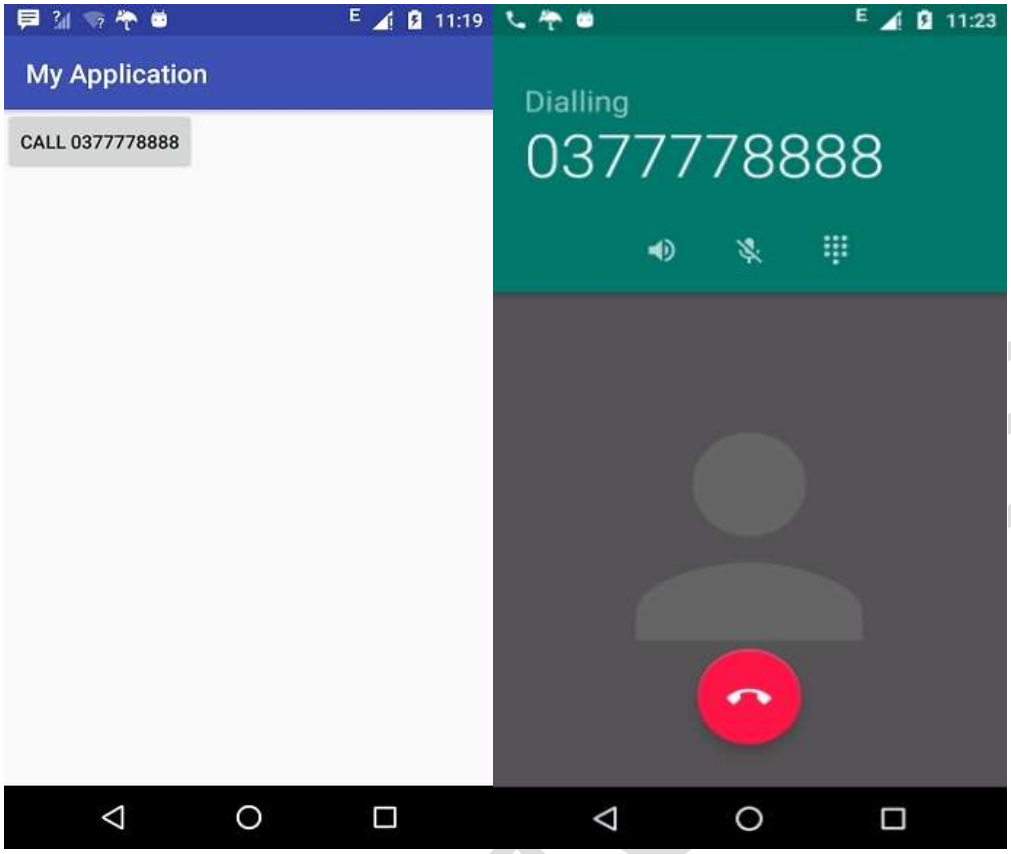
- Following will be the content of **res/values/strings.xml** to define two new constants −

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">My Application</string>
</resources>
```

- Following is the default content of **AndroidManifest.xml** −

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.saira_000.myapplication" >

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

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >

      <activity
         android:name="com.example.saira_000.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 **My Application** 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.Select your mobile device as an option and then check your mobile device which will display following screen –



- Now use **Call** button to make phone call as shown above.

UNIT 2

- **UI Design:**
- LinearLayout
- Android LinearLayout is a view group that aligns all children in either *vertically* or *horizontally*.

- ***LINEAR LAYOUT***
- LinearLayout Attributes
- Following are the important attributes specific to LinearLayout −

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:baselineAligned** This must be a boolean value, either "true" or "false" and prevents the layout from aligning its children's baselines.
3	**android:baselineAlignedChildIndex** When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align.
4	**android:divider** This is drawable to use as a vertical divider between buttons. You use a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
5	**android:gravity** This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
6	**android:orientation** This specifies the direction of arrangement and you will use "horizontal" for a row, "vertical" for a column. The default is horizontal.
7	**android:weightSum** Sum up of child weight

- Example

- This example will take you through simple steps to show how to create your own Android application using Linear Layout. Follow the following steps to modify the Android application we created in *Hello World Example* chapter −

Step	Description
1	You will use Android Studio to create an Android application and name it as *Demo* under a package *com.example.demo*as explained in the *Hello World Example* chapter.
2	Modify the default content of *res/layout/activity_main.xml*file to include few buttons in linear layout.
3	No need to change string Constants.Android studio takes care of default strings
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

```
package com.example.demo;

import android.os.Bundle;
import android.app.Activity;

public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
}
```

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

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

  <Button android:id="@+id/btnStartService"
    android:layout_width="270dp"
    android:layout_height="wrap_content"
    android:text="start_service"/>

  <Button android:id="@+id/btnPauseService"
    android:layout_width="270dp"
    android:layout_height="wrap_content"
    android:text="pause_service"/>

  <Button android:id="@+id/btnStopService"
    android:layout_width="270dp"
    android:layout_height="wrap_content"
    android:text="stop_service"/>
```

```
</LinearLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">HelloWorld</string>
   <string name="action_settings">Settings</string>
</resources>
```

- Let's try to run our modified **Hello World!** application we just modified. I assume you had created your **AVD** while doing environment setup. To run the app from Android studio,

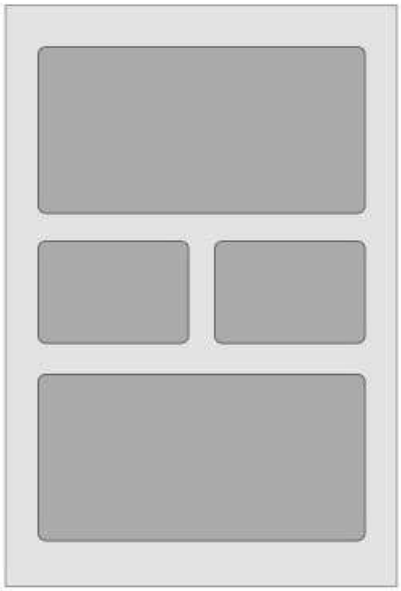
open one of your project's activity files and click Run icon from the toolbar. Android studio installs theapp on your AVD and starts it and if everything is fine with your setup

- and application, it will display following Emulator window −

Now let's change the orientation of Layout as **android:orientation="horizontal"** and try to run the same application, it will give following screen –

- RelativeLayout

- Android RelativeLayout enables you to specify how child views are positioned relative to each other. The position of each view can be specified as relative to sibling elements or relative to the parent.



- ***RELATIVE LAYOUT***
- RelativeLayout Attributes
- Following are the important attributes specific to RelativeLayout −

Sr.No.	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:gravity** This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
3	**android:ignoreGravity** This indicates what view should not be affected by gravity.

- Using RelativeLayout, you can align two elements by right border, or make one below another, centered in the screen, centered left, and so on. By default, all child views are drawn at the top-left of the layout, so you must define the position of each view using the various layout properties available from **RelativeLayout.LayoutParams** and few of the important attributes are given below −

Sr.No.	Attribute & Description
1	**android:layout_above** Positions the bottom edge of this view above the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name"
2	**android:layout_alignBottom** Makes the bottom edge of this view match the bottom edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
3	**android:layout_alignLeft** Makes the left edge of this view match the left edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
4	**android:layout_alignParentBottom** If true, makes the bottom edge of this view match the bottom edge of the parent. Must be a

	boolean value, either "true" or "false".
5	**android:layout_alignParentEnd** If true, makes the end edge of this view match the end edge of the parent. Must be a boolean value, either "true" or "false".
6	**android:layout_alignParentLeft** If true, makes the left edge of this view match the left edge of the parent. Must be a boolean value, either "true" or "false".
7	**android:layout_alignParentRight** If true, makes the right edge of this view match the right edge of the parent. Must be a boolean value, either "true" or "false".
8	**android:layout_alignParentStart** If true, makes the start edge of this view match the start edge of the parent. Must be a boolean value, either "true" or "false".
9	**android:layout_alignParentTop** If true, makes the top edge of this view match the top edge of the parent. Must be a boolean value, either "true" or "false".
10	**android:layout_alignRight** Makes the right edge of this view match the right edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
11	**android:layout_alignStart** Makes the start edge of this view match the start edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
12	**android:layout_alignTop** Makes the top edge of this view match the top edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
13	**android:layout_below** Positions the top edge of this view below the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
14	**android:layout_centerHorizontal** If true, centers this child horizontally within its parent. Must be a boolean value, either "true" or "false".
15	**android:layout_centerInParent** If true, centers this child horizontally and vertically within its parent. Must be a boolean value, either "true" or "false".
16	**android:layout_centerVertical** If true, centers this child vertically within its parent. Must be a boolean value, either "true" or "false".
17	**android:layout_toEndOf** Positions the start edge of this view to the end of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
18	**android:layout_toLeftOf** Positions the right edge of this view to the left of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
19	**android:layout_toRightOf** Positions the left edge of this view to the right of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
20	**android:layout_toStartOf** Positions the end edge of this view to the start of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

- Example
- This example will take you through simple steps to show how to create your own Android application using Relative Layout. Follow the following steps to modify the Android application we created in *Hello World Example* chapter −

Step	Description
1	You will use Android Studio IDE to create an Android application and name it as *demo* under a package *com.example.demo* as explained in the *Hello World Example*chapter.
2	Modify the default content of *res/layout/activity_main.xml*file to include few widgets in Relative layout.
3	Define required constants in *res/values/strings.xml* file
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

```
package com.example.demo;

import android.os.Bundle;
import android.app.Activity;

public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

}
```

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

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:paddingLeft="16dp"
  android:paddingRight="16dp" >

  <EditText
    android:id="@+id/name"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/reminder" />

  <LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_alignParentStart="true"
    android:layout_below="@+id/name">

    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="New Button"
```

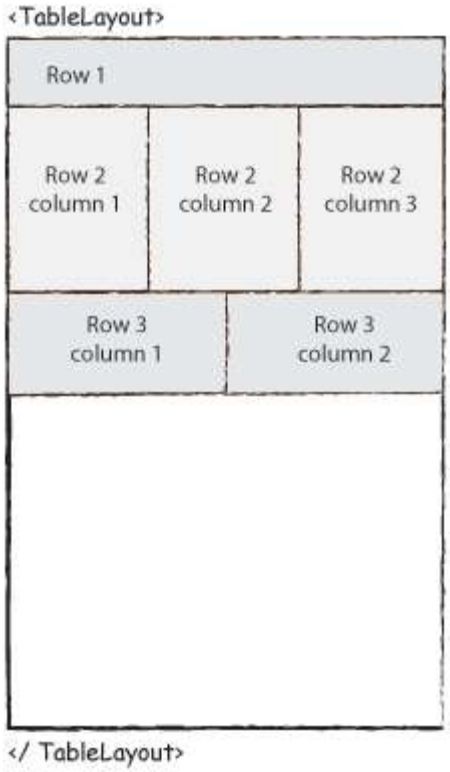
```
        android:id="@+id/button" />

      <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Button"
        android:id="@+id/button2" />

    </LinearLayout>

</RelativeLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="action_settings">Settings</string>
  <string name="reminder">Enter your name</string>
</resources>
```

- Let's try to run our modified **Hello World!** application we just modified. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar. Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −

- **Android Table Layout**
  - Android TableLayout going to be arranged groups of views into rows and columns. You will use the <TableRow> element to build a row in the table. Each row has zero or more cells; each cell can hold one View object.
  - TableLayout containers do not display border lines for their rows, columns, or cells.



TableLayout Attributes

Following are the important attributes specific to TableLayout −

Sr.No.	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:collapseColumns** This specifies the zero-based index of the columns to collapse. The column indices must be separated by a comma: 1, 2, 5.
3	**android:shrinkColumns** The zero-based index of the columns to shrink. The column indices must be separated by a comma: 1, 2, 5.
4	**android:stretchColumns** The zero-based index of the columns to stretch. The column indices must be separated by a comma: 1, 2, 5.

- Example
- This example will take you through simple steps to show how to create your own Android application using Table Layout. Follow the following steps to modify the Android application we created in *Hello World Example* chapter −

Step	Description
1	You will use Android Studio IDE to create an Android application and name it as *demo* under a package *com.example.demo* as explained in the *Hello World Example*chapter.
2	Modify the default content of *res/layout/activity_main.xml*file to include few widgets in table layout.

3	No need to modify string.xml, Android studio takes care of default constants
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

```
package com.example.demo;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
}
```

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

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">

  <TableRow
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
      android:text="Time"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_column="1" />

    <TextClock
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/textClock"
      android:layout_column="2" />

  </TableRow>

  <TableRow>

    <TextView
      android:text="First Name"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_column="1" />
```

```
      <EditText
        android:width="200px"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    </TableRow>

    <TableRow>

      <TextView
        android:text="Last Name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="1" />

      <EditText
        android:width="100px"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    </TableRow>

    <TableRow
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">

      <RatingBar
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/ratingBar"
        android:layout_column="2" />
    </TableRow>

    <TableRow
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"/>

    <TableRow
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">

      <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Submit"
        android:id="@+id/button"
        android:layout_column="2" />
    </TableRow>

  </TableLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">HelloWorld</string>
```

```
    <string name="action_settings">Settings</string>
</resources>
```

- Let's try to run our modified **Hello World!** application we just modified. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar. Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −

- Android Frame Layout
    - Frame Layout is designed to block out an area on the screen to display a single item. Generally, FrameLayout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other.
    - FrameLayout Attributes
    - Following are the important attributes specific to FrameLayout −

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:foreground** This defines the drawable to draw over the content and possible values may be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
3	**android:foregroundGravity** Defines the gravity to apply to the foreground drawable. The gravity defaults to fill. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
4	**android:measureAllChildren** Determines whether to measure all children or just those in the VISIBLE or INVISIBLE state when measuring. Defaults to false.

- Example

- This example will take you through simple steps to show how to create your own Android application using frame layout. Follow the following steps to modify the Android application we created in *Hello World Example* chapter −

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *demo* under a package *com.example.demo* as explained in the *Hello World Example*chapter.
2	Modify the default content of *res/layout/activity_main.xml*file to include few widgets in frame layout.
3	No need to change string.xml, android takes care default constants
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

```
package com.example.demo;

import android.os.Bundle;
import android.app.Activity;

public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
}
```

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

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">

  <ImageView
    android:src="@drawable/ic_launcher"
    android:scaleType="fitCenter"
    android:layout_height="250px"
    android:layout_width="250px"/>

  <TextView
    android:text="Frame Demo"
    android:textSize="30px"
    android:textStyle="bold"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent"
    android:gravity="center"/>
</FrameLayout>
```

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

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

```
<resources>
   <string name="app_name">demo</string>
   <string name="action_settings">Settings</string>
</resources>
```

- Let's try to run our modified **Hello World!** application we just modified. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar. Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −

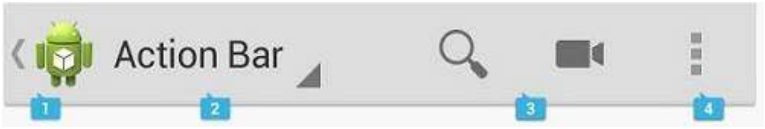
Action Bar

The action bar is a dedicated bar at the top of each screen that is generally persistent throughout the app. It provides you several key function which are as following −

- Makes important actions prominent and accessible
- Supports consistent navigation and view switching within apps
- Reduces clutter by providing an action overflow for rarely used actions
- Provides a dedicated space for giving your app an identity

Action Bar Components

Action Bar has four major components which can be seen in the following image.



These components name and functionality is discussed below −

Sr.No	Action Bar Components
1	**App Icon** The app icon establishes your app's identity. It can be replaced with a different logo or branding if you wish.

2	**View control** If your app displays data in different views, this segment of the action bar allows users to switch views.
3	**Action buttons** Show the most important actions of your app in the actions section.
4	**Action overflow** Move less often used actions to the action overflow.

TextView Control

A **TextView** displays text to the user and optionally allows them to edit it. A TextView is a complete text editor, however the basic class is configured to not allow editing.

TextView Attributes

Following are the important attributes related to TextView control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Sr.No.	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the control.
2	**android:capitalize** If set, specifies that this TextView has a textual input method and should automatically capitalize what the user types. • Don't automatically capitalize anything - 0 • Capitalize the first word of each sentence - 1 • Capitalize the first letter of every word - 2 • Capitalize every character - 3
3	**android:cursorVisible** Makes the cursor visible (the default) or invisible. Default is false.
4	**android:editable** If set to true, specifies that this TextView has an input method.
5	**android:fontFamily** Font family (named by string) for the text.
6	**android:gravity** Specifies how to align the text by the view's x- and/or y-axis when the text is smaller than the view.
7	**android:hint** Hint text to display when the text is empty.
8	**android:inputType** The type of data being placed in a text field. Phone, Date, Time, Number, Password etc.
9	**android:maxHeight** Makes the TextView be at most this many pixels tall.
10	**android:maxWidth** Makes the TextView be at most this many pixels wide.
11	**android:minHeight** Makes the TextView be at least this many pixels tall.
12	**android:minWidth** Makes the TextView be at least this many pixels wide.
13	**android:password** Whether the characters of the field are displayed as password dots instead of themselves. Possible value either "true" or "false".
14	**android:phoneNumber**

	If set, specifies that this TextView has a phone number input method. Possible value either "true" or "false".
15	**android:text** Text to display.
16	**android:textAllCaps** Present the text in ALL CAPS. Possible value either "true" or "false".
17	**android:textColor** Text color. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
18	**android:textColorHighlight** Color of the text selection highlight.
19	**android:textColorHint** Color of the hint text. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
20	**android:textIsSelectable** Indicates that the content of a non-editable text can be selected. Possible value either "true" or "false".
21	**android:textSize** Size of the text. Recommended dimension type for text is "sp" for scaled-pixels (example: 15sp).
22	**android:textStyle** Style (bold, italic, bolditalic) for the text. You can use or more of the following values separated by '|'. • normal - 0 • bold - 1 • italic - 2
23	**android:typeface** Typeface (normal, sans, serif, monospace) for the text. You can use or more of the following values separated by '|'. • normal - 0 • sans - 1 • serif - 2 • monospace - 3

- Example
- This example will take you through simple steps to show how to create your own Android application using Linear Layout and TextView.

Step	Description
1	You will use Android studio to create an Android application and name it as *demo* under a package *com.example.demo*as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add necessary code .
2	Modify the default content of *res/layout/activity_main.xml*file to include Android UI control.
3	No need to change default string constants at string.xml file. Android studio takes care of default string constants.
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

```
package com.example.demo;
```

```
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    //--- text view---
    TextView txtView = (TextView) findViewById(R.id.text_id);
  }
}
```

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

```
<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/text_id"
    android:layout_width="300dp"
    android:layout_height="200dp"
    android:capitalize="characters"
    android:text="hello_world"
    android:textColor="@android:color/holo_blue_dark"
    android:textColorHighlight="@android:color/primary_text_dark"
    android:layout_centerVertical="true"
    android:layout_alignParentEnd="true"
    android:textSize="50dp"/>

</RelativeLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">demo</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

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

```
<application
  android:allowBackup="true"
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:supportsRtl="true"
  android:theme="@style/AppTheme" >

  <activity
    android:name="com.example.demo.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 **demo** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −

- **EditText Control**
- A EditText is an overlay over TextView that configures itself to be editable. It is the predefined subclass of TextView that includes rich editing capabilities.

- **EditText Attributes**
- Following are the important attributes related to EditText control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.
- Inherited from **android.widget.TextView** Class −

Sr.No	Attribute & Description
1	**android:autoText** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom** This is the drawable to be drawn below the text.
3	**android:drawableRight** This is the drawable to be drawn to the right of the text.
4	**android:editable** If set, specifies that this TextView has an input method.
5	**android:text** This is the Text to display.

- Inherited from **android.view.View** Class −

Sr.No	Attribute & Description
1	**android:background** This is a drawable to use as the background.
2	**android:contentDescription** This defines text that briefly describes content of the view.
3	**android:id** This supplies an identifier name for this view.
4	**android:onClick** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility** This controls the initial visibility of the view.

- Example
- This example will take you through simple steps to show how to create your own Android application using Linear Layout and EditText.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *demo* under a package *com.example.demo* as explained in the *Hello World Example*chapter.
2	Modify *src/MainActivity.java* file to add a click event.
3	Modify the default content of *res/layout/activity_main.xml*file to include Android UI control.
4	Define required necessary string constants in *res/values/strings.xml* file
5	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

```
package com.example.demo;

import android.os.Bundle;
import android.app.Activity;

import android.view.View;
import android.view.View.OnClickListener;

import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {
  EditText eText;
  Button btn;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    eText = (EditText) findViewById(R.id.edittext);
    btn = (Button) findViewById(R.id.button);
    btn.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        String str = eText.getText().toString();
        Toast msg = Toast.makeText(getBaseContext(),str,Toast.LENGTH_LONG);
        msg.show();
      }
    });
  }
}
```

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

```
<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/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="14dp"
    android:layout_marginTop="18dp"
    android:text="@string/example_edittext" />
```

```
<Button
  android:id="@+id/button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignLeft="@+id/textView1"
  android:layout_below="@+id/textView1"
  android:layout_marginTop="130dp"
  android:text="@string/show_the_text" />

<EditText
  android:id="@+id/edittext"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:layout_alignLeft="@+id/button"
  android:layout_below="@+id/textView1"
  android:layout_marginTop="61dp"
  android:ems="10"
  android:text="@string/enter_text" android:inputType="text" />

</RelativeLayout>
```

- Following will be the content of **res/values/strings.xml** to define these new constants −

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">demo</string>
  <string name="example_edittext">Example showing EditText</string>
  <string name="show_the_text">Show the Text</string>
  <string name="enter_text">text changes</string>
</resources>
```

- Following is the default content of **AndroidManifest.xml** −

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

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.demo.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 **demo** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −

- **Button View**
- A Button is a Push-button which can be pressed, or clicked, by the user to perform an action.

- **Button Attributes**
- Following are the important attributes related to Button control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.
- Inherited from **android.widget.TextView** Class −

Sr.No	Attribute & Description
1	**android:autoText** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom** This is the drawable to be drawn below the text.
3	**android:drawableRight** This is the drawable to be drawn to the right of the text.
4	**android:editable** If set, specifies that this TextView has an input method.
5	**android:text** This is the Text to display.

- Inherited from **android.view.View** Class −

Attribute	Description
1	**android:background**

	This is a drawable to use as the background.
2	**android:contentDescription** This defines text that briefly describes content of the view.
3	**android:id** This supplies an identifier name for this view.
4	**android:onClick** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility** This controls the initial visibility of the view.

- Example
- This example will take you through simple steps to show how to create your own Android application using Linear Layout and Button.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *myapplication* under a package *com.example.saira_000.myapplication* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
3	Modify the default content of *res/layout/activity_main.xml*file to include Android UI control.
4	No need to declare default string constants at string.xml, Android studio takes care of default string constants.
5	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

```
package com.example.saira_000.myapplication;

import android.content.Intent;
import android.net.Uri;
import android.support.v7.app.ActionBarActivity;
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 ActionBarActivity {
  Button b1,b2,b3;

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

    b1=(Button)findViewById(R.id.button);
    b1.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
```

```
            Toast.makeText(MainActivity.this,"YOUR
MESSAGE",Toast.LENGTH_LONG).show();
         }
      });
   }
}
```

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

```
<?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">

   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Button Control"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:textSize="30dp" />

   <TextView
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tutorials point"
      android:textColor="#ff87ff09"
      android:textSize="30dp"
      android:layout_below="@+id/textView1"
      android:layout_centerHorizontal="true" />

   <ImageButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/imageButton"
      android:src="@drawable/abc"
      android:layout_below="@+id/textView2"
      android:layout_centerHorizontal="true" />

   <EditText
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/editText"
      android:layout_below="@+id/imageButton"
      android:layout_alignRight="@+id/imageButton"
      android:layout_alignEnd="@+id/imageButton" />

   <Button
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button"
    android:id="@+id/button"
    android:layout_alignTop="@+id/editText"
    android:layout_alignLeft="@+id/textView1"
    android:layout_alignStart="@+id/textView1"
    android:layout_alignRight="@+id/editText"
    android:layout_alignEnd="@+id/editText" />

</RelativeLayout>
```

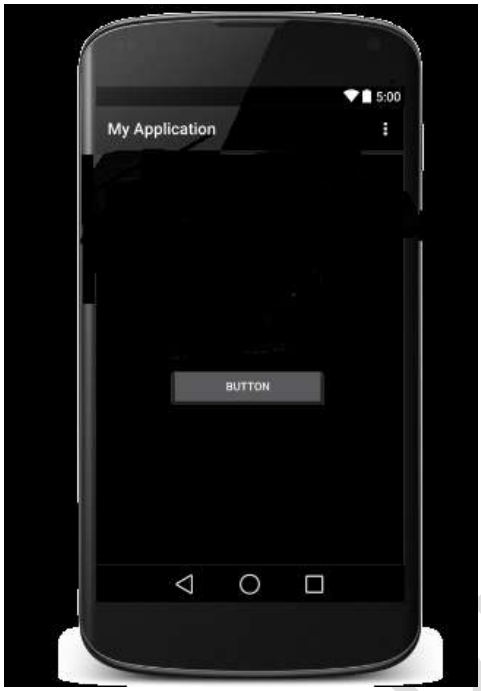
- Following will be the content of **res/values/strings.xml** to define these new constants −

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">myapplication</string>
</resources>
```

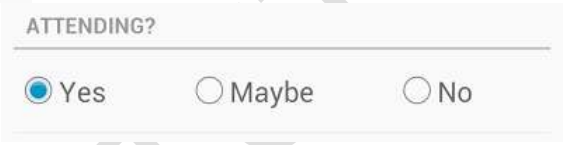
- Following is the default content of **AndroidManifest.xml** −

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.saira_000.myapplication" >

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.guidemo4.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 **GUIDemo4** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar.Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −



- **RadioButton View,**
- A RadioButton has two states: either checked or unchecked.This allows the user to select one option from a set.



- Example
- This example will take you through simple steps to show how to create your own Android application using Linear Layout and RadioButton.

Step	Description
1	You will use Android studio to create an Android application and name it as *My Application* under a package *com.example.saira_000.myapplication* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
2	Modify the default content of *res/layout/activity_main.xml*file to include Android UI control.
3	Android studio takes care of default constants so no need to declare default constants at string.xml file
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

- Following is the content of the modified main activity file **src/MainActivity.java**. This file can include each of the fundamental lifecycle methods.
- In the below example abc indicates the image of tutorial

```
package com.example.saira_000.myapplication;

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

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

public class MainActivity extends ActionBarActivity {
  RadioGroup rg1;
  RadioButton rb1;
  Button b1;

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

  private void addListenerRadioButton() {
    rg1 = (RadioGroup) findViewById(R.id.radioGroup);
    b1 = (Button) findViewById(R.id.button2);
    b1.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        int selected=rg1.getCheckedRadioButtonId();
        rb1=(RadioButton)findViewById(selected);
        Toast.makeText(MainActivity.this,rb1.getText(),Toast.LENGTH_LONG).show();
      }
    });
  }
}
```

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

```
<?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"
  tools:context=".MainActivity">

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Example of Radio Button"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="30dp" />

  <TextView
    android:id="@+id/textView2"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials point"
    android:textColor="#ff87ff09"
    android:textSize="30dp"
    android:layout_above="@+id/imageButton"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="40dp" />

  <ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button2"
    android:text="ClickMe"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true" />

  <RadioGroup
    android:id="@+id/radioGroup"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_below="@+id/imageButton"
    android:layout_alignLeft="@+id/textView2"
    android:layout_alignStart="@+id/textView2">

    <RadioButton
      android:layout_width="142dp"
      android:layout_height="wrap_content"
      android:text="JAVA"
      android:id="@+id/radioButton"
      android:textSize="25dp"
      android:textColor="@android:color/holo_red_light"
      android:checked="false"
      android:layout_gravity="center_horizontal" />

    <RadioButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="ANDROID"
      android:id="@+id/radioButton2"
      android:layout_gravity="center_horizontal"
      android:checked="false"
      android:textColor="@android:color/holo_red_dark"
      android:textSize="25dp" />

    <RadioButton
      android:layout_width="136dp"
```

```
        android:layout_height="wrap_content"
        android:text="HTML"
        android:id="@+id/radioButton3"
        android:layout_gravity="center_horizontal"
        android:checked="false"
        android:textSize="25dp"
        android:textColor="@android:color/holo_red_dark" />

    </RadioGroup>

</RelativeLayout>
```

- Following will be the content of **res/values/strings.xml** to define these new constants −

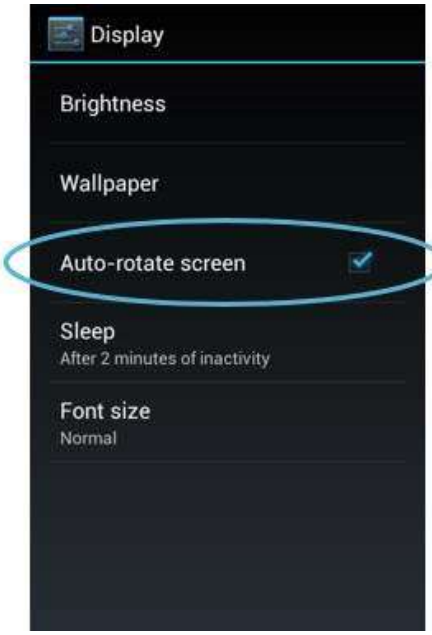
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My Application</string>
</resources>
```

- Following is the default content of **AndroidManifest.xml** −

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.saira_000.myapplication" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="com.example.My Application.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 **My Application** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar. Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application.

- ## CheckBox View
- A CheckBox is an on/off switch that can be toggled by the user. You should use check-boxes when presenting users with a group of selectable options that are not mutually exclusive.



- **CheckBox Attributes**
- Following are the important attributes related to CheckBox control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.
- Inherited from **android.widget.TextView** Class −

Sr.No	Attribute & Description
1	**android:autoText** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom** This is the drawable to be drawn below the text.
3	**android:drawableRight** This is the drawable to be drawn to the right of the text.
4	**android:editable** If set, specifies that this TextView has an input method.
5	**android:text** This is the Text to display.

Inherited from **android.view.View** Class −

Sr.No	Attribute & Description
1	**android:background** This is a drawable to use as the background.
2	**android:contentDescription** This defines text that briefly describes content of the view.
3	**android:id** This supplies an identifier name for this view.
4	**android:onClick** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility** This controls the initial visibility of the view.

- Example
- This example will take you through simple steps to show how to create your own Android application using Linear Layout and CheckBox.

Step	Description
1	You will use Android Studio IDE to create an Android application and name it as *myapplication* under a package *com.example.myapplication* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
3	Modify the default content of *res/layout/activity_main.xml*file to include Android UI control.
4	No need to declare default string constants. Android studio takes care of default constants at string.xml
5	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

```
package com.example.myapplication;

import android.os.Bundle;
import android.app.Activity;
import android.widget.Button;

import android.view.View;
import android.view.View.OnClickListener;

import android.widget.CheckBox;
import android.widget.Toast;

public class MainActivity extends Activity {
  CheckBox ch1,ch2;
  Button b1,b2;

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

    ch1=(CheckBox)findViewById(R.id.checkBox1);
    ch2=(CheckBox)findViewById(R.id.checkBox2);

    b1=(Button)findViewById(R.id.button);
    b2=(Button)findViewById(R.id.button2);
    b2.setOnClickListener(new View.OnClickListener() {

      @Override
      public void onClick(View v) {
        finish();
      }
    });
    b1.setOnClickListener(new View.OnClickListener() {

      @Override
```

```
        public void onClick(View v) {
          StringBuffer result = new StringBuffer();
          result.append("Thanks : ").append(ch1.isChecked());
          result.append("\nThanks: ").append(ch2.isChecked());
          Toast.makeText(MainActivity.this, result.toString(),
            Toast.LENGTH_LONG).show();
        }
      });
    }
}
```

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

```
<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/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Example of checkbox"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="30dp" />

  <CheckBox
    android:id="@+id/checkBox1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Do you like Tutorials Point"
    android:layout_above="@+id/button"
    android:layout_centerHorizontal="true" />

  <CheckBox
    android:id="@+id/checkBox2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Do you like android "
    android:checked="false"
    android:layout_above="@+id/checkBox1"
    android:layout_alignLeft="@+id/checkBox1"
    android:layout_alignStart="@+id/checkBox1" />

  <TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
```

```
      android:layout_height="wrap_content"
      android:layout_alignLeft="@+id/checkBox1"
      android:layout_below="@+id/textView1"
      android:layout_marginTop="39dp"
      android:text="Tutorials point"
      android:textColor="#ff87ff09"
      android:textSize="30dp"
      android:layout_alignRight="@+id/textView1"
      android:layout_alignEnd="@+id/textView1" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Ok"
      android:id="@+id/button"
      android:layout_alignParentBottom="true"
      android:layout_alignLeft="@+id/checkBox1"
      android:layout_alignStart="@+id/checkBox1" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Cancel"
      android:id="@+id/button2"
      android:layout_alignParentBottom="true"
      android:layout_alignRight="@+id/textView2"
      android:layout_alignEnd="@+id/textView2" />

   <ImageButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/imageButton"
      android:src="@drawable/abc"
      android:layout_centerVertical="true"
      android:layout_centerHorizontal="true" />

</RelativeLayout>
```

- Following will be the content of **res/values/strings.xml** to define these new constants −

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">MyApplication</string>
</resources>
```

- Following is the default content of **AndroidManifest.xml** −

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

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
```

```
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.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 **MyApplication** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −

- **ImageButton View**
- An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user.
- ImageButton Attributes
- Following are the important attributes related to ImageButton control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.
- Inherited from **android.widget.ImageView** Class −

Sr.No	Attribute & Description
1	**android:adjustViewBounds** Set this to true if you want the ImageView to adjust its bounds to preserve the aspect ratio of its drawable.
2	**android:baseline** This is the offset of the baseline within this view.
3	**android:baselineAlignBottom** If true, the image view will be baseline aligned with based on its bottom edge.
4	**android:cropToPadding** If true, the image will be cropped to fit within its padding.
5	**android:src** This sets a drawable as the content of this ImageView.

Inherited from **android.view.View** Class −

Sr.No	Attribute & Description
1	**android:background** This is a drawable to use as the background.
2	**android:contentDescription** This defines text that briefly describes content of the view.
3	**android:id** This supplies an identifier name for this view
4	**android:onClick** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility** This controls the initial visibility of the view.

- Example
- This example will take you through simple steps to show how to create your own Android application using Linear Layout and ImageButton.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *myapplication* under a package *com.example.myapplication* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
3	Modify the default content of *res/layout/activity_main.xml*file to include Android UI control.
4	No need to define default constants in android, Android studio takes care of default constants.
5	Run the application to launch Android emulator and verify the result of the changes done in the application.

- Following is the content of the modified main activity file **src/com.example.myapplication/MainActivity.java**. This file can include each of the fundamental lifecycle methods.
- In the below example abc indicates the image of tutorial

```
package com.example.myapplication;

import android.os.Bundle;
import android.app.Activity;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;
```

```
import android.widget.Toast;

public class MainActivity extends Activity {
  ImageButton imgButton;

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

    imgButton =(ImageButton)findViewById(R.id.imageButton);
    imgButton.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Toast.makeText(getApplicationContext(),"You download is
          resumed",Toast.LENGTH_LONG).show();
      }
    });
  }
}
```

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

```
<?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">

  <TextView android:text="Tutorials Point"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="30dp"
    android:layout_alignParentTop="true"
    android:layout_alignRight="@+id/imageButton"
    android:layout_alignEnd="@+id/imageButton" />

  <ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true"
    android:src="@drawable/abc"/>

</RelativeLayout>
```

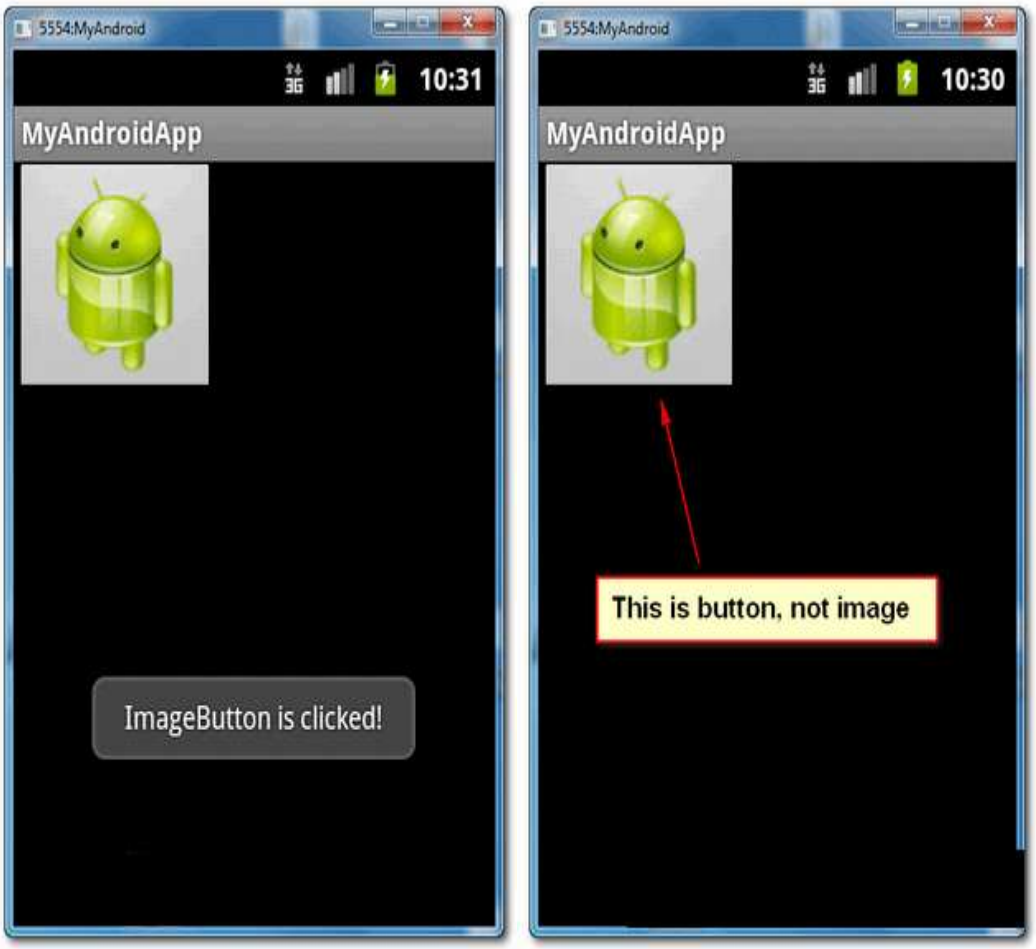
- Following will be the content of **res/values/strings.xml** to define these new constants −

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">myapplication</string>
</resources>
```

- Following is the default content of **AndroidManifest.xml** −

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

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.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 **myapplication** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar. Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −



- **ToggleButton View**
- A ToggleButton displays checked/unchecked states as a button. It is basically an on/off button with a light indicator.
- **ToggleButton Attributes**
- Following are the important attributes related to ToggleButton control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Sr.No.	Attribute & Description
1	**android:disabledAlpha** This is the alpha to apply to the indicator when disabled.
2	**android:textOff** This is the text for the button when it is not checked.
3	**android:textOn** This is the text for the button when it is checked.

- Inherited from **android.widget.TextView** Class −

Sr.No.	Attribute & Description
1	**android:autoText** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom** This is the drawable to be drawn below the text.

3	**android:drawableRight** This is the drawable to be drawn to the right of the text.
4	**android:editable** If set, specifies that this TextView has an input method.
5	**android:text** This is the Text to display.

- Inherited from **android.view.View** Class −

Sr.No.	Attribute & Description
1	**android:background** This is a drawable to use as the background.
2	**android:contentDescription** This defines text that briefly describes content of the view.
3	**android:id** This supplies an identifier name for this view,
4	**android:onClick** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility** This controls the initial visibility of the view.

- Example
- This example will take you through simple steps to show how to create your own Android application using Linear Layout and ToggleButton.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *My Application* under a package *com.example.saira_000.myapplication* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
2	Modify the default content of *res/layout/activity_main.xml*file to include Android UI control.
3	No need to declare default constants.Android studio takes care of default constants at string.xml
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

- Following is the content of the modified main activity file **src/MainActivity.java**. This file can include each of the fundamental lifecycle methods.
- In the below example abc indicates the image of tutorial

```
package com.example.saira_000.myapplication;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import android.widget.ToggleButton;

public class MainActivity extends ActionBarActivity {
  ToggleButton tg1,tg2;
  Button b1;
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```
        tg1=(ToggleButton)findViewById(R.id.toggleButton);
        tg2=(ToggleButton)findViewById(R.id.toggleButton2);

        b1=(Button)findViewById(R.id.button2);
        b1.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View v) {
            StringBuffer result = new StringBuffer();
            result.append("You have clicked first ON Button-:) ").append(tg1.getText());
            result.append("You have clicked Second ON Button -:) ").append(tg2.getText());
            Toast.makeText(MainActivity.this, result.toString(),Toast.LENGTH_SHORT).show();
          }
        });
      }
    }
```

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

```
<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/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials point"
    android:textColor="#ff87ff09"
    android:textSize="30dp"
    android:layout_above="@+id/imageButton"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="40dp" />

  <ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />

  <ToggleButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="On"
    android:id="@+id/toggleButton"
    android:checked="true"
    android:layout_below="@+id/imageButton"
    android:layout_toEndOf="@+id/button2/>
```

```
    <ToggleButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Off"
      android:id="@+id/toggleButton2"
      android:checked="true"
      android:layout_alignTop="@+id/toggleButton" />

    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/button2"
      android:text="ClickMe"
      android:layout_alignParentBottom="true"
      android:layout_centerHorizontal="true" />

  </RelativeLayout>
```

- Following will be the content of **res/values/strings.xml** to define these new constants −

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">My Application</string>
</resources>
```

- Following is the default content of **AndroidManifest.xml** −

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.saira_000.myapplication" >

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.My Application.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 **My Application** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Android studio

installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −

- The following screen will appear −

- If you have clicked first on Button, you would get a message on Toast as **You have clicked first ON Button-:)** or else if you clicked on second on button, you would get a message on Toast as **You have clicked Second ON Button -:)**

- **RatingBar View**
- In Android, you can use "android.widget.RatingBar" to display rating bar component in stars icon. The user is able to touch, drag or click on the stars to set the rating value easily.

1. Rating Bar

Open "**res/layout/main.xml**" file, add a rating bar component, few textviews and a button.

File : res/layout/main.xml

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

  <TextView
    android:id="@+id/lblRateMe"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Rate Me"
    android:textAppearance="?android:attr/textAppearanceMedium" />

  <RatingBar
    android:id="@+id/ratingBar"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:numStars="4"
        android:stepSize="1.0"
        android:rating="2.0" />

    <Button
        android:id="@+id/btnSubmit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Submit" />

    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/lblResult"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Result : "
            android:textAppearance="?android:attr/textAppearanceLarge" />

        <TextView
            android:id="@+id/txtRatingValue"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text=""
            android:textAppearance="?android:attr/textAppearanceSmall" />

    </LinearLayout>

</LinearLayout>
```

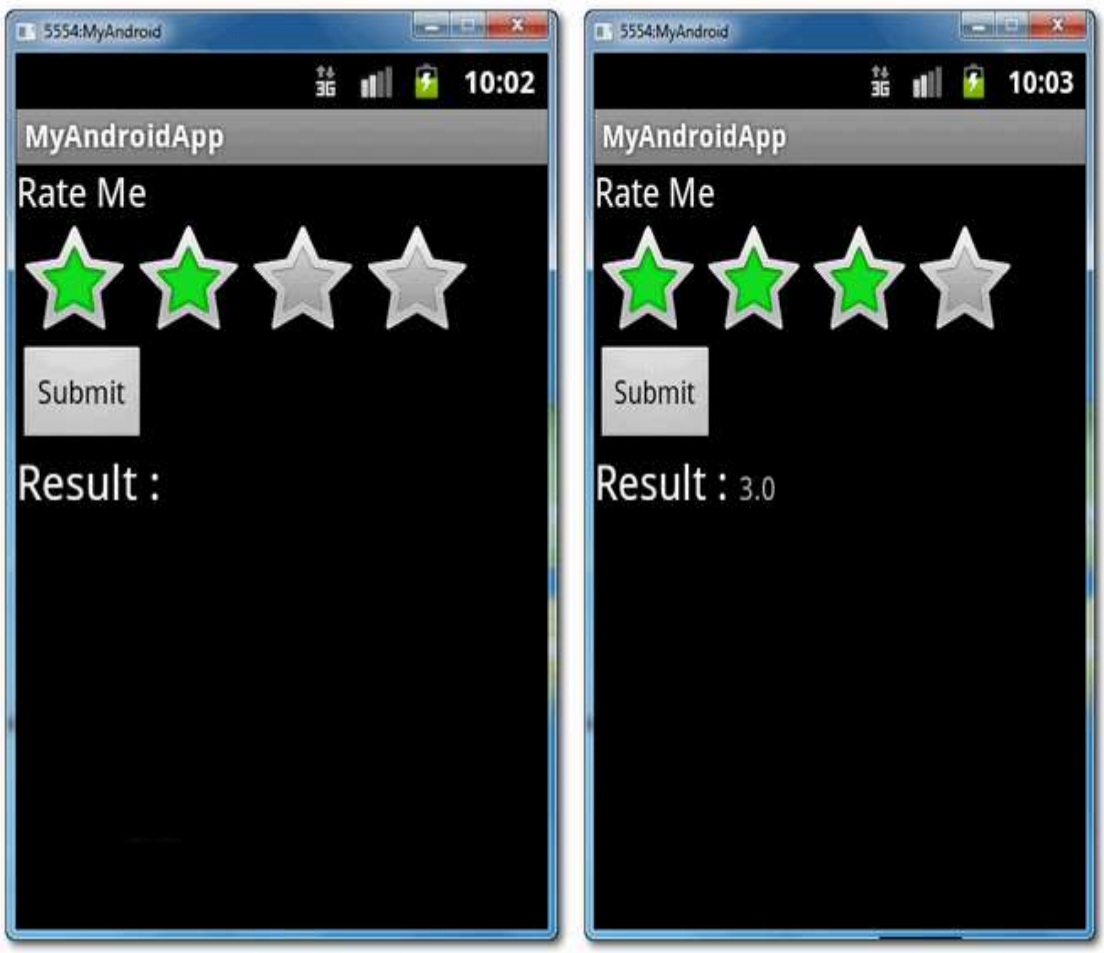
2. Code Code

Inside activity "onCreate()" method, attach a listener on rating bar, fire when rating value is changed. Another listener on button, fire when button is clicked. Read the code's comment, it should be self-explanatory.

File : MyAndroidAppActivity.java

```
package com.mkyong.android;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.RatingBar;
```

```
import android.widget.RatingBar.OnRatingBarChangeListener;
import android.widget.TextView;
import android.widget.Toast;

public class MyAndroidAppActivity extends Activity {

  private RatingBar ratingBar;
  private TextView txtRatingValue;
  private Button btnSubmit;

  @Override
  public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);

   addListenerOnRatingBar();
   addListenerOnButton();

  }

  public void addListenerOnRatingBar() {

   ratingBar = (RatingBar) findViewById(R.id.ratingBar);
   txtRatingValue = (TextView) findViewById(R.id.txtRatingValue);

   //if rating value is changed,
   //display the current rating value in the result (textview) automatically
   ratingBar.setOnRatingBarChangeListener(new OnRatingBarChangeListener() {
          public void onRatingChanged(RatingBar ratingBar, float rating,
                  boolean fromUser) {

                  txtRatingValue.setText(String.valueOf(rating));
          }
   });
  }
  public void addListenerOnButton() {

   ratingBar = (RatingBar) findViewById(R.id.ratingBar);
   btnSubmit = (Button) findViewById(R.id.btnSubmit);

   //if click on me, then display the current rating value.
   btnSubmit.setOnClickListener(new OnClickListener() {

          @Override
          public void onClick(View v) {

                  Toast.makeText(MyAndroidAppActivity.this,
                          String.valueOf(ratingBar.getRating()),
                                  Toast.LENGTH_SHORT).show();
          }

   });

  }
}
```



UI Events:

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

here are following three concepts related to Android Event Management −

- **Event Listeners** − An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.

- **Event Listeners Registration** − Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.

- **Event Handlers** − When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

Event Listeners & Event Handlers

Event Handler	Event Listener & Description
onClick()	**OnClickListener()** This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.
onLongClick()	**OnLongClickListener()** This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	**OnFocusChangeListener()** This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.
onKey()	**OnFocusChangeListener()** This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.
onTouch()	**OnTouchListener()** This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.
onMenuItemClick()	**OnMenuItemClickListener()** This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.
onCreateContextMenu()	**onCreateContextMenuItemListener()** This is called when the context menu is being built(as the result of a sustained "long click)

- There are many more event listeners available as a part of **View**class like OnHoverListener, OnDragListener etc which may be needed for your application. So I recommend to refer official documentation for Android application development in case you are going to develop a sophisticated apps.
- Event Listeners Registration
- Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event. Though there are several tricky ways to register your event listener for any event, but I'm going to list down only top 3 ways, out of which you can use any of them based on the situation.
- Using an Anonymous Inner Class
- Activity class implements the Listener interface.
- Using Layout file activity_main.xml to specify event handler directly.

Below section will provide you detailed examples on all the three scenarios −

- Touch Mode
- Users can interact with their devices by using hardware keys or buttons or touching the screen.Touching the screen puts the device into touch mode. The user can then interact with it by touching the on-screen virtual buttons, images, etc.You can check if the device is in touch mode by calling the View class's isInTouchMode() method.

- Focus
- A view or widget is usually highlighted or displays a flashing cursor when it's in focus. This indicates that it's ready to accept input from the user.

- **isFocusable()** − it returns true or false

- **isFocusableInTouchMode()** − checks to see if the view is focusable in touch mode. (A view may be focusable when using a hardware key but not when the device is in touch mode)

android:foucsUp="@=id/button_l"

onTouchEvent()

```
public boolean onTouchEvent(motionEvent event){
  switch(event.getAction()){
    case TOUCH_DOWN:
    Toast.makeText(this,"you have clicked down Touch
button",Toast.LENTH_LONG).show();
    break();

    case TOUCH_UP:
    Toast.makeText(this,"you have clicked up touch button",Toast.LENTH_LONG).show();
    break;

    case TOUCH_MOVE:
    Toast.makeText(this,"you have clicked move touch
button"Toast.LENTH_LONG).show();
    break;
  }
  return super.onTouchEvent(event) ;
}
```

- **Event Handling Examples**
- Event Listeners Registration Using an Anonymous Inner Class
- Here you will create an anonymous implementation of the listener and will be useful if each class is applied to a single control only and you have advantage to pass arguments to event handler. In this approach event handler methods can access private data of Activity. No reference is needed to call to Activity.

- But if you applied the handler to more than one control, you would have to cut and paste the code for the handler and if the code for the handler is long, it makes the code harder to maintain.

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

Step	Description

1	You will use Android studio IDE to create an Android application and name it as *myapplication* under a package *com.example.myapplication* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add click event listeners and handlers for the two buttons defined.
3	Modify the detault content of *res/layout/activity_main.xml*file to include Android UI controls.
4	No need to declare default string constants.Android studio takes care default constants.
5	Run the application to launch Android emulator and verify the result of the changes done in the aplication.

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

```
package com.example.myapplication;

import android.app.ProgressDialog;
import android.os.Bundle;
import android.support.v7.app.ActionBarActivity;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends ActionBarActivity {
  private ProgressDialog progress;
  Button b1,b2;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    progress = new ProgressDialog(this);

    b1=(Button)findViewById(R.id.button);
    b2=(Button)findViewById(R.id.button2);
    b1.setOnClickListener(new View.OnClickListener() {

      @Override
      public void onClick(View v) {
        TextView txtView = (TextView) findViewById(R.id.textView);
        txtView.setTextSize(25);
      }
    });

    b2.setOnClickListener(new View.OnClickListener() {

      @Override
      public void onClick(View v) {
        TextView txtView = (TextView) findViewById(R.id.textView);
        txtView.setTextSize(55);
      }
```

```
        });
    }
}
```

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

Here abc indicates about tutorial logo

```
<?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"
  tools:context=".MainActivity">

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Event Handling "
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="30dp"/>

  <TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials point "
    android:textColor="#ff87ff09"
    android:textSize="30dp"
    android:layout_above="@+id/imageButton"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="40dp" />

  <ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />

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

```
    android:layout_height="wrap_content"
    android:text="Small font"
    android:id="@+id/button"
    android:layout_below="@+id/imageButton"
    android:layout_centerHorizontal="true" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Large Font"
    android:id="@+id/button2"
    android:layout_below="@+id/button"
    android:layout_alignRight="@+id/button"
    android:layout_alignEnd="@+id/button" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:id="@+id/textView"
    android:layout_below="@+id/button2"
    android:layout_centerHorizontal="true"
    android:textSize="25dp" />

</RelativeLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">myapplication</string>
</resources>
```

- Following is the default content of **AndroidManifest.xml** −

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

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.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 **myapplication** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar. Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −

- Now you try to click on two buttons, one by one and you will see that font of the **Hello World** text will change, which happens because registered click event handler method is being called against each click event.

- **Intercepting Touch Events**
- Android provides special types of touch screen events such as pinch , double tap, scrolls , long presses and flinch. These are all known as gestures.
- Android provides GestureDetector class to receive motion events and tell us that these events correspond to gestures or not. To use it , you need to create an object of GestureDetector and then extend another class with **GestureDetector.SimpleOnGestureListener** to act as a listener and override some methods. Its syntax is given below −

```
GestureDetector myG;
myG = new GestureDetector(this,new Gesture());

class Gesture extends GestureDetector.SimpleOnGestureListener{
  public boolean onSingleTapUp(MotionEvent ev) {
```

```
    }

    public void onLongPress(MotionEvent ev) {
    }

    public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX,
    float distanceY) {
    }

    public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
    float velocityY) {
    }
}
```

- Handling Pinch Gesture
- Android provides **ScaleGestureDetector** class to handle gestures like pinch e.t.c. In order to use it, you need to instantiate an object of this class. Its syntax is as follow −

```
ScaleGestureDetector SGD;
SGD = new ScaleGestureDetector(this,new ScaleListener());
```

- The first parameter is the context and the second parameter is the event listener. We have to define the event listener and override a function **OnTouchEvent** to make it working. Its syntax is given below −

```
public boolean onTouchEvent(MotionEvent ev) {
   SGD.onTouchEvent(ev);
   return true;
}

private class ScaleListener extends ScaleGestureDetector.SimpleOnScaleGestureListener {
   @Override
   public boolean onScale(ScaleGestureDetector detector) {
      float scale = detector.getScaleFactor();
      return true;
   }
}
```

- Apart from the pinch gestures , there are other methods available that notify more about touch events. They are listed below −

Sr.No	Method & description
1	**getEventTime()** This method get the event time of the current event being processed..
2	**getFocusX()** This method get the X coordinate of the current gesture's focal point.
3	**getFocusY()** This method get the Y coordinate of the current gesture's focal point.
4	**getTimeDelta()** This method return the time difference in milliseconds between the previous accepted scaling event and the current scaling event.
5	**isInProgress()** This method returns true if a scale gesture is in progress..
6	**onTouchEvent(MotionEvent event)**

	This method accepts MotionEvents and dispatches events when appropriate.

- Example
- Here is an example demonstrating the use of ScaleGestureDetector class. It creates a basic application that allows you to zoom in and out through pinch.
- To experiment with this example , you can run this on an actual device or in an emulator with touch screen enabled.

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 necessary code.
3	Modify the res/layout/activity_main to add respective XML components
4	Run the application and choose a running android device and install the application on it and verify the results

- Following is the content of the modified main activity file **src/MainActivity.java**.

```
package com.example.sairamkrishna.myapplication;

import android.app.Activity;
import android.graphics.Matrix;
import android.os.Bundle;

import android.view.MotionEvent;
import android.view.ScaleGestureDetector;
import android.widget.ImageView;

public class MainActivity extends Activity {
   private ImageView iv;
   private Matrix matrix = new Matrix();
   private float scale = 1f;
   private ScaleGestureDetector SGD;

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

      iv=(ImageView)findViewById(R.id.imageView);
      SGD = new ScaleGestureDetector(this,new ScaleListener());
   }

   public boolean onTouchEvent(MotionEvent ev) {
      SGD.onTouchEvent(ev);
      return true;
   }

   private class ScaleListener extends ScaleGestureDetector.
      SimpleOnScaleGestureListener {

      @Override
      public boolean onScale(ScaleGestureDetector detector) {
```

```
            scale *= detector.getScaleFactor();
            scale = Math.max(0.1f, Math.min(scale, 5.0f));
            matrix.setScale(scale, scale);
            iv.setImageMatrix(matrix);
            return true;
        }
    }
}
```

- Following is the modified content of the xml **res/layout/activity_main.xml**.
- Here abc indicates the logo of tutorial

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

   <TextView android:text="Gestures
      Example" 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:scaleType="matrix"
      android:layout_below="@+id/textView"
      android:layout_alignParentLeft="true"
      android:layout_alignParentStart="true"
      android:layout_alignParentBottom="true"
      android:layout_alignParentRight="true"
      android:layout_alignParentEnd="true" />
```

```
</RelativeLayout>
```

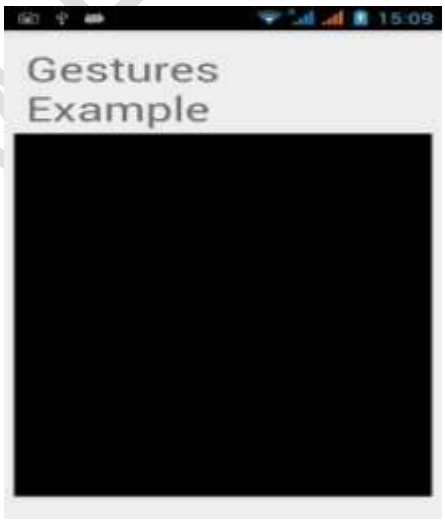
- Following is the content of the **res/values/string.xml**.

```
<resources>
  <string name="app_name>My Application</string>
</resources>
```

- Following is the content of **AndroidManifest.xml** file.

```
<?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="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.sairamkrishna.myapplicationMainActivity"
      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.The sample output should be like this –



- Now just place two fingers over android screen , and separate them a part and you will see that the android image is zooming.
- Now again place two fingers over android screen, and try to close them and you will see that the android image is now shrinking.

- **Data binding in applications**

- Data binding is a mechanism in applications that provides a simple and easy way for Windows Runtime apps to display and interact with data. In this mechanism, the management of data is entirely separated from the way data.

- Data binding allows the flow of data between UI elements and data object on user interface. When a binding is established and the data or your business model changes, then it reflects the updates automatically to the UI elements and vice versa. It is also possible to bind, not to a standard data source, but to another element on the page.

- **ListView**

- Android **ListView** is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an **Adapter** that pulls content from a source such as an array or database.

- An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter holds the data and send the data to adapter view, the view can takes the data from adapter view and shows the data on different views like as spinner, list view, grid view etc.

- The **ListView** and **GridView** are subclasses of **AdapterView**and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

- Android provides several subclasses of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView (i.e. ListView or GridView). The common adapters are **ArrayAdapter**,**Base Adapter**, **CursorAdapter**,**SimpleCursorAdapter**,**SpinnerAdapter** and **WrapperListAdapter**. We will see separate examples for both the adapters.

- **ListView Attributes**
- Following are the important attributes specific to GridView −

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:divider** This is drawable or color to draw between list items.
3	**android:dividerHeight** This specifies height of the divider. This could be in px, dp, sp, in, or mm.
4	**android:entries** Specifies the reference to an array resource that will populate the ListView.
5	**android:footerDividersEnabled** When set to false, the ListView will not draw the divider before each footer view. The default value is true.
6	**android:headerDividersEnabled** When set to false, the ListView will not draw the divider after each header view. The default value is true.

ArrayAdapter

You can use this adapter when your data source is an array. By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a **TextView**. Consider you have an array of strings you want to display in a ListView, initialize a new **ArrayAdapter** using a constructor to specify the layout for each string and the string array −

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,R.layout.ListView,StringArray);
```

Here are arguments for this constructor −

- First argument **this** is the application context. Most of the case, keep it **this**.
- Second argument will be layout defined in XML file and having **TextView** for each string in the array.
- Final argument is an array of strings which will be populated in the text view.

Once you have array adapter created, then simply call **setAdapter()** on your **ListView** object as follows −

```
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);
```

You will define your list view under res/layout directory in an XML file. For our example we are going to using activity_main.xml file.

- Example
- Following is the example which will take you through simple steps to show how to create your own Android application using ListView. Follow the following steps to modify the Android application we created in *Hello World Example* chapter −

Step	Description
1	You will use Android Studio IDE to create an Android application and name it as *ListDisplay* under a package *com.example.ListDisplay* as explained in the *Hello World Example* chapter.
2	Modify the default content of *res/layout/activity_main.xml*file to include ListView content with the self explanatory attributes.
3	No need to change string.xml, Android studio takes care of default string constants.
4	Create a Text View file *res/layout/activity_listview.xml*. This file will have setting to display all the list items. So you can customize its fonts, padding, color etc. using this file.
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

- Following is the content of the modified main activity file **src/com.example.ListDisplay/ListDisplay.java**. This file can include each of the fundamental life cycle methods.

```
package com.example.ListDisplay;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class ListDisplay extends Activity {
  // Array of strings...
  String[] mobileArray = {"Android","IPhone","WindowsMobile","Blackberry",
    "WebOS","Ubuntu","Windows7","Max OS X"};

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

    ArrayAdapter adapter = new ArrayAdapter<String>(this,
      R.layout.activity_listview, mobileArray);

    ListView listView = (ListView) findViewById(R.id.mobile_list);
    listView.setAdapter(adapter);
  }
}
```

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

```
<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"
  tools:context=".ListActivity" >

  <ListView
    android:id="@+id/mobile_list"
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
  </ListView>

</LinearLayout>
```

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

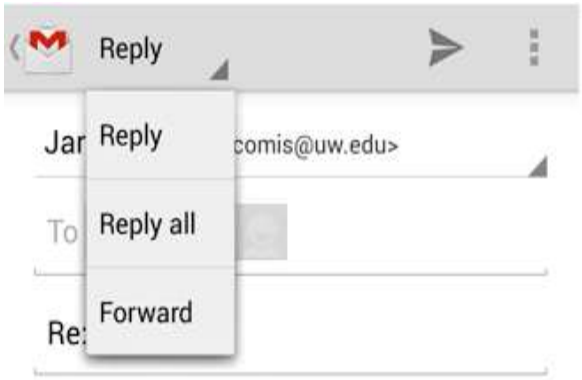
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">ListDisplay</string>
  <string name="action_settings">Settings</string>
</resources>
```

- Following will be the content of **res/layout/activity_listview.xml** file −

```
<?xml version="1.0" encoding="utf-8"?>
<!--  Single List Item Design -->

<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/label"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:padding="10dip"
  android:textSize="16dip"
  android:textStyle="bold" >
</TextView>
```

- Let's try to run our modified **Hello World!** application we just modified. I assume you had created your **AVD** while doing environment set-up. To run the app from Android studio, open one of your project's activity files and click Run icon from the tool bar. Android studio installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display following Emulator window −

- Spinner
- Spinner allows you to select an item from a drop down menu



- ***SPINNER EXAMPLE***
- Example
- This example demonstrates the category of computers, you need to select a category from the category.

- To experiment with this example, you need to run this on an actual device on after developing the application according to the steps below.

Steps	Description
1	You will use Android studio to create an Android application and name it as AndroidSpinnerExample under a package com.example.spinner.
2	Modify src/AndroidSpinnerExampleActivity.java file to create a simple list view with items which are showing as spinner items
3	Modify res/layout/activity_main.xml file to add respective XML code.
4	No need to define default string constants. Android studio takes care of default string constants at string.xml
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.spinner/AndroidSpinnerExampleActivity.java.**

```
package com.example.spinner;

import java.util.ArrayList;
import java.util.List;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;
import android.widget.AdapterView.OnItemSelectedListener;
```

```
class AndroidSpinnerExampleActivity extends Activity implements
OnItemSelectedListener{
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Spinner element
    Spinner spinner = (Spinner) findViewById(R.id.spinner);

    // Spinner click listener
    spinner.setOnItemSelectedListener(this);

    // Spinner Drop down elements
    List<String> categories = new ArrayList<String>();
    categories.add("Automobile");
    categories.add("Business Services");
    categories.add("Computers");
    categories.add("Education");
    categories.add("Personal");
    categories.add("Travel");

    // Creating adapter for spinner
    ArrayAdapter<String> dataAdapter = new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item, categories);

    // Drop down layout style - list view with radio button

dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

    // attaching data adapter to spinner
    spinner.setAdapter(dataAdapter);
  }

  @Override
  public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
    // On selecting a spinner item
    String item = parent.getItemAtPosition(position).toString();

    // Showing selected spinner item
    Toast.makeText(parent.getContext(), "Selected: " + item,
Toast.LENGTH_LONG).show();
  }
  public void onNothingSelected(AdapterView<?> arg0) {
    // TODO Auto-generated method stub
  }
}
```

- Modify the content of **res/layout/activity_main.xml** to the following

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:padding="10dip"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content">

  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dip"
    android:text="Category:"
    android:layout_marginBottom="5dp"/>

  <Spinner
    android:id="@+id/spinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:prompt="@string/spinner_title"/>

</LinearLayout>
```

Modify the **res/values/string.xml** to the following

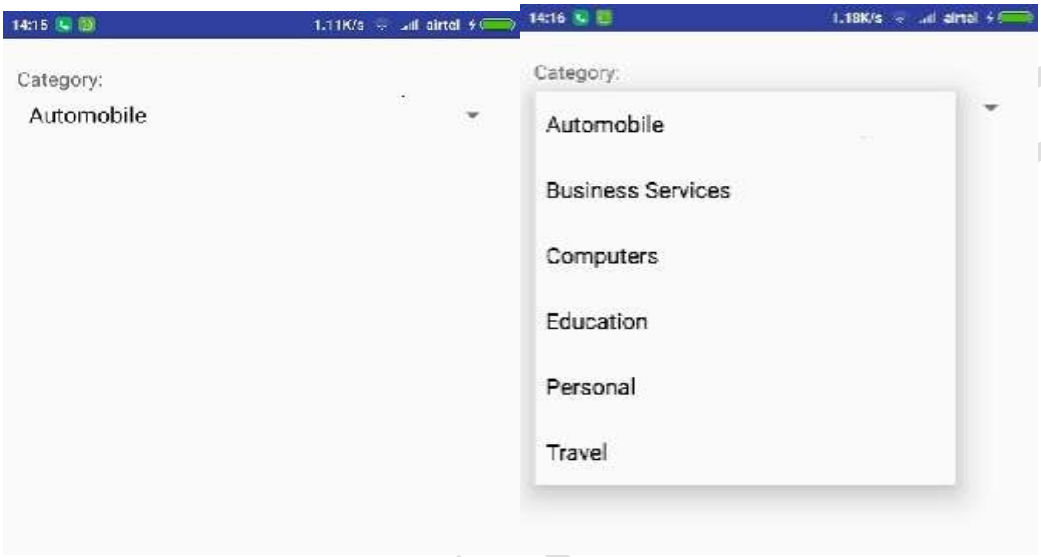
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">AndroidSpinnerExample</string>
</resources>
```

- This is the default **AndroidManifest.xml**

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

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.spinner.AndroidSpinnerExampleActivity"
      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 AndroidSpinnerExample 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.



- Android **GridView** shows items in two-dimensional scrolling grid (rows & columns) and the grid items are not necessarily predetermined but they automatically inserted to the layout using a **ListAdapter**
- An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter can be used to supply the data to like spinner, list view, grid view etc.
- The **ListView** and **GridView** are subclasses of **AdapterView**and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

- **GridView Attributes**
- Following are the important attributes specific to GridView −

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:columnWidth** This specifies the fixed width for each column. This could be in px, dp, sp, in, or mm.
3	**android:gravity** Specifies the gravity within each cell. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
4	**android:horizontalSpacing** Defines the default horizontal spacing between columns. This could be in px, dp, sp, in, or mm.
5	**android:numColumns** Defines how many columns to show. May be an integer value, such as "100" or auto_fit which

	means display as many columns as possible to fill the available space.
6	**android:stretchMode** Defines how columns should stretch to fill the available empty space, if any. This must be either of the values − • none − Stretching is disabled. • spacingWidth − The spacing between each column is stretched. • columnWidth − Each column is stretched equally. • spacingWidthUniform − The spacing between each column is uniformly stretched..
7	**android:verticalSpacing** Defines the default vertical spacing between rows. This could be in px, dp, sp, in, or mm.

- Example
- This example will take you through simple steps to show how to create your own Android application using GridView. Follow the following steps to modify the Android application we created in *Hello World Example* chapter −

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *HelloWorld* under a package *com.example.helloworld* as explained in the *Hello World Example* chapter.
2	Modify the detault content of *res/layout/activity_main.xml*file to include GridView content with the self explanatory attributes.
3	No need to change string.xml, Android studio takes care of defaults strings which are placed at string.xml
4	Let's put few pictures in *res/drawable-hdpi* folder. I have put sample0.jpg, sample1.jpg, sample2.jpg, sample3.jpg, sample4.jpg, sample5.jpg, sample6.jpg and sample7.jpg.
5	Create a new class called **ImageAdapter** under a package com.example.helloworld that extends BaseAdapter. This class will implement functionality of an adapter to be used to fill the view.
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

```
package com.example.helloworld;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.GridView;

public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    GridView gridview = (GridView) findViewById(R.id.gridview);
    gridview.setAdapter(new ImageAdapter(this));
  }
}
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/gridview"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:columnWidth="90dp"
   android:numColumns="auto_fit"
   android:verticalSpacing="10dp"
   android:horizontalSpacing="10dp"
   android:stretchMode="columnWidth"
   android:gravity="center"
/>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">HelloWorld</string>
   <string name="action_settings">Settings</string>
</resources>
```

- Following will be the content of **src/com.example.helloworld/ImageAdapter.java** file −

```
package com.example.helloworld;

import android.content.Context;

import android.view.View;
import android.view.ViewGroup;

import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;

public class ImageAdapter extends BaseAdapter {
   private Context mContext;

   // Constructor
   public ImageAdapter(Context c) {
      mContext = c;
   }

   public int getCount() {
      return mThumbIds.length;
   }

   public Object getItem(int position) {
      return null;
   }

   public long getItemId(int position) {
      return 0;
```

```
    }

    // create a new ImageView for each item referenced by the Adapter
    public View getView(int position, View convertView, ViewGroup parent) {
      ImageView imageView;

      if (convertView == null) {
        imageView = new ImageView(mContext);
        imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        imageView.setPadding(8, 8, 8, 8);
      }
      else
      {
        imageView = (ImageView) convertView;
      }
      imageView.setImageResource(mThumbIds[position]);
      return imageView;
    }

    // Keep all Images in array
    public Integer[] mThumbIds = {
      R.drawable.sample_2, R.drawable.sample_3,
      R.drawable.sample_4, R.drawable.sample_5,
      R.drawable.sample_6, R.drawable.sample_7,
      R.drawable.sample_0, R.drawable.sample_1,
      R.drawable.sample_2, R.drawable.sample_3,
      R.drawable.sample_4, R.drawable.sample_5,
      R.drawable.sample_6, R.drawable.sample_7,
      R.drawable.sample_0, R.drawable.sample_1,
      R.drawable.sample_2, R.drawable.sample_3,
      R.drawable.sample_4, R.drawable.sample_5,
      R.drawable.sample_6, R.drawable.sample_7
    };
}
```

- Let's try to run our modified **Hello World!** application we just modified. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar. Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −

AutoCompleteTextView

- If you want to get suggestions , when you type in an editable text field , you can do this via AutoCompleteTextView. It provides suggestions automatically when the user is typing. The list of suggestions is displayed in a drop down menu from which the user can choose an item to replace the content of the edit box with.
- In order to use AutoCompleteTextView you have to first create an AutoCompletTextView Field in the xml. Its syntax is given below.

```
<AutoCompleteTextView
   android:id="@+id/autoCompleteTextView1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_alignParentTop="true"
   android:layout_centerHorizontal="true"
   android:layout_marginTop="65dp"
   android:ems="10" >
```

- After that, you have to get a reference of this textview in java. Its syntax is given below.

```
private AutoCompleteTextView actv;
actv = (AutoCompleteTextView) findViewById(R.id.autoCompleteTextView1);
```

- The the next thing you need to do is to specify the list of suggestions items to be displayed. You can specify the list items as a string array in java or in strings.xml. Its syntax is given below.

```
String[] countries = getResources().getStringArray(R.array.list_of_countries);
ArrayAdapter<String> adapter = new ArrayAdapter<String>
  (this,android.R.layout.simple_list_item_1,countries);
actv.setAdapter(adapter);
```

- The array adapter class is responsible for displaying the data as list in the suggestion box of the text field. The **setAdapter**method is used to set the adapter of the autoCompleteTextView. Apart from these methods, the other methods of Auto Complete are listed below.

Sr.No	Method & description
1	**getAdapter()** This method returns a filterable list adapter used for auto completion
2	**getCompletionHint()** This method returns optional hint text displayed at the bottom of the the matching list
3	**getDropDownAnchor()** This method returns returns the id for the view that the auto-complete drop down list is anchored to.
4	**getListSelection()** This method returns the position of the dropdown view selection, if there is one
5	**isPopupShowing()** This method indicates whether the popup menu is showing
6	**setText(CharSequence text, boolean filter)** This method sets text except that it can disable filtering
7	**showDropDown()** This method displays the drop down on screen.

- Example
- The below example demonstrates the use of AutoCompleteTextView class. It crates a basic application that allows you to type in and it displays suggestions on your device.
- To experiment with this example , you need to run this on an actual device or in an emulator.

Steps	Description
1	You will use Android Studio to create an Android application under a package package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add AutoCompleteTextView 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.content.Context;

import android.media.AudioManager;
import android.media.MediaPlayer;
import android.media.MediaRecorder;

import android.os.Bundle;
import android.os.Environment;

import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;

import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;
import android.widget.Button;
import android.widget.EditText;
```

```
import android.widget.ImageView;
import android.widget.MultiAutoCompleteTextView;
import android.widget.Toast;
import java.io.IOException;


public class MainActivity extends Activity {
  AutoCompleteTextView text;
  MultiAutoCompleteTextView text1;
  String[] languages={"Android ","java","IOS","SQL","JDBC","Web services"};

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

    text=(AutoCompleteTextView)findViewById(R.id.autoCompleteTextView1);
    text1=(MultiAutoCompleteTextView)findViewById(R.id.multiAutoCompleteTextView1);

    ArrayAdapter adapter = new
      ArrayAdapter(this,android.R.layout.simple_list_item_1,languages);

    text.setAdapter(adapter);
    text.setThreshold(1);

    text1.setAdapter(adapter);
    text1.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
  }
}
```

- Here is the content of **activity_main.xml**
- Here abc indicates about logo of tutorial

```
<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">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Android Auto Complete"
    android:id="@+id/textView"
    android:textSize="30dp"
    android:layout_alignParentTop="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />
```

```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Tutorial"
  android:id="@+id/textView2"
  android:textColor="#ff3eff0f"
  android:textSize="35dp"
  android:layout_below="@+id/textView"
  android:layout_centerHorizontal="true" />

<ImageView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/imageView"
  android:src="@drawable/logo"
  android:layout_below="@+id/textView2"
  android:layout_alignLeft="@+id/textView2"
  android:layout_alignStart="@+id/textView2"
  android:layout_alignRight="@+id/textView2"
  android:layout_alignEnd="@+id/textView2" />

<AutoCompleteTextView
  android:id="@+id/autoCompleteTextView1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:ems="10"
  android:layout_below="@+id/imageView"
  android:layout_alignLeft="@+id/imageView"
  android:layout_alignStart="@+id/imageView"
  android:layout_marginTop="72dp"
  android:hint="AutoComplete TextView">
  <requestFocus />
</AutoCompleteTextView>

<MultiAutoCompleteTextView
  android:id="@+id/multiAutoCompleteTextView1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:ems="10"
  android:layout_below="@+id/autoCompleteTextView1"
  android:layout_alignLeft="@+id/autoCompleteTextView1"
  android:layout_alignStart="@+id/autoCompleteTextView1"
  android:hint="Multi Auto Complete " />

</RelativeLayout>
```

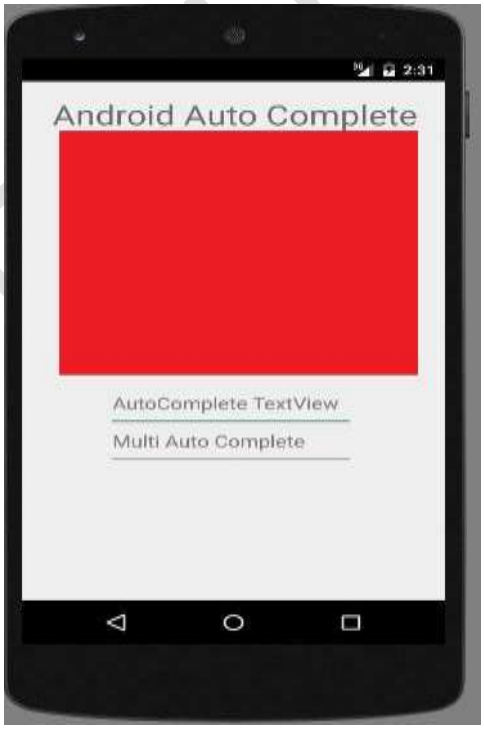
- Here is the content of **Strings.xml**

```
<resources>
  <string name="app_name">My Application</string>
</resources>
```

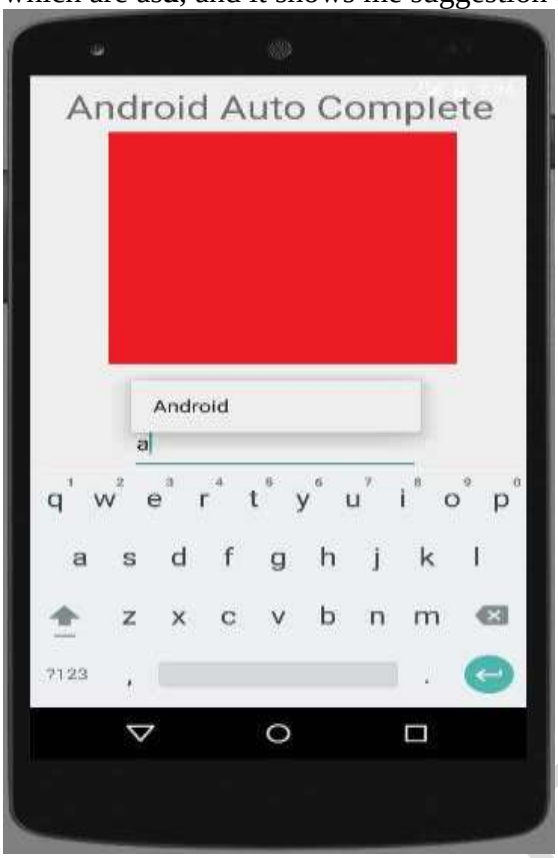
- Here is the content of **AndroidManifest.xml**

```
<?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="@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 AVD while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar. Android studio will install this application in your AVD and your AVD will display following screen.



Now just type in the text view to see suggestions of the Languages. As i just type one letter which are as**a**, and it shows me suggestion of language.



- **Displaying Pictures and Menus with Views**

ImageView

Displays image resources, for example Bitmap or Drawable resources. ImageView is also commonly used to apply tints to an image and handle image scaling.

The following XML snippet is a common example of using an ImageView to display an image resource:

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <ImageView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@mipmap/ic_launcher"
      />
</LinearLayout>
```

1. Add Image to Resources

Put your images into folder “res/drawable-ldpi“, “res/drawable-mdpi” or “res/drawable-hdpi“.

2. Add ImageView

Open "**res/layout/main.xml**" file, just add an ImageView and Button for demonstration. By default, imageView1 will display "android.png".

File : res/layout/main.xml

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

   <ImageView
      android:id="@+id/imageView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/android" />

   <Button
      android:id="@+id/btnChangeImage"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Change Image" />

</LinearLayout>
```

3. Code

Simple, when button is clicked, change it to "android3d.png".

File : MyAndroidAppActivity.java

```
package com.mkyong.android;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.ImageView;
import android.view.View;
import android.view.View.OnClickListener;
public class MyAndroidAppActivity extends Activity {
  Button button;
  ImageView image;
  @Override
  public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        addListenerOnButton();
  }
  public void addListenerOnButton() {
        image = (ImageView) findViewById(R.id.imageView1);
        button = (Button) findViewById(R.id.btnChangeImage);
        button.setOnClickListener(new OnClickListener() {
              @Override
              public void onClick(View arg0) {
                    image.setImageResource(R.drawable.android3d);
```

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

4. Demo

Run the application. 

1. Result, “android.png” is displayed. Click on the button, image will changed to “android3d.png”. 

- WebView
- WebView is a view that display web pages inside your application. You can also specify HTML string and can show it inside your application using WebView. WebView makes turns your application to a web application.
- In order to add WebView to your application, you have to add **<WebView>** element to your xml layout file. Its syntax is as follows −

```
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/webview"
  android:layout_width="fill_parent"
```

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

- In order to use it, you have to get a reference of this view in Java file. To get a reference, create an object of the class WebView. Its syntax is −

```
WebView browser = (WebView) findViewById(R.id.webview);
```

- In order to load a web url into the WebView, you need to call a method **loadUrl(String url)** of the WebView class, specifying the required url. Its syntax is:

```
browser.loadUrl("http://www.tutorial.com");
```

- Apart from just loading url, you can have more control over your WebView by using the methods defined in WebView class. They are listed as follows −

Sr.No	Method & Description
1	**canGoBack()** This method specifies the WebView has a back history item.
2	**canGoForward()** This method specifies the WebView has a forward history item.
3	**clearHistory()** This method will clear the WebView forward and backward history.
4	**destroy()** This method destroy the internal state of WebView.
5	**findAllAsync(String find)** This method find all instances of string and highlight them.
6	**getProgress()** This method gets the progress of the current page.
7	**getTitle()** This method return the title of the current page.
8	**getUrl()** This method return the url of the current page.

- If you click on any link inside the webpage of the WebView, that page will not be loaded inside your WebView. In order to do that you need to extend your class from **WebViewClient** and override its method. Its syntax is −

```
private class MyBrowser extends WebViewClient {
   @Override
   public boolean shouldOverrideUrlLoading(WebView view, String url) {
      view.loadUrl(url);
      return true;
   }
}
```

- **Example**
- Here is an example demonstrating the use of WebView Layout. It creates a basic web application that will ask you to specify a url and will load this url website in the WebView.
- To experiment with this example, you need to run this on an actual device on which internet is running.

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 WebView code.
3	Modify the res/layout/activity_main to add respective XML components
4	Modify the AndroidManifest.xml to add the necessary permissions
5	Run the application and choose a running android device and install the application on it and verify the results.

- Following is the content of the modified main activity file **src/MainActivity.java**.

```
package com.example.sairamkrishna.myapplication;
import android.app.Activity;
import android.os.Bundle;

import android.view.View;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Button;
import android.widget.EditText;


public class MainActivity extends Activity  {
   Button b1;
   EditText ed1;

   private WebView wv1;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      b1=(Button)findViewById(R.id.button);
      ed1=(EditText)findViewById(R.id.editText);

      wv1=(WebView)findViewById(R.id.webView);
      wv1.setWebViewClient(new MyBrowser());

      b1.setOnClickListener(new View.OnClickListener() {
       @Override
       public void onClick(View v) {
         String url = ed1.getText().toString();

         wv1.getSettings().setLoadsImagesAutomatically(true);
         wv1.getSettings().setJavaScriptEnabled(true);
         wv1.setScrollBarStyle(View.SCROLLBARS_INSIDE_OVERLAY);
         wv1.loadUrl(url);
       }
      });
   }

   private class MyBrowser extends WebViewClient {
      @Override
      public boolean shouldOverrideUrlLoading(WebView view, String url) {
```

```
        view.loadUrl(url);
        return true;
      }
    }
}
```

- Following is the modified content of the xml **res/layout/activity_main.xml**.
- In the following code **abc** indicates the logo of tutorial.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">

  <TextView android:text="WebView" 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" />

  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:hint="Enter Text"
    android:focusable="true"
    android:textColorHighlight="#ff7eff15"
    android:textColorHint="#ffff25e6"
    android:layout_marginTop="46dp"
    android:layout_below="@+id/imageView"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_alignRight="@+id/imageView"
    android:layout_alignEnd="@+id/imageView" />

  <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" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Enter"
    android:id="@+id/button"
    android:layout_alignTop="@+id/editText"
    android:layout_toRightOf="@+id/imageView"
    android:layout_toEndOf="@+id/imageView" />

  <WebView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/webView"
    android:layout_below="@+id/button"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:layout_alignParentBottom="true" />

</RelativeLayout>
```

- Following is the content of the **res/values/string.xml**.

```
<resources>
  <string name="app_name">My Application</string>
</resources>
```

- Following is the content of **AndroidManifest.xml** file.

```
<?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.INTERNET" />
  <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 WebView application. To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Android studio will display as shown below
- Now just specify a url on the url field and press the browse button that appears,to launch the website. But before that please make sure that you are connected to the internet. After pressing the button, the following screen would appear –
- Note. By just changing the url in the url field, your WebView will open your desired website.

- **Notifications**
- A **notification** is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area. To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

- Android **Toast** class provides a handy way to show users alerts but problem is that these alerts are not persistent which means alert flashes on the screen for a few seconds and then disappears.

-
- To see the details of the notification, you will have to select the icon which will display notification drawer having detail about the notification. While working with emulator with

virtual device, you will have to click and drag down the status bar to expand it which will give you detail as follows. This will be just **64 dp** tall and called normal view.

- Create and Send Notifications
- You have simple way to create a notification. Follow the following steps in your application to create a notification −

Step 1 - Create Notification Builder

As a first step is to create a notification builder using *NotificationCompat.Builder.build()*. You will use Notification Builder to set various Notification properties like its small and large icons, title, priority etc.

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

Step 2 - Setting Notification Properties

Once you have **Builder** object, you can set its Notification properties using Builder object as per your requirement. But this is mandatory to set at least following −

- A small icon, set by **setSmallIcon()**
- A title, set by **setContentTitle()**
- Detail text, set by **setContentText()**

```
mBuilder.setSmallIcon(R.drawable.notification_icon);
mBuilder.setContentTitle("Notification Alert, Click Me!");
mBuilder.setContentText("Hi, This is Android Notification Detail!");
```

You have plenty of optional properties which you can set for your notification. To learn more about them, see the reference documentation for NotificationCompat.Builder.

Step 3 - Attach Actions

This is an optional part and required if you want to attach an action with the notification. An action allows users to go directly from the notification to an **Activity** in your application, where they can look at one or more events or do further work.

The action is defined by a **PendingIntent** containing an **Intent**that starts an Activity in your application. To associate the PendingIntent with a gesture, call the appropriate method of *NotificationCompat.Builder*. For example, if you want to start Activity when the user clicks the notification text in the notification drawer, you add the PendingIntent by calling **setContentIntent()**.

A PendingIntent object helps you to perform an action on your applications behalf, often at a later time, without caring of whether or not your application is running.

We take help of stack builder object which will contain an artificial back stack for the started Activity. This ensures that navigating backward from the Activity leads out of your application to the Home screen.

```
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
```

```
stackBuilder.addParentStack(ResultActivity.class);

// Adds the Intent that starts the Activity to the top of the stack
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
stackBuilder.getPendingIntent(0,PendingIntent.FLAG_UPDATE_CURRENT);
mBuilder.setContentIntent(resultPendingIntent);
```

Step 4 - Issue the notification

Finally, you pass the Notification object to the system by calling NotificationManager.notify() to send your notification. Make sure you call **NotificationCompat.Builder.build()** method on builder object before notifying it. This method combines all of the options that have been set and return a new **Notification**object.

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

// notificationID allows you to update the notification later on.
mNotificationManager.notify(notificationID, mBuilder.build());
```

The NotificationCompat.Builder Class

The NotificationCompat.Builder class allows easier control over all the flags, as well as help constructing the typical notification layouts. Following are few important and most frequently used methods available as a part of NotificationCompat.Builder class.

Sr.No.	Constants & Description
1	**Notification build()** Combine all of the options that have been set and return a new Notification object.
2	**NotificationCompat.Builder setAutoCancel (boolean autoCancel)** Setting this flag will make it so the notification is automatically canceled when the user clicks it in the panel.
3	**NotificationCompat.Builder setContent (RemoteViews views)** Supply a custom RemoteViews to use instead of the standard one.
4	**NotificationCompat.Builder setContentInfo (CharSequence info)** Set the large text at the right-hand side of the notification.
5	**NotificationCompat.Builder setContentIntent (PendingIntent intent)** Supply a PendingIntent to send when the notification is clicked.
6	**NotificationCompat.Builder setContentText (CharSequence text)** Set the text (second row) of the notification, in a standard notification.
7	**NotificationCompat.Builder setContentTitle (CharSequence title)** Set the text (first row) of the notification, in a standard notification.
8	**NotificationCompat.Builder setDefaults (int defaults)** Set the default notification options that will be used.
9	**NotificationCompat.Builder setLargeIcon (Bitmap icon)** Set the large icon that is shown in the ticker and notification.
10	**NotificationCompat.Builder setNumber (int number)** Set the large number at the right-hand side of the notification.
11	**NotificationCompat.Builder setOngoing (boolean ongoing)** Set whether this is an ongoing notification.
12	**NotificationCompat.Builder setSmallIcon (int icon)** Set the small icon to use in the notification layouts.

13	**NotificationCompat.Builder setStyle (NotificationCompat.Style style)** Add a rich notification style to be applied at build time.
14	**NotificationCompat.Builder setTicker (CharSequence tickerText)** Set the text that is displayed in the status bar when the notification first arrives.
15	**NotificationCompat.Builder setVibrate (long[] pattern)** Set the vibration pattern to use.
16	**NotificationCompat.Builder setWhen (long when)** Set the time that the event occurred. Notifications in the panel are sorted by this time.

Example

Following example shows the functionality of a Android notification using a **NotificationCompat.Builder** Class which has been introduced in Android 4.1.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *tutorial* under a package *com.example.notificationdemo*.
2	Modify *src/MainActivity.java* file and add the code to notify(""), if user click on the button,it will call android notification service.
3	Create a new Java file *src/NotificationView.java*, which will be used to display new layout as a part of new activity which will be started when user will click any of the notifications
4	Modify layout XML file *res/layout/activity_main.xml* to add Notification button in relative layout.
5	Create a new layout XML file *res/layout/notification.xml*. This will be used as layout file for new activity which will start when user will click any of the notifications.
6	No need to change default string constants. Android studio takes care of default string constants
7	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

```
package com.example.notificationdemo;

import android.app.Activity;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.support.v4.app.NotificationCompat;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity {
  Button b1;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```
    b1 = (Button)findViewById(R.id.button);
    b1.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        addNotification();
      }
    });
  }

  private void addNotification() {
    NotificationCompat.Builder builder =
      new NotificationCompat.Builder(this)
      .setSmallIcon(R.drawable.abc)
      .setContentTitle("Notifications Example")
      .setContentText("This is a test notification");

    Intent notificationIntent = new Intent(this, MainActivity.class);
    PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notificationIntent,
      PendingIntent.FLAG_UPDATE_CURRENT);
    builder.setContentIntent(contentIntent);

    // Add as notification
    NotificationManager manager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
    manager.notify(0, builder.build());
  }
}
```

Following will be the content of **res/layout/notification.xml**file −

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

  <TextView
    android:layout_width="fill_parent"
    android:layout_height="400dp"
    android:text="Hi, Your Detailed notification view goes here...." />
</LinearLayout>
```

- Following is the content of the modified main activity file **src/com.example.notificationdemo/NotificationView.java**.

```
package com.example.notificationdemo;

import android.os.Bundle;
import android.app.Activity;

public class NotificationView extends Activity{
  @Override
```

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

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

```
<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/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Notification Example"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="30dp" />

  <TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials point "
    android:textColor="#ff87ff09"
    android:textSize="30dp"
    android:layout_below="@+id/textView1"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="48dp" />

  <ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_below="@+id/textView2"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="42dp" />

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

```
    android:text="Notification"
    android:id="@+id/button"
    android:layout_marginTop="62dp"
    android:layout_below="@+id/imageButton"
    android:layout_centerHorizontal="true" />

</RelativeLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="action_settings">Settings</string>
  <string name="app_name">tutorial </string>
</resources>
```

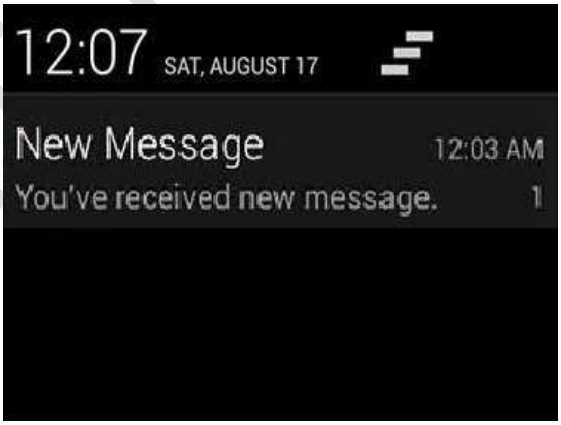
- Following is the default content of **AndroidManifest.xml** −

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

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.notificationdemo.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>

    <activity android:name=".NotificationView"
      android:label="Details of notification"
      android:parentActivityName=".MainActivity">
      <meta-data
      android:name="android.support.PARENT_ACTIVITY"
      android:value=".MainActivity"/>
    </activity>

  </application>
</manifest>
```

- Let's try to run your **tutorial** application. I assume you had created your **AVD** while doing environment set-up. To run the APP from Android Studio, open one of your project's activity files and click Run icon from the toolbar. Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −

- Now click **button**, you will see at the top a message "New Message Alert!" will display momentarily and after that you will have following screen having a small icon at the top left corner.

- Now lets expand the view, long click on the small icon, after a second it will display date information and this is the time when you should drag status bar down without releasing mouse. You will see status bar will expand and you will get following screen −



- ## Storage Options
- Android provides several options for you to save persistent application data. The solution you choose depends on your specific needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires.

- Your data storage options are the following:

- Shared Preferences
- Store private primitive data in key-value pairs.
- Internal Storage
- Store private data on the device memory.
- External Storage
- Store public data on the shared external storage.
- SQLite Databases
- Store structured data in a private database.
- Network Connection
- Store data on the web with your own network server.

Internal Storage

- Android provides many kinds of storage for applications to store their data. These storage places are shared preferences, internal and external storage, SQLite storage, and storage via network connection.
- In this chapter we are going to look at the internal storage. Internal storage is the storage of the private data on the device memory.
- By default these files are private and are accessed by only your application and get deleted , when user delete your application.
- Writing file
- In order to use internal storage to write some data in the file, call the openFileOutput() method with the name of the file and the mode. The mode could be private , public e.t.c. Its syntax is given below −

```
FileOutputStream fOut = openFileOutput("file name here",MODE_WORLD_READABLE);
```

- The method openFileOutput() returns an instance of FileOutputStream. So you receive it in the object of FileInputStream. After that you can call write method to write data on the file. Its syntax is given below −

```
String str = "data";
fOut.write(str.getBytes());
fOut.close();
```

Reading file

- In order to read from the file you just created , call the openFileInput() method with the name of the file. It returns an instance of FileInputStream. Its syntax is given below −

```
FileInputStream fin = openFileInput(file);
```

- After that, you can call read method to read one character at a time from the file and then you can print it. Its syntax is given below −

```
int c;
String temp="";
while( (c = fin.read()) != -1){
   temp = temp + Character.toString((char)c);
}
```

```
//string temp contains all the data of the file.
fin.close();
```

- Apart from the the methods of write and close, there are other methods provided by the **FileOutputStream** class for better writing files. These methods are listed below −

Sr.No	Method & description
1	**FileOutputStream(File file, boolean append)** This method constructs a new FileOutputStream that writes to file.
2	**getChannel()** This method returns a write-only FileChannel that shares its position with this stream
3	**getFD()** This method returns the underlying file descriptor
4	**write(byte[] buffer, int byteOffset, int byteCount)** This method Writes count bytes from the byte array buffer starting at position offset to this stream

- Apart from the the methods of read and close, there are other methods provided by the **FileInputStream** class for better reading files. These methods are listed below −

Sr.No	Method & description
1	**available()** This method returns an estimated number of bytes that can be read or skipped without blocking for more input
2	**getChannel()** This method returns a read-only FileChannel that shares its position with this stream
3	**getFD()** This method returns the underlying file descriptor
4	**read(byte[] buffer, int byteOffset, int byteCount)** This method reads at most length bytes from this stream and stores them in the byte array b starting at offset

- **Example**
- Here is an example demonstrating the use of internal storage to store and read files. It creates a basic storage application that allows you to read and write from internal storage.
- To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Android Studio IDE to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components
4	Run the application and choose a running android device and install the application on it and verify the results

- Following is the content of the modified main activity file **src/MainActivity.java**.

```
package com.example.sairamkrishna.myapplication;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
```

```
import android.widget.TextView;
import android.widget.Toast;
import java.io.FileInputStream;
import java.io.FileOutputStream;

public class MainActivity extends Activity  {
  Button b1,b2;
  TextView tv;
  EditText ed1;

  String data;
  private String file = "mydata";

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

    b1=(Button)findViewById(R.id.button);
    b2=(Button)findViewById(R.id.button2);

    ed1=(EditText)findViewById(R.id.editText);
    tv=(TextView)findViewById(R.id.textView2);
    b1.setOnClickListener(new View.OnClickListener() {

      @Override
      public void onClick(View v) {
        data=ed1.getText().toString();
        try {
          FileOutputStream fOut = openFileOutput(file,MODE_WORLD_READABLE);
          fOut.write(data.getBytes());
          fOut.close();
          Toast.makeText(getBaseContext(),"file saved",Toast.LENGTH_SHORT).show();
        }
        catch (Exception e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
        }
      }
    });

    b2.setOnClickListener(new View.OnClickListener() {

      @Override
      public void onClick(View v) {
        try {
          FileInputStream fin = openFileInput(file);
          int c;
          String temp="";
          while( (c = fin.read()) != -1){
            temp = temp + Character.toString((char)c);
          }
          tv.setText(temp);
          Toast.makeText(getBaseContext(),"file read",Toast.LENGTH_SHORT).show();
        }
```

```
            catch(Exception e){
            }
          }
        });
    }
}
```

- Following is the modified content of the xml **res/layout/activity_main.xml**.
- In the following code **abc** indicates the logo of tutorial.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">

  <TextView android:text="Internal storage" 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" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Save"
    android:id="@+id/button"
    android:layout_alignParentBottom="true"
    android:layout_alignLeft="@+id/textView"
    android:layout_alignStart="@+id/textView" />

  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:hint="Enter Text"
    android:focusable="true"
    android:textColorHighlight="#ff7eff15"
    android:textColorHint="#ffff25e6"
    android:layout_below="@+id/imageView"
    android:layout_alignRight="@+id/textView"
```

```
    android:layout_alignEnd="@+id/textView"
    android:layout_marginTop="42dp"
    android:layout_alignLeft="@+id/imageView"
    android:layout_alignStart="@+id/imageView" />

  <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" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="load"
    android:id="@+id/button2"
    android:layout_alignTop="@+id/button"
    android:layout_alignRight="@+id/editText"
    android:layout_alignEnd="@+id/editText" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Read"
    android:id="@+id/textView2"
    android:layout_below="@+id/editText"
    android:layout_toLeftOf="@+id/button2"
    android:layout_toStartOf="@+id/button2"
    android:textColor="#ff5bff1f"
    android:textSize="25dp" />

</RelativeLayout>
```

- Following is the content of the **res/values/string.xml**.

```
<resources>
  <string name="app_name">My Application</string>
</resources>
```

- Following is the content of **AndroidManifest.xml** file.

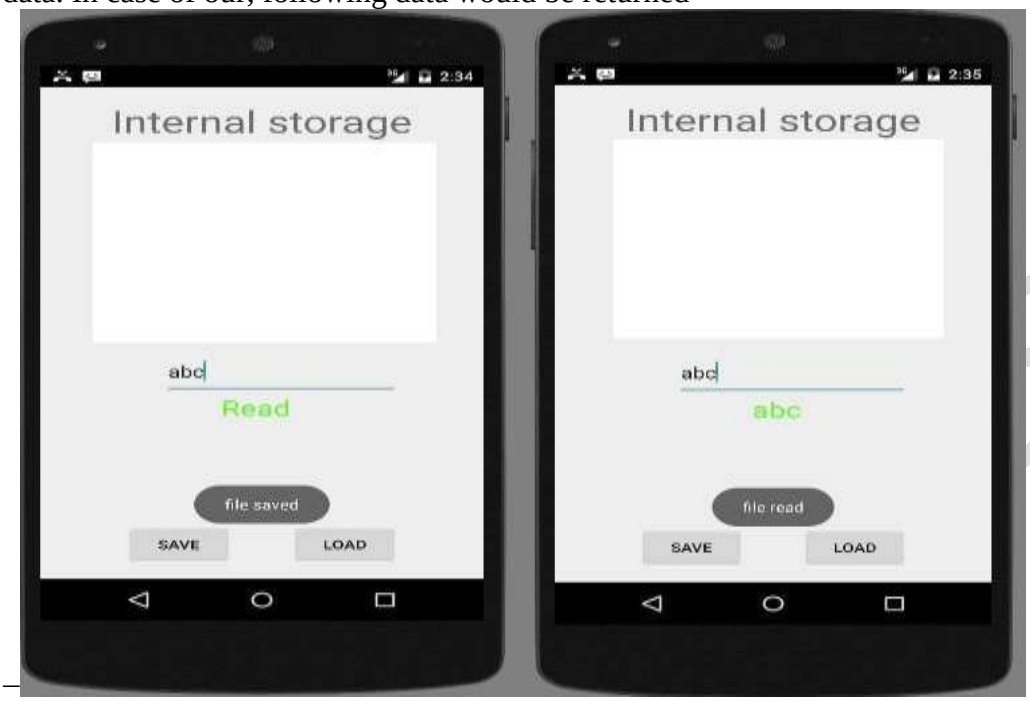
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.sairamkrishna.myapplication" >
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme"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

    <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 our Storage application we just modified. I assume you had created your **AVD** while doing environment setup. To run the app from Android studio, open one of your project's activity files and click Run icon from the tool bar. Android studio installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display following Emulator window −

- Now what you need to do is to enter any text in the field. For example , i have entered some text. Press the save button. The following notification would appear in you AVD −

- Now when you press the load button, the application will read the file , and display the data. In case of our, following data would be returned



- Example of reading and writing data in the android external storage

activity_main.xml

- Drag the 2 edittexts, 2 textviews and 2 buttons from the pallete, now the activity_main.xml file will like this:

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" >

   <EditText
      android:id="@+id/editText1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentRight="true"
      android:layout_alignParentTop="true"
      android:layout_marginRight="20dp"
      android:layout_marginTop="24dp"
      android:ems="10" >

      <requestFocus />
   </EditText>

   <EditText
```

```
    android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@+id/editText1"
    android:layout_below="@+id/editText1"
    android:layout_marginTop="24dp"
    android:ems="10" />

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/editText1"
    android:layout_alignBottom="@+id/editText1"
    android:layout_alignParentLeft="true"
    android:text="File Name:" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/editText2"
    android:layout_alignBottom="@+id/editText2"
    android:layout_alignParentLeft="true"
    android:text="Data:" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText2"
    android:layout_below="@+id/editText2"
    android:layout_marginLeft="70dp"
    android:layout_marginTop="16dp"
    android:text="save" />

<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/button1"
    android:layout_alignBottom="@+id/button1"
    android:layout_toRightOf="@+id/button1"
    android:text="read" />

</RelativeLayout>
```

- *Provide permission for the external storage*
- You need to provide the WRITE_EXTERNAL_STORAGE permission.

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

File: Activity_Manifest.xml

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

xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.externalstorage"
   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk
      android:minSdkVersion="8"
      android:targetSdkVersion="16" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name="com.example.externalstorage.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

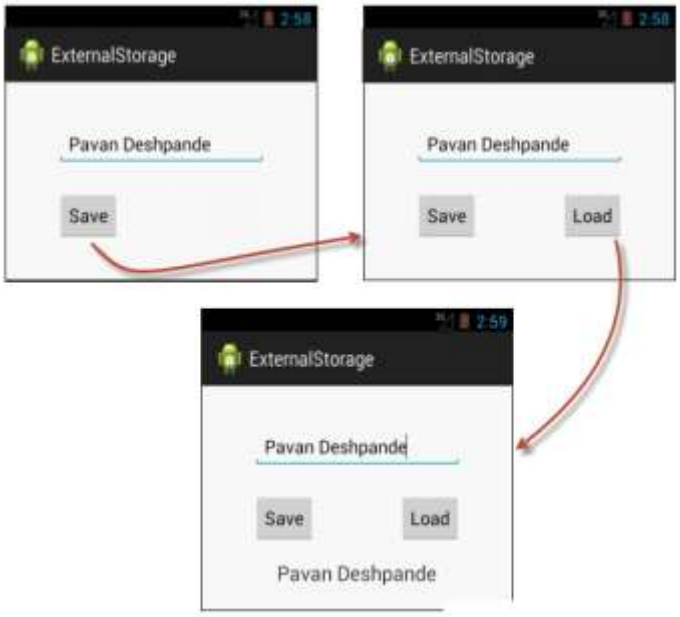
- Let's write the code to write and read data from the android external storage.

File: MainActivity.java

```
package com.example.externalstorage;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
```

```
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {
   EditText editTextFileName,editTextData;
   Button saveButton,readButton;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      editTextFileName=(EditText)findViewById(R.id.editText1);
      editTextData=(EditText)findViewById(R.id.editText2);
      saveButton=(Button)findViewById(R.id.button1);
      readButton=(Button)findViewById(R.id.button2);

      //Performing action on save button
      saveButton.setOnClickListener(new OnClickListener(){

         @Override
         public void onClick(View arg0) {
            String filename=editTextFileName.getText().toString();
            String data=editTextData.getText().toString();

            FileOutputStream fos;
              try {
                 File myFile = new File("/sdcard/"+filename);
                  myFile.createNewFile();
                  FileOutputStream fOut = new

FileOutputStream(myFile);
                  OutputStreamWriter myOutWriter = new

OutputStreamWriter(fOut);
                  myOutWriter.append(data);
                  myOutWriter.close();
                  fOut.close();

         Toast.makeText(getApplicationContext(),filename + "

saved",Toast.LENGTH_LONG).show();
```

```
            } catch (FileNotFoundException e) {e.printStackTrace();}
            catch (IOException e) {e.printStackTrace();}

        }

    });

    //Performing action on Read Button
    readButton.setOnClickListener(new OnClickListener(){

        @Override
        public void onClick(View arg0) {
            String filename=editTextFileName.getText().toString();
            StringBuffer stringBuffer = new StringBuffer();
            String aDataRow = "";
            String aBuffer = "";
            try {
                File myFile = new File("/sdcard/"+filename);
                FileInputStream fIn = new FileInputStream(myFile);
                BufferedReader myReader = new BufferedReader(
                        new InputStreamReader(fIn));

                while ((aDataRow = myReader.readLine()) != null) {
                    aBuffer += aDataRow + "\n";
                }
                myReader.close();

            } catch (IOException e) {
                e.printStackTrace();
            }
            Toast.makeText(getApplicationContext

(),aBuffer,Toast.LENGTH_LONG).show();

        }

    });
  }

  @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;
  }

}
```



- SQLite Database
- In this tutorial I am taking an example of storing user contacts in SQLite database. I am using a table called Contacts to store user contacts. This table contains three columns **id (INT)**, **name (TEXT)**, **phone_number(TEXT)**.

Contacts Table Structure

```
Table Structure                                    AndroidHive

    Table Name: Contacts

    +-------------------+---------------+------+
    | Field             | Type          | Key  |
    +-------------------+---------------+------+
    | id                | INT           | PRI  |
    | name              | TEXT          |      |
    | phone_number      | TEXT          |      |
    +-------------------+---------------+------+
```

- **Writing Contact Class**
- Before you go further you need to write your Contact class with all getter and setter methods to maintain single contact as an object.

Contact.java

```
package com.androidhive.androidsqlite;

public class Contact {

    //private variables
    int _id;
    String _name;
    String _phone_number;

    // Empty constructor
    public Contact(){
```

```
    }
    // constructor
    public Contact(int id, String name, String _phone_number){
        this._id = id;
        this._name = name;
        this._phone_number = _phone_number;
    }

    // constructor
    public Contact(String name, String _phone_number){
        this._name = name;
        this._phone_number = _phone_number;
    }
    // getting ID
    public int getID(){
        return this._id;
    }

    // setting id
    public void setID(int id){
        this._id = id;
    }

    // getting name
    public String getName(){
        return this._name;
    }

    // setting name
    public void setName(String name){
        this._name = name;
    }

    // getting phone number
    public String getPhoneNumber(){
        return this._phone_number;
    }

    // setting phone number
    public void setPhoneNumber(String phone_number){
        this._phone_number = phone_number;
    }
}
```

- **Writing SQLite Database Handler Class**
- We need to write our own class to handle all database CRUD(Create, Read, Update and Delete) operations.

1. Create a new project by going to **File ⇒ New Android Project**.
2. Once the project is created, create a new class in your project src directory and name it as **DatabaseHandler.java** (**Right Click on src/package ⇒ New ⇒ Class**)
3. Now extend your DatabaseHandler.java class from **SQLiteOpenHelper**.

public class DatabaseHandler extends SQLiteOpenHelper {

4. After extending your class from SQLiteOpenHelper you need to override two methods **onCreate()** and **onUpgrage()**

onCreate() – These is where we need to write create table statements. This is called when database is created.

onUpgrade() – This method is called when database is upgraded like modifying the table structure, adding constraints to database etc.,

```
public class DatabaseHandler extends SQLiteOpenHelper {

    // All Static variables
    // Database Version
    private static final int DATABASE_VERSION = 1;

    // Database Name
    private static final String DATABASE_NAME = "contactsManager";

    // Contacts table name
    private static final String TABLE_CONTACTS = "contacts";

    // Contacts Table Columns names
    private static final String KEY_ID = "id";
    private static final String KEY_NAME = "name";
    private static final String KEY_PH_NO = "phone_number";

    public DatabaseHandler(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    // Creating Tables
    @Override
    public void onCreate(SQLiteDatabase db) {
        String CREATE_CONTACTS_TABLE = "CREATE TABLE " + TABLE_CONTACTS + "("
                + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NAME + " TEXT,"
                + KEY_PH_NO + " TEXT" + ")";
        db.execSQL(CREATE_CONTACTS_TABLE);
    }

    // Upgrading database
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Drop older table if existed
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTACTS);

        // Create tables again
        onCreate(db);
    }
```

⇒All CRUD Operations (Create, Read, Update and Delete)

Now we need to write methods for handling all database read and write operations. Here we are implementing following methods for our contacts table.

```
// Adding new contact
public void addContact(Contact contact) {}

// Getting single contact
public Contact getContact(int id) {}

// Getting All Contacts
public List<Contact> getAllContacts() {}

// Getting contacts Count
public int getContactsCount() {}
// Updating single contact
public int updateContact(Contact contact) {}

// Deleting single contact
public void deleteContact(Contact contact) {}
```

⇒Inserting new Record

The **addContact()** method accepts Contact object as parameter. We need to build ContentValues parameters using Contact object. Once we inserted data in database we need to close the database connection.

```
addContact()
   // Adding new contact
public void addContact(Contact contact) {
   SQLiteDatabase db = this.getWritableDatabase();

   ContentValues values = new ContentValues();
   values.put(KEY_NAME, contact.getName()); // Contact Name
   values.put(KEY_PH_NO, contact.getPhoneNumber()); // Contact Phone Number

   // Inserting Row
   db.insert(TABLE_CONTACTS, null, values);
   db.close(); // Closing database connection
}
```

⇒Reading Row(s)

The following method **getContact()** will read single contact row. It accepts id as parameter and will return the matched row from the database.

```
getContact()
   // Getting single contact
public Contact getContact(int id) {
   SQLiteDatabase db = this.getReadableDatabase();

   Cursor cursor = db.query(TABLE_CONTACTS, new String[] { KEY_ID,
         KEY_NAME, KEY_PH_NO }, KEY_ID + "=?",
         new String[] { String.valueOf(id) }, null, null, null, null);
   if (cursor != null)
      cursor.moveToFirst();

   Contact contact = new Contact(Integer.parseInt(cursor.getString(0)),
         cursor.getString(1), cursor.getString(2));
```

```
    // return contact
    return contact;
}
```

getAllContacts() will return all contacts from database in array list format of Contact class type. You need to write a for loop to go through each contact.

```
getAllContacts()
    // Getting All Contacts
 public List<Contact> getAllContacts() {
    List<Contact> contactList = new ArrayList<Contact>();
    // Select All Query
    String selectQuery = "SELECT  * FROM " + TABLE_CONTACTS;

    SQLiteDatabase db = this.getWritableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);

    // looping through all rows and adding to list
    if (cursor.moveToFirst()) {
        do {
            Contact contact = new Contact();
            contact.setID(Integer.parseInt(cursor.getString(0)));
            contact.setName(cursor.getString(1));
            contact.setPhoneNumber(cursor.getString(2));
            // Adding contact to list
            contactList.add(contact);
        } while (cursor.moveToNext());
    }

    // return contact list
    return contactList;
}
```

getContactsCount() will return total number of contacts in SQLite database.

```
getContactsCount()
// Getting contacts Count
    public int getContactsCount() {
        String countQuery = "SELECT  * FROM " + TABLE_CONTACTS;
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(countQuery, null);
        cursor.close();

        // return count
        return cursor.getCount();
    }
```

⇒Updating Record

updateContact() will update single contact in database. This method accepts Contact class object as parameter.

```
updateContact()
    // Updating single contact
public int updateContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();
```

```
    ContentValues values = new ContentValues();
    values.put(KEY_NAME, contact.getName());
    values.put(KEY_PH_NO, contact.getPhoneNumber());

    // updating row
    return db.update(TABLE_CONTACTS, values, KEY_ID + " = ?",
            new String[] { String.valueOf(contact.getID()) });
}
```

⇒Deleting Record

deleteContact() will delete single contact from database.

Writing Contact Class

Before you go further you need to write your Contact class with all getter and setter methods to maintain single contact as an object.

Contact.java

```
package com.androidhive.androidsqlite;

public class Contact {

    //private variables
    int _id;
    String _name;
    String _phone_number;

    // Empty constructor
    public Contact(){

    }
    // constructor
    public Contact(int id, String name, String _phone_number){
        this._id = id;
        this._name = name;
        this._phone_number = _phone_number;
    }

    // constructor
    public Contact(String name, String _phone_number){
        this._name = name;
        this._phone_number = _phone_number;
    }
    // getting ID
    public int getID(){
        return this._id;
    }

    // setting id
    public void setID(int id){
        this._id = id;
```

```
    }

    // getting name
    public String getName(){
        return this._name;
    }

    // setting name
    public void setName(String name){
        this._name = name;
    }

    // getting phone number
    public String getPhoneNumber(){
        return this._phone_number;
    }

    // setting phone number
    public void setPhoneNumber(String phone_number){
        this._phone_number = phone_number;
    }
}
```

- **Writing SQLite Database Handler Class**
- We need to write our own class to handle all database CRUD(Create, Read, Update and Delete) operations.

1. Create a new project by going to **File ⇒ New Android Project**.
2. Once the project is created, create a new class in your project src directory and name it as **DatabaseHandler.java** (**Right Click on src/package ⇒ New ⇒ Class**)
3. Now extend your DatabaseHandler.java class from **SQLiteOpenHelper**.

```
public class DatabaseHandler extends SQLiteOpenHelper {
```

4. After extending your class from SQLiteOpenHelper you need to override two methods **onCreate()** and **onUpgrage()**
onCreate() – These is where we need to write create table statements. This is called when database is created.
onUpgrade() – This method is called when database is upgraded like modifying the table structure, adding constraints to database etc.,

```
public class DatabaseHandler extends SQLiteOpenHelper {

    // All Static variables
    // Database Version
    private static final int DATABASE_VERSION = 1;

    // Database Name
    private static final String DATABASE_NAME = "contactsManager";

    // Contacts table name
    private static final String TABLE_CONTACTS = "contacts";
```

```
// Contacts Table Columns names
private static final String KEY_ID = "id";
private static final String KEY_NAME = "name";
private static final String KEY_PH_NO = "phone_number";

public DatabaseHandler(Context context) {
   super(context, DATABASE_NAME, null, DATABASE_VERSION);
}

// Creating Tables
@Override
public void onCreate(SQLiteDatabase db) {
   String CREATE_CONTACTS_TABLE = "CREATE TABLE " + TABLE_CONTACTS + "("
         + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NAME + " TEXT,"
         + KEY_PH_NO + " TEXT" + ")";
   db.execSQL(CREATE_CONTACTS_TABLE);
}

// Upgrading database
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
   // Drop older table if existed
   db.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTACTS);

   // Create tables again
   onCreate(db);
}
```

⇒All CRUD Operations (Create, Read, Update and Delete)

- Now we need to write methods for handling all database read and write operations. Here we are implementing following methods for our contacts table.

```
// Adding new contact
public void addContact(Contact contact) {}

// Getting single contact
public Contact getContact(int id) {}

// Getting All Contacts
public List<Contact> getAllContacts() {}

// Getting contacts Count
public int getContactsCount() {}
// Updating single contact
public int updateContact(Contact contact) {}

// Deleting single contact
public void deleteContact(Contact contact) {}
```

⇒Inserting new Record

- The **addContact()** method accepts Contact object as parameter. We need to build ContentValues parameters using Contact object. Once we inserted data in database we need to close the database connection.

```
addContact()
   // Adding new contact
public void addContact(Contact contact) {
   SQLiteDatabase db = this.getWritableDatabase();

   ContentValues values = new ContentValues();
   values.put(KEY_NAME, contact.getName()); // Contact Name
   values.put(KEY_PH_NO, contact.getPhoneNumber()); // Contact Phone Number

   // Inserting Row
   db.insert(TABLE_CONTACTS, null, values);
   db.close(); // Closing database connection
}
```

⇒Reading Row(s)

- The following method **getContact()** will read single contact row. It accepts id as parameter and will return the matched row from the database.

```
getContact()
   // Getting single contact
public Contact getContact(int id) {
   SQLiteDatabase db = this.getReadableDatabase();

   Cursor cursor = db.query(TABLE_CONTACTS, new String[] { KEY_ID,
         KEY_NAME, KEY_PH_NO }, KEY_ID + "=?",
         new String[] { String.valueOf(id) }, null, null, null, null);
   if (cursor != null)
      cursor.moveToFirst();

   Contact contact = new Contact(Integer.parseInt(cursor.getString(0)),
         cursor.getString(1), cursor.getString(2));
   // return contact
   return contact;
}
```

- **getAllContacts()** will return all contacts from database in array list format of Contact class type. You need to write a for loop to go through each contact.

```
getAllContacts()
   // Getting All Contacts
 public List<Contact> getAllContacts() {
   List<Contact> contactList = new ArrayList<Contact>();
   // Select All Query
   String selectQuery = "SELECT  * FROM " + TABLE_CONTACTS;

   SQLiteDatabase db = this.getWritableDatabase();
   Cursor cursor = db.rawQuery(selectQuery, null);
```

```
    // looping through all rows and adding to list
    if (cursor.moveToFirst()) {
        do {
            Contact contact = new Contact();
            contact.setID(Integer.parseInt(cursor.getString(0)));
            contact.setName(cursor.getString(1));
            contact.setPhoneNumber(cursor.getString(2));
            // Adding contact to list
            contactList.add(contact);
        } while (cursor.moveToNext());
    }

    // return contact list
    return contactList;
}
```

- **getContactsCount()** will return total number of contacts in SQLite database.

```
getContactsCount()

// Getting contacts Count
    public int getContactsCount() {
        String countQuery = "SELECT  * FROM " + TABLE_CONTACTS;
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(countQuery, null);
        cursor.close();

        // return count
        return cursor.getCount();
    }
```

⇒Updating Record

- **updateContact()** will update single contact in database. This method accepts Contact class object as parameter.

```
updateContact()

    // Updating single contact
public int updateContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(KEY_NAME, contact.getName());
    values.put(KEY_PH_NO, contact.getPhoneNumber());

    // updating row
    return db.update(TABLE_CONTACTS, values, KEY_ID + " = ?",
            new String[] { String.valueOf(contact.getID()) });
}
```

⇒Deleting Record

- **deleteContact()** will delete single contact from database.

UNIT 3

- Checking Network Availability
- A device can have various types of network connections. This chapter focuses on using either a Wi-Fi or a mobile network connection.

- Permission to access the network
- In order to perform network operations in your application, your manifest must include the following permissions:

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

- Checking Network Connection
- Before you perform any network operations, you must first check that are you connected to that network or internet e.t.c. For this android provides **ConnectivityManager** class. You need to instantiate an object of this class by calling **getSystemService()** method. Its syntax is given below −

```
ConnectivityManager check = (ConnectivityManager)
this.context.getSystemService(Context.CONNECTIVITY_SERVICE);
```

- Once you instantiate the object of ConnectivityManager class, you can use **getAllNetworkInfo** method to get the information of all the networks. This method returns an array of **NetworkInfo**. So you have to receive it like this.

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

- The last thing you need to do is to check **Connected State** of the network. Its syntax is given below −

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

- Apart from this connected states, there are other states a network can achieve. They are listed below −

Sr.No	State
1	Connecting
2	Disconnected
3	Disconnecting
4	Suspended
5	Unknown

- Performing Network Operations
- After checking that you are connected to the internet, you can perform any network operation. Here we are fetching the html of a website from a url.

- Android provides **HttpURLConnection** and **URL** class to handle these operations. You need to instantiate an object of URL class by providing the link of website. Its syntax is as follows −

```
String link = "http://www.google.com";
URL url = new URL(link);
```

- After that you need to call **openConnection** method of url class and receive it in a HttpURLConnection object. After that you need to call the **connect** method of HttpURLConnection class.

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.connect();
```

- And the last thing you need to do is to fetch the HTML from the website. For this you will use **InputStream** and **BufferedReader** class. Its syntax is given below –

```
InputStream is = conn.getInputStream();
BufferedReader reader = new BufferedReader(new InputStreamReader(is, "UTF-8"));
String webPage = "",data="";

while ((data = reader.readLine()) != null){
  webPage += data + "\n";
}
```

- Apart from this connect method, there are other methods available in HttpURLConnection class. They are listed below −

Sr.No	Method & description
1	**disconnect()** This method releases this connection so that its resources may be either reused or closed
2	**getRequestMethod()** This method returns the request method which will be used to make the request to the remote HTTP server
3	**getResponseCode()** This method returns response code returned by the remote HTTP server
4	**setRequestMethod(String method)** This method Sets the request command which will be sent to the remote HTTP server
5	**usingProxy()** This method returns whether this connection uses a proxy server or not

- Example

- The below example demonstrates the use of HttpURLConnection class. It creates a basic application that allows you to download HTML from a given web page.
- To experiment with this example, you need to run this on an actual device on which wifi internet is connected .

Steps	Description
1	You will use Android studio IDE to create an Android application under a package com.tutorial.myapplication.
2	Modify src/MainActivity.java file to add Activity code.
4	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
6	Modify AndroidManifest.xml to add necessary permissions.
7	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.tutorial.myapplication;

import android.app.ProgressDialog;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;

import android.net.ConnectivityManager;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.support.v7.app.ActionBarActivity;
import android.view.View;

import android.widget.Button;
import android.widget.ImageView;
import android.widget.Toast;

import java.io.IOException;
import java.io.InputStream;

import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;

public class MainActivity extends ActionBarActivity {
  private ProgressDialog progressDialog;
  private Bitmap bitmap = null;
  Button b1;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    b1 = (Button) findViewById(R.id.button);

    b1.setOnClickListener(new View.OnClickListener() {
      @Override
```

```
    public void onClick(View v) {
      checkInternetConenction();
      downloadImage("http://www.tutorial.com/green/images/logo.png");
    }
  });
}

private void downloadImage(String urlStr) {
  progressDialog = ProgressDialog.show(this, "", "Downloading Image from " + urlStr);
  final String url = urlStr;

  new Thread() {
    public void run() {
      InputStream in = null;

      Message msg = Message.obtain();
      msg.what = 1;

      try {
        in = openHttpConnection(url);
        bitmap = BitmapFactory.decodeStream(in);
        Bundle b = new Bundle();
        b.putParcelable("bitmap", bitmap);
        msg.setData(b);
        in.close();
      }catch (IOException e1) {
        e1.printStackTrace();
      }
      messageHandler.sendMessage(msg);
    }
  }.start();
}

private InputStream openHttpConnection(String urlStr) {
  InputStream in = null;
  int resCode = -1;

  try {
    URL url = new URL(urlStr);
    URLConnection urlConn = url.openConnection();

    if (!(urlConn instanceof HttpURLConnection)) {
      throw new IOException("URL is not an Http URL");
    }

    HttpURLConnection httpConn = (HttpURLConnection) urlConn;
    httpConn.setAllowUserInteraction(false);
    httpConn.setInstanceFollowRedirects(true);
    httpConn.setRequestMethod("GET");
    httpConn.connect();
    resCode = httpConn.getResponseCode();

    if (resCode == HttpURLConnection.HTTP_OK) {
      in = httpConn.getInputStream();
    }
```

```
    }catch (MalformedURLException e) {
      e.printStackTrace();
    }catch (IOException e) {
      e.printStackTrace();
    }
    return in;
  }

  private Handler messageHandler = new Handler() {
    public void handleMessage(Message msg) {
      super.handleMessage(msg);
      ImageView img = (ImageView) findViewById(R.id.imageView);
      img.setImageBitmap((Bitmap) (msg.getData().getParcelable("bitmap")));
      progressDialog.dismiss();
    }
  };

  private boolean checkInternetConenction() {
    // get Connectivity Manager object to check connection
    ConnectivityManager connec

=(ConnectivityManager)getSystemService(getBaseContext().CONNECTIVITY_SERVICE);

    // Check for network connections
    if ( connec.getNetworkInfo(0).getState() ==
      android.net.NetworkInfo.State.CONNECTED ||
      connec.getNetworkInfo(0).getState() ==
      android.net.NetworkInfo.State.CONNECTING ||
      connec.getNetworkInfo(1).getState() ==
      android.net.NetworkInfo.State.CONNECTING ||
      connec.getNetworkInfo(1).getState() == android.net.NetworkInfo.State.CONNECTED )
{
        Toast.makeText(this, " Connected ", Toast.LENGTH_LONG).show();
        return true;
      }else if (
        connec.getNetworkInfo(0).getState() ==
        android.net.NetworkInfo.State.DISCONNECTED ||
        connec.getNetworkInfo(1).getState() ==
        android.net.NetworkInfo.State.DISCONNECTED  ) {
          Toast.makeText(this, " Not Connected ", Toast.LENGTH_LONG).show();
          return false;
        }
      return false;
    }

}
```

- Here is the content of **activity_main.xml**.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
  android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
```

```
android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">

<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="UI Animator Viewer"
  android:id="@+id/textView"
  android:textSize="25sp"
  android:layout_centerHorizontal="true" />

<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Tutorials point"
  android:id="@+id/textView2"
  android:layout_below="@+id/textView"
  android:layout_alignRight="@+id/textView"
  android:layout_alignEnd="@+id/textView"
  android:textColor="#ff36ff15"
  android:textIsSelectable="false"
  android:textSize="35dp" />

<ImageView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/imageView"
  android:layout_below="@+id/textView2"
  android:layout_centerHorizontal="true" />

<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Button"
  android:id="@+id/button"
  android:layout_below="@+id/imageView"
  android:layout_centerHorizontal="true"
  android:layout_marginTop="76dp" />

</RelativeLayout>
```

- Here is the content of **Strings.xml**.

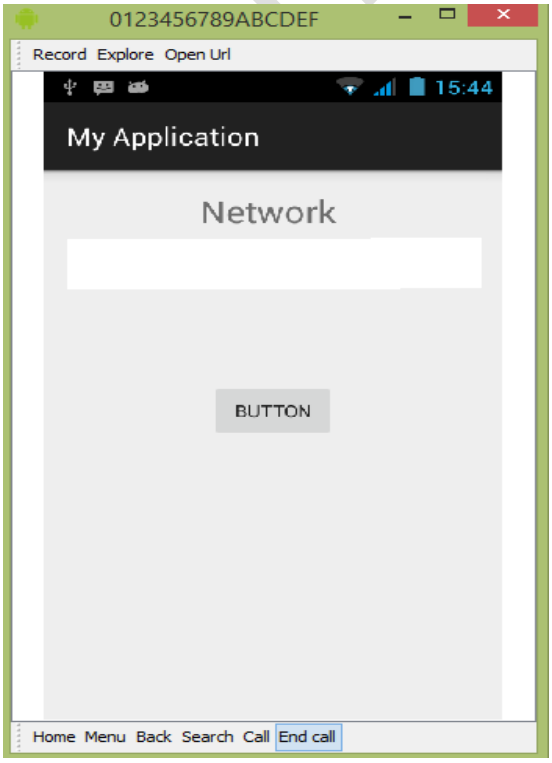
```
<resources>
  <string name="app_name">My Application</string>
</resources>
```

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

```
    <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 tool bar. Before starting your application, Android studio will display following window to select an option where you want to run your Android application.



- Now just click on button, It will check internet connection as well as it will download image.

- Sending Email
- **Email** is messages distributed by electronic means from one system user to one or more recipients via a network.
- Before starting Email Activity, You must know Email functionality with intent, Intent is carrying data from one component to another component with-in the application or outside the application.

- To send an email from your application, you don't have to implement an email client from the beginning, but you can use an existing one like the default Email app provided from Android, Gmail, Outlook, K-9 Mail etc. For this purpose, we need to write an Activity that launches an email client, using an implicit Intent with the right action and data. In this example, we are going to send an email from our app by using an Intent object that launches existing email clients.

- Following section explains different parts of our Intent object required to send an email.

- Intent Object - Action to send Email
- You will use **ACTION_SEND** action to launch an email client installed on your Android device. Following is simple syntax to create an intent with ACTION_SEND action.

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
```

- Intent Object - Data/Type to send Email
- To send an email you need to specify **mailto:** as URI using setData() method and data type will be to **text/plain** using setType() method as follows −

```
emailIntent.setData(Uri.parse("mailto:"));
emailIntent.setType("text/plain");
```

- Intent Object - Extra to send Email
- Android has built-in support to add TO, SUBJECT, CC, TEXT etc. fields which can be attached to the intent before sending the intent to a target email client. You can use following extra fields in your email −

Sr.No.	Extra Data & Description
1	**EXTRA_BCC** A String[] holding e-mail addresses that should be blind carbon copied.
2	**EXTRA_CC** A String[] holding e-mail addresses that should be carbon copied.
3	**EXTRA_EMAIL** A String[] holding e-mail addresses that should be delivered to.
4	**EXTRA_HTML_TEXT** A constant String that is associated with the Intent, used with ACTION_SEND to supply an alternative to EXTRA_TEXT as HTML formatted text.
5	**EXTRA_SUBJECT** A constant string holding the desired subject line of a message.
6	**EXTRA_TEXT** A constant CharSequence that is associated with the Intent, used with ACTION_SEND to supply the literal data to be sent.
7	**EXTRA_TITLE** A CharSequence dialog title to provide to the user when used with a ACTION_CHOOSER.

- Here is an example showing you how to assign extra data to your intent −

```
emailIntent.putExtra(Intent.EXTRA_EMAIL  , new String[]{"Recipient"});
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "subject");
emailIntent.putExtra(Intent.EXTRA_TEXT   , "Message Body");
```

- The out-put of above code is as below shown an image

- ***EMAIL EXAMPLE***
- Example
- Following example shows you in practical how to use Intent object to launch Email client to send an Email to the given recipients.

- To Email experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you might get struggle with emulator which may not work properly. Second you will need to have an Email client like GMail(By default every android version having Gmail client App) or K9mail installed on your device.

Step	Description
1	You will use Android studio to create an Android application and name it as *Tutorial* under a package *com.example.tutorial*.
2	Modify *src/MainActivity.java* file and add required code to take care of sending email.
3	Modify layout XML file *res/layout/activity_main.xml* add any GUI component if required. I'm adding a simple button to launch Email Client.
4	Modify *res/values/strings.xml* to define required constant values
5	Modify *AndroidManifest.xml* as shown below
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

- Following is the content of the modified main activity file **src/com.example.Tutorial/MainActivity.java**.

```
package com.example.tutorial;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```
    Button startBtn = (Button) findViewById(R.id.sendEmail);
    startBtn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        sendEmail();
      }
    });
  }

  protected void sendEmail() {
    Log.i("Send email", "");
    String[] TO = {""};
    String[] CC = {""};
    Intent emailIntent = new Intent(Intent.ACTION_SEND);

    emailIntent.setData(Uri.parse("mailto:"));
    emailIntent.setType("text/plain");
    emailIntent.putExtra(Intent.EXTRA_EMAIL, TO);
    emailIntent.putExtra(Intent.EXTRA_CC, CC);
    emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Your subject");
    emailIntent.putExtra(Intent.EXTRA_TEXT, "Email message goes here");

    try {
      startActivity(Intent.createChooser(emailIntent, "Send mail..."));
      finish();
      Log.i("Finished sending email...", "");
    } catch (android.content.ActivityNotFoundException ex) {
      Toast.makeText(MainActivity.this, "There is no email client installed.",
Toast.LENGTH_SHORT).show();
    }
  }
}
```

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

- Here abc indicates about tutorial logo

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Sending Mail Example"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="30dp" />

  <TextView
```

```
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials point "
    android:textColor="#ff87ff09"
    android:textSize="30dp"
    android:layout_above="@+id/imageButton"
    android:layout_alignRight="@+id/imageButton"
    android:layout_alignEnd="@+id/imageButton" />

  <ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />

  <Button
    android:id="@+id/sendEmail"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/compose_email"/>

</LinearLayout>
```

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

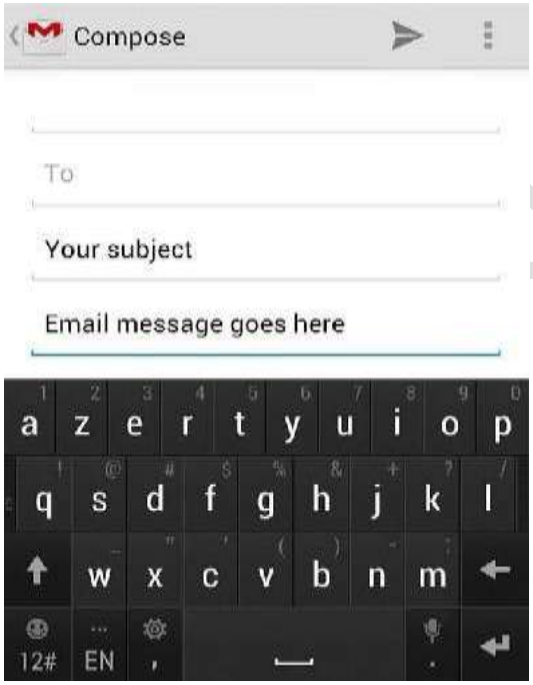
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Tutorial</string>
  <string name="compose_email">Compose Email</string>
</resources>
```

- Following is the default content of **AndroidManifest.xml** −

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

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.tutorial.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>
```

- I'm going to use Gmail client to send my email which will have all the provided defaults fields available as shown below. Here **From:** will be default email ID you have registered for your Android device.



- Displaying Maps
- Android allows us to integrate google maps in our application. You can show any location on the map , or can show different routes on the map e.t.c. You can also customize the map according to your choices.

- 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" />
```

- 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 Tutorial = new LatLng(21 , 57);
Marker TP = googleMap.addMarker(new MarkerOptions()
  .position(Tutorial).title("Tutorial"));
```

- Channing Map Type
- You can also change the type of the MAP. There are four different types of map and each give 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 −

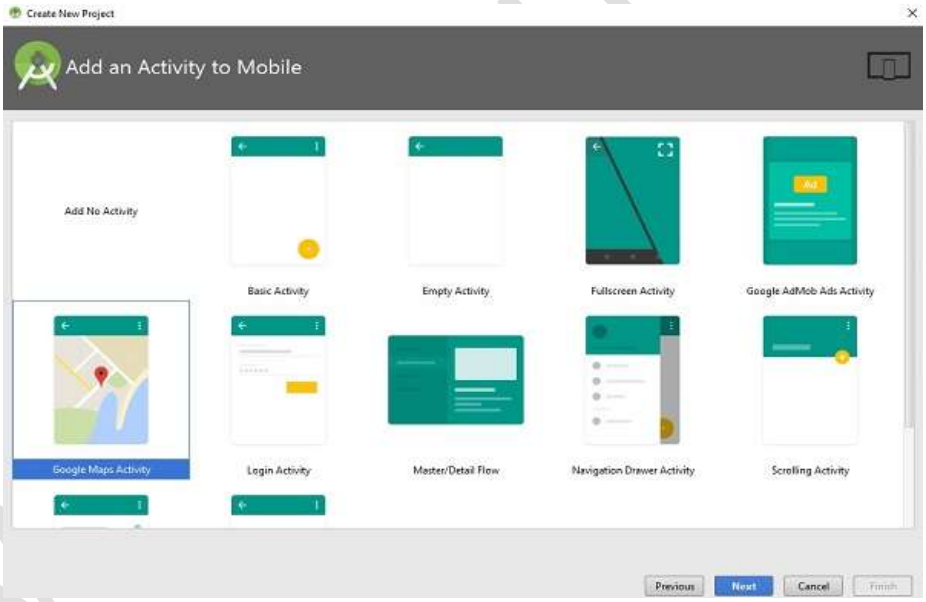
Sr.No	Method & description
1	**addCircle(CircleOptions options)** This method add a circle to the map
2	**addPolygon(PolygonOptions options)** This method add a polygon to the map

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

- **Example**
- Here is an example demonstrating the use of GoogleMap class. It creates a basic M application that allows you to navigate through the map.

- To experiment with this example , you can run this on an actual device or in an emulator.

- Create a project with google maps activity as shown below −



- It will open the following screen and copy the console url for API Key from your screen.
- Copy this and paste it to your browser. It will give the following screen.
- Click on continue and click on Create API Key.
- Here is the content of **activity_main.xml**.

```
<fragment 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:id="@+id/map"
android:name="com.google.android.gms.maps.SupportMapFragment"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="com.example.tutorial.myapplication.MapsActivity" />
```

- Here is the content of **MapActivity.java**.

- In the below code we have given sample latitude and longitude details

```
package com.example.tutorial.myapplication;

import android.support.v4.app.FragmentActivity;
import android.os.Bundle;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {

  private GoogleMap mMap;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_maps);
    // Obtain the SupportMapFragment and get notified when the map is ready to be used.
    SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
        .findFragmentById(R.id.map);
    mapFragment.getMapAsync(this);
  }

  /**
    * Manipulates the map once available.
    * This callback is triggered when the map is ready to be used.
    * This is where we can add markers or lines, add listeners or move the camera.
    * In this case, we just add a marker near Sydney, Australia.
    * If Google Play services is not installed on the device.
    * This method will only be triggered once the user has installed
      Google Play services and returned to the app.
  */

  @Override
  public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    // Add a marker in Sydney and move the camera
```

```
    LatLng Tutorial = new LatLng(21, 57);
    mMap.addMarker(new
      MarkerOptions().position(Tutorial).title("Tutorial.com"));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(Tutorial));
  }
}
```

- Following is the content of **AndroidManifest.xml** file.

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

  <!--
    The ACCESS_COARSE/FINE_LOCATION permissions are not required to use
    Google Maps Android API v2, but you must specify either coarse or fine
    location permissions for the 'MyLocation' functionality.
  -->

  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_COARSE_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">

    <!--
      The API key for Google Maps-based APIs is defined as a string resource.
      (See the file "res/values/google_maps_api.xml").
      Note that the API key is linked to the encryption key used to sign the APK.
      You need a different API key for each encryption key, including the release key
      that is used to sign the APK for publishing.
      You can define the keys for the debug and
        release targets in src/debug/ and src/release/.
    -->

    <meta-data
      android:name="com.google.android.geo.API_KEY"
      android:value="AIzaSyAXhBdyKxUo_cb-EkSgWJQTdqR0QjLcqes" />

    <activity
      android:name=".MapsActivity"
      android:label="@string/title_activity_maps">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
```

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

</manifest>
```

Output should be like this

- monitoring a Location,
- Displaying a Location Address
- Once you have **Location** object, you can use **Geocoder.getFromLocation()** method to get an address for a given latitude and longitude. This method is synchronous, and may take a long time to do its work, so you should call the method from the **doInBackground()** method of an **AsyncTask**class.

- The **AsyncTask** must be subclassed to be used and the subclass will override **doInBackground(Params...)** method to perform a task in the background and **onPostExecute(Result)** method is invoked on the UI thread after the background computation finishes and at the time to display the result. There is one more important method available in AyncTask which is **execute(Params... params)**, this method executes the task with the specified parameters.

- **Example**
- Following example shows you in practical how to to use Location Services in your app to get the current location and its equivalent addresses etc.

- To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

- Create Android Application

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *Tutorial* under

	a package *com.example.tutorial.myapplication*.
2	add *src/GPSTracker.java* file and add required code.
3	Modify *src/MainActivity.java* file and add required code as shown below to take care of getting current location and its equivalent address.
4	Modify layout XML file *res/layout/activity_main.xml* to add all GUI components which include three buttons and two text views to show location/address.
5	Modify *res/values/strings.xml* to define required constant values
6	Modify *AndroidManifest.xml* as shown below
7	Run the application to launch Android emulator and verify the result of the changes done in the application.

- Following is the content of the modified main activity file **MainActivity.java**.

```
package com.example.tutorial.myapplication;

import android.Manifest;
import android.app.Activity;
import android.os.Bundle;
import android.support.v4.app.ActivityCompat;
import android.test.mock.MockPackageManager;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {

  Button btnShowLocation;
  private static final int REQUEST_CODE_PERMISSION = 2;
  String mPermission = Manifest.permission.ACCESS_FINE_LOCATION;

  // GPSTracker class
  GPSTracker gps;

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

    try {
      if (ActivityCompat.checkSelfPermission(this, mPermission)
          != MockPackageManager.PERMISSION_GRANTED) {

        ActivityCompat.requestPermissions(this, new String[]{mPermission},
            REQUEST_CODE_PERMISSION);

        // If any permission above not allowed by user, this condition will
          execute every time, else your else part will work
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
```

```
        btnShowLocation = (Button) findViewById(R.id.button);

        // show location button click event
        btnShowLocation.setOnClickListener(new View.OnClickListener() {

          @Override
          public void onClick(View arg0) {
            // create class object
            gps = new GPSTracker(MainActivity.this);

            // check if GPS enabled
            if(gps.canGetLocation()){

              double latitude = gps.getLatitude();
              double longitude = gps.getLongitude();

              // \n is for new line
              Toast.makeText(getApplicationContext(), "Your Location is - \nLat: "
                + latitude + "\nLong: " + longitude, Toast.LENGTH_LONG).show();
            }else{
              // can't get location
              // GPS or Network is not enabled
              // Ask user to enable GPS/network in settings
              gps.showSettingsAlert();
            }

          }
        });
      }
    }
```

- Following is the content of the modified main activity file **GPSTracker.java**.

```
package com.example.tutorial.myapplication;

import android.app.AlertDialog;
import android.app.Service;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.IBinder;
import android.provider.Settings;
import android.util.Log;

public class GPSTracker extends Service implements LocationListener {
```

```
  private final Context mContext;

  // flag for GPS status
  boolean isGPSEnabled = false;

  // flag for network status
  boolean isNetworkEnabled = false;

  // flag for GPS status
  boolean canGetLocation = false;

  Location location; // location
  double latitude; // latitude
  double longitude; // longitude

  // The minimum distance to change Updates in meters
  private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 10; // 10 meters

  // The minimum time between updates in milliseconds
  private static final long MIN_TIME_BW_UPDATES = 1000 * 60 * 1; // 1 minute

  // Declaring a Location Manager
  protected LocationManager locationManager;

  public GPSTracker(Context context) {
    this.mContext = context;
    getLocation();
  }

  public Location getLocation() {
    try {
      locationManager = (LocationManager)
mContext.getSystemService(LOCATION_SERVICE);

      // getting GPS status
      isGPSEnabled =
locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);

      // getting network status
      isNetworkEnabled = locationManager
        .isProviderEnabled(LocationManager.NETWORK_PROVIDER);

      if (!isGPSEnabled && !isNetworkEnabled) {
        // no network provider is enabled
      } else {
        this.canGetLocation = true;
        // First get location from Network Provider
        if (isNetworkEnabled) {
          locationManager.requestLocationUpdates(
            LocationManager.NETWORK_PROVIDER,
```

```
            MIN_TIME_BW_UPDATES,
            MIN_DISTANCE_CHANGE_FOR_UPDATES, this);

          Log.d("Network", "Network");
          if (locationManager != null) {
            location = locationManager
              .getLastKnownLocation(LocationManager.NETWORK_PROVIDER);

            if (location != null) {
              latitude = location.getLatitude();
              longitude = location.getLongitude();
            }
          }
        }

        // if GPS Enabled get lat/long using GPS Services
        if (isGPSEnabled) {
          if (location == null) {
            locationManager.requestLocationUpdates(
              LocationManager.GPS_PROVIDER,
              MIN_TIME_BW_UPDATES,
              MIN_DISTANCE_CHANGE_FOR_UPDATES, this);

            Log.d("GPS Enabled", "GPS Enabled");
            if (locationManager != null) {
              location = locationManager
                .getLastKnownLocation(LocationManager.GPS_PROVIDER);

              if (location != null) {
                latitude = location.getLatitude();
                longitude = location.getLongitude();
              }
            }
          }
        }

      } catch (Exception e) {
        e.printStackTrace();
      }

      return location;
    }

    /**
      * Stop using GPS listener
      * Calling this function will stop using GPS in your app
    * */

    public void stopUsingGPS(){
```

```
  if(locationManager != null){
    locationManager.removeUpdates(GPSTracker.this);
  }
}

/**
  * Function to get latitude
* */

public double getLatitude(){
  if(location != null){
    latitude = location.getLatitude();
  }

  // return latitude
  return latitude;
}

/**
  * Function to get longitude
* */

public double getLongitude(){
  if(location != null){
    longitude = location.getLongitude();
  }

  // return longitude
  return longitude;
}

/**
  * Function to check GPS/wifi enabled
  * @return boolean
* */

public boolean canGetLocation() {
  return this.canGetLocation;
}

/**
  * Function to show settings alert dialog
  * On pressing Settings button will lauch Settings Options
* */

public void showSettingsAlert(){
  AlertDialog.Builder alertDialog = new AlertDialog.Builder(mContext);

  // Setting Dialog Title
  alertDialog.setTitle("GPS is settings");
```

```
        // Setting Dialog Message
        alertDialog.setMessage("GPS is not enabled. Do you want to go to settings menu?");

        // On pressing Settings button
        alertDialog.setPositiveButton("Settings", new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog,int which) {
            Intent intent = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
            mContext.startActivity(intent);
          }
        });

        // on pressing cancel button
        alertDialog.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int which) {
            dialog.cancel();
          }
        });

        // Showing Alert Message
        alertDialog.show();
      }

      @Override
      public void onLocationChanged(Location location) {
      }

      @Override
      public void onProviderDisabled(String provider) {
      }

      @Override
      public void onProviderEnabled(String provider) {
      }

      @Override
      public void onStatusChanged(String provider, int status, Bundle extras) {
      }

      @Override
      public IBinder onBind(Intent arg0) {
        return null;
      }
    }
```

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

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

```
android:orientation = "vertical" >


<Button
  android:id = "@+id/button"
  android:layout_width = "fill_parent"
  android:layout_height = "wrap_content"
  android:text = "getlocation"/>

</LinearLayout>
```

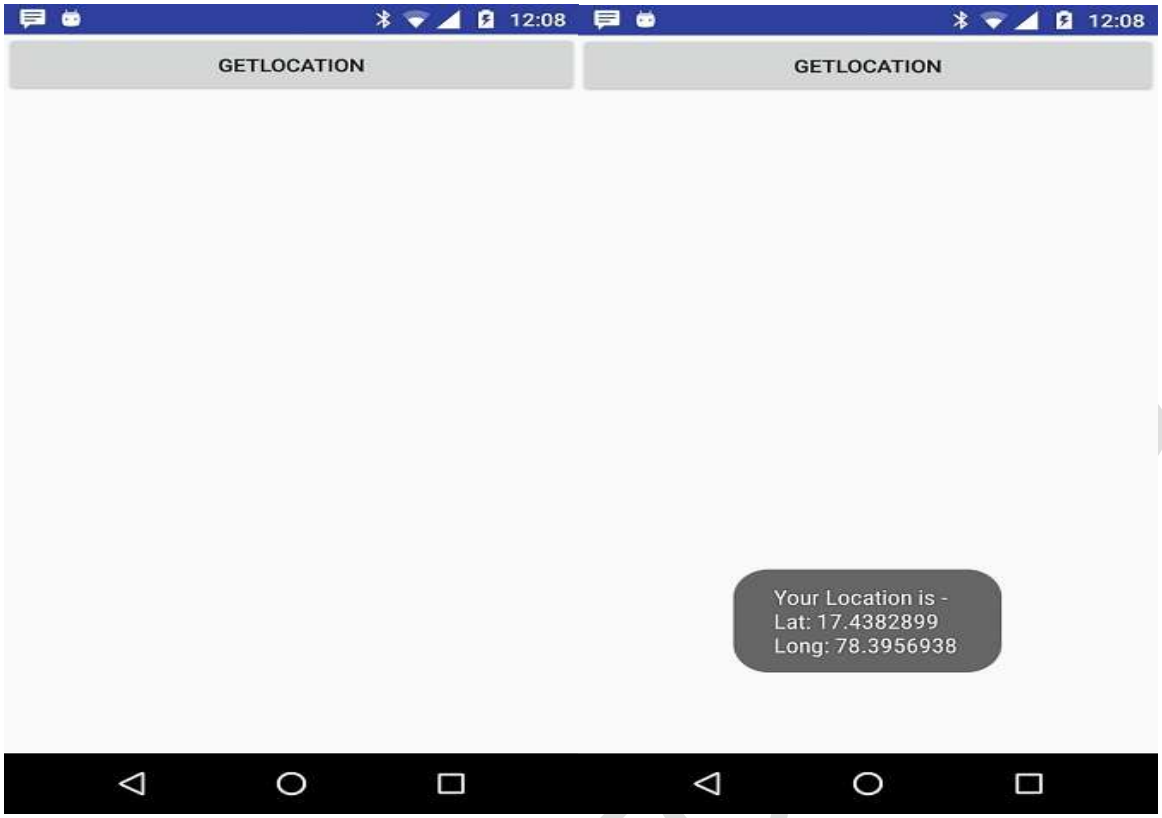
- Following will be the content of **res/values/strings.xml** to define two new constants −

```
<?xml version = "1.0" encoding = "utf-8"?>
<resources>
  <string name = "app_name">Tutorial</string>
</resources>
```

- Following is the default content of **AndroidManifest.xml** −

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

    <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>
```

- Let's try to run your **Tutorial** application. I assume that, 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 installer will display following window to select an option where you want to run your Android application.



- **Geocoder**
- A class for handling geocoding and reverse geocoding. Geocoding is the process of transforming a street address or other description of a location into a (latitude, longitude) coordinate. Reverse geocoding is the process of transforming a (latitude, longitude) coordinate into a (partial) address. The amount of detail in a reverse geocoded location description may vary, for example one might contain the full street address of the closest building, while another might contain only a city name and postal code. The Geocoder class requires a backend service that is not included in the core android framework. The Geocoder query methods will return an empty list if there no backend service in the platform. Use the isPresent() method to determine whether a Geocoder implementation exists.
- Geocoding is the process of converting the addresses (postal address) into geo coordinates as latitude and longitude. Reverse geocoding is converting a geo coordinate latitude and longitude to an address. In this tutorial we will be doing reverse geo coding and get the addresses of the passed coordinates.
- Android Reverse Geocoding Example to find Address

Step1: Define Permissions

We need location access permission to find the latitude and longitude of the Android device. Following two lines are the key to give permission to access the location.

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

So the complete manifest file will be as below:

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

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

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name=".MyActivity"
         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 2: Accessing the Geo Location for Lat and Long**
- Following class is the key element in accessing the latitude and longitude of the Android device. It implements the LocationListener and gets the location coordinate updates. We have designed our requirement not to be a continous update for location. On demand we will get the location coordinates.
- **AppLocationService.java**

```
package android.javapapers.com.androidgeocodelocation;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.IBinder;

public class AppLocationService extends Service implements LocationListener {

   protected LocationManager locationManager;
   Location location;
```

```
    private static final long MIN_DISTANCE_FOR_UPDATE = 10;
    private static final long MIN_TIME_FOR_UPDATE = 1000 * 60 * 2;

    public AppLocationService(Context context) {
       locationManager = (LocationManager) context
             .getSystemService(LOCATION_SERVICE);
    }

    public Location getLocation(String provider) {
       if (locationManager.isProviderEnabled(provider)) {
          locationManager.requestLocationUpdates(provider,
                MIN_TIME_FOR_UPDATE, MIN_DISTANCE_FOR_UPDATE, this);
          if (locationManager != null) {
             location = locationManager.getLastKnownLocation(provider);
             return location;
          }
       }
       return null;
    }

    @Override
    public void onLocationChanged(Location location) {
    }

    @Override
    public void onProviderDisabled(String provider) {
    }

    @Override
    public void onProviderEnabled(String provider) {
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {
    }

    @Override
    public IBinder onBind(Intent arg0) {
       return null;
    }

}
```

Step 3: Reverse Geocoding to get Location Address

- Following class is the key element for reverse geocoding to get the address for the passed latitude and longitude coordinates. We access the GeocoderGoogle API for reverse geocoding and get every line of address like street, city, pin / zip code and etc.

LocationAddress.java

```
package android.javapapers.com.androidgeocodelocation;

import android.content.Context;
import android.location.Address;
import android.location.Geocoder;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;

import java.io.IOException;
import java.util.List;
import java.util.Locale;

public class LocationAddress {
   private static final String TAG = "LocationAddress";

   public static void getAddressFromLocation(final double latitude, final double longitude,
                                             final Context context, final Handler handler) {
      Thread thread = new Thread() {
         @Override
         public void run() {
            Geocoder geocoder = new Geocoder(context, Locale.getDefault());
            String result = null;
            try {
               List<Address> addressList = geocoder.getFromLocation(
                     latitude, longitude, 1);
               if (addressList != null && addressList.size() > 0) {
                  Address address = addressList.get(0);
                  StringBuilder sb = new StringBuilder();
                  for (int i = 0; i < address.getMaxAddressLineIndex(); i++) {
                     sb.append(address.getAddressLine(i)).append("\n");
                  }
                  sb.append(address.getLocality()).append("\n");
                  sb.append(address.getPostalCode()).append("\n");
                  sb.append(address.getCountryName());
                  result = sb.toString();
               }
            } catch (IOException e) {
               Log.e(TAG, "Unable connect to Geocoder", e);
            } finally {
               Message message = Message.obtain();
               message.setTarget(handler);
               if (result != null) {
                  message.what = 1;
                  Bundle bundle = new Bundle();
                  result = "Latitude: " + latitude + " Longitude: " + longitude +
                        "\n\nAddress:\n" + result;
                  bundle.putString("address", result);
                  message.setData(bundle);
```

```
                } else {
                    message.what = 1;
                    Bundle bundle = new Bundle();
                    result = "Latitude: " + latitude + " Longitude: " + longitude +
                            "\n Unable to get address for this lat-long.";
                    bundle.putString("address", result);
                    message.setData(bundle);
                }
                message.sendToTarget();
            }
        }
    };
    thread.start();
  }
}
```

Step 4: Android UI

- How these pieces fit together is with the following Android activity.

MyActivity.java

```
package android.javapapers.com.androidgeocodelocation;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.location.Location;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.provider.Settings;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MyActivity extends Activity {

  Button btnGPSShowLocation;
  Button btnShowAddress;
  TextView tvAddress;

  AppLocationService appLocationService;

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

```
        tvAddress = (TextView) findViewById(R.id.tvAddress);
        appLocationService = new AppLocationService(
                MyActivity.this);

        btnGPSShowLocation = (Button) findViewById(R.id.btnGPSShowLocation);
        btnGPSShowLocation.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View arg0) {
                Location gpsLocation = appLocationService
                        .getLocation(LocationManager.GPS_PROVIDER);
                if (gpsLocation != null) {
                    double latitude = gpsLocation.getLatitude();
                    double longitude = gpsLocation.getLongitude();
                    String result = "Latitude: " + gpsLocation.getLatitude() +
                            " Longitude: " + gpsLocation.getLongitude();
                    tvAddress.setText(result);
                } else {
                    showSettingsAlert();
                }
            }
        });

        btnShowAddress = (Button) findViewById(R.id.btnShowAddress);
        btnShowAddress.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View arg0) {

                Location location = appLocationService
                        .getLocation(LocationManager.GPS_PROVIDER);

                //you can hard-code the lat & long if you have issues with getting it
                //remove the below if-condition and use the following couple of lines
                //double latitude = 37.422005;
                //double longitude = -122.084095

                if (location != null) {
                    double latitude = location.getLatitude();
                    double longitude = location.getLongitude();
                    LocationAddress locationAddress = new LocationAddress();
                    locationAddress.getAddressFromLocation(latitude, longitude,
                            getApplicationContext(), new GeocoderHandler());
                } else {
                    showSettingsAlert();
                }

            }
        });

    }
```

```
    public void showSettingsAlert() {
       AlertDialog.Builder alertDialog = new AlertDialog.Builder(
             MyActivity.this);
       alertDialog.setTitle("SETTINGS");
       alertDialog.setMessage("Enable Location Provider! Go to settings menu?");
       alertDialog.setPositiveButton("Settings",
             new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                   Intent intent = new Intent(
                         Settings.ACTION_LOCATION_SOURCE_SETTINGS);
                   MyActivity.this.startActivity(intent);
                }
             });
       alertDialog.setNegativeButton("Cancel",
             new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                   dialog.cancel();
                }
             });
       alertDialog.show();
    }

    private class GeocoderHandler extends Handler {
       @Override
       public void handleMessage(Message message) {
          String locationAddress;
          switch (message.what) {
             case 1:
                Bundle bundle = message.getData();
                locationAddress = bundle.getString("address");
                break;
             default:
                locationAddress = null;
          }
          tvAddress.setText(locationAddress);
       }
    }
}
```

Android UI layout file to show the address and location

```
<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=".MyActivity">
```

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

<Button
   style="?android:attr/buttonStyleSmall"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Show Location"
   android:id="@+id/btnGPSShowLocation"
   android:layout_toEndOf="@+id/textView"
   android:layout_marginTop="53dp"
   android:layout_below="@+id/textView"
   android:layout_alignParentStart="true" />

<Button
   style="?android:attr/buttonStyleSmall"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Show Address"
   android:id="@+id/btnShowAddress"
   android:layout_toEndOf="@+id/tvAddress"
   android:layout_below="@+id/btnGPSShowLocation"
   android:layout_alignParentStart="true" />

<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:id="@+id/tvAddress"
   android:layout_alignParentBottom="true"
   android:layout_marginBottom="134dp"
   android:layout_alignParentEnd="true" />

</RelativeLayout>
```

- **MediaPlayer**
- Android provides many ways to control playback of audio/video files and streams. One of this way is through a class called **MediaPlayer**.
- Android is providing MediaPlayer class to access built-in mediaplayer services like playing audio,video e.t.c. In order to use MediaPlayer, we have to call a static Method **create()** of this class. This method returns an instance of MediaPlayer class. Its syntax is as follows −

```
MediaPlayer mediaPlayer = MediaPlayer.create(this, R.raw.song);
```

- The second parameter is the name of the song that you want to play. You have to make a new folder under your project with name **raw** and place the music file into it.

- Once you have created the Mediaplayer object you can call some methods to start or stop the music. These methods are listed below.

```
mediaPlayer.start();
mediaPlayer.pause();
```

- On call to **start()** method, the music will start playing from the beginning. If this method is called again after the **pause()**method, the music would start playing from where it is left and not from the beginning.
- In order to start music from the beginning, you have to call **reset()** method. Its syntax is given below.

```
mediaPlayer.reset();
```

- Apart from the start and pause method, there are other methods provided by this class for better dealing with audio/video files. These methods are listed below −

Sr.No	Method & description
1	**isPlaying()** This method just returns true/false indicating the song is playing or not
2	**seekTo(position)** This method takes an integer, and move song to that particular second
3	**getCurrentDuration()** This method returns the current position of song in milliseconds
4	**getDuration()** This method returns the total time duration of song in milliseconds
5	**reset()** This method resets the media player
6	**release()** This method releases any resource attached with MediaPlayer object
7	**setVolume(float leftVolume, float rightVolume)** This method sets the up down volume for this player
8	**setDataSource(FileDescriptor fd)** This method sets the data source of audio/video file
9	**selectTrack(int index)** This method takes an integer, and select the track from the list on that particular index
10	**getTrackInfo()** This method returns an array of track information

- Example
- Here is an example demonstrating the use of MediaPlayer class. It creates a basic media player that allows you to forward, backward, play and pause a song.
- To experiment with this example, you need to run this on an actual device to hear the audio sound.

Steps	Description
1	You will use Android studio IDE to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add MediaPlayer code.
3	Modify the res/layout/activity_main to add respective XML components
4	Create a new folder under MediaPlayer with name as raw and place an mp3 music file in it with name as song.mp3

5	Run the application and choose a running android device and install the application on it and verify the results

- Following is the content of the modified main activity file **src/MainActivity.java**.

```
package com.example.sairamkrishna.myapplication;

import android.app.Activity;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.Handler;
import android.view.View;

import android.widget.Button;
import android.widget.ImageView;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;
import java.util.concurrent.TimeUnit;


public class MainActivity extends Activity {
  private Button b1,b2,b3,b4;
  private ImageView iv;
  private MediaPlayer mediaPlayer;

  private double startTime = 0;
  private double finalTime = 0;

  private Handler myHandler = new Handler();;
  private int forwardTime = 5000;
  private int backwardTime = 5000;
  private SeekBar seekbar;
  private TextView tx1,tx2,tx3;

  public static int oneTimeOnly = 0;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    b1 = (Button) findViewById(R.id.button);
    b2 = (Button) findViewById(R.id.button2);
    b3 = (Button)findViewById(R.id.button3);
    b4 = (Button)findViewById(R.id.button4);
    iv = (ImageView)findViewById(R.id.imageView);

    tx1 = (TextView)findViewById(R.id.textView2);
    tx2 = (TextView)findViewById(R.id.textView3);
    tx3 = (TextView)findViewById(R.id.textView4);
    tx3.setText("Song.mp3");

    mediaPlayer = MediaPlayer.create(this, R.raw.song);
    seekbar = (SeekBar)findViewById(R.id.seekBar);
    seekbar.setClickable(false);
```

```
b2.setEnabled(false);

b3.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    Toast.makeText(getApplicationContext(), "Playing
      sound",Toast.LENGTH_SHORT).show();
    mediaPlayer.start();

    finalTime = mediaPlayer.getDuration();
    startTime = mediaPlayer.getCurrentPosition();

    if (oneTimeOnly == 0) {
      seekbar.setMax((int) finalTime);
      oneTimeOnly = 1;
    }

    tx2.setText(String.format("%d min, %d sec",
      TimeUnit.MILLISECONDS.toMinutes((long) finalTime),
      TimeUnit.MILLISECONDS.toSeconds((long) finalTime) -
      TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes((long)
        finalTime)))
    );

    tx1.setText(String.format("%d min, %d sec",
      TimeUnit.MILLISECONDS.toMinutes((long) startTime),
      TimeUnit.MILLISECONDS.toSeconds((long) startTime) -
      TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes((long)
        startTime)))
    );

    seekbar.setProgress((int)startTime);
    myHandler.postDelayed(UpdateSongTime,100);
    b2.setEnabled(true);
    b3.setEnabled(false);
  }
});

b2.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    Toast.makeText(getApplicationContext(), "Pausing
      sound",Toast.LENGTH_SHORT).show();
    mediaPlayer.pause();
    b2.setEnabled(false);
    b3.setEnabled(true);
  }
});

b1.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    int temp = (int)startTime;

    if((temp+forwardTime)<=finalTime){
```

```
            startTime = startTime + forwardTime;
            mediaPlayer.seekTo((int) startTime);
            Toast.makeText(getApplicationContext(),"You have Jumped forward 5
              seconds",Toast.LENGTH_SHORT).show();
          }else{
            Toast.makeText(getApplicationContext(),"Cannot jump forward 5
              seconds",Toast.LENGTH_SHORT).show();
          }
        }
      });

      b4.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
          int temp = (int)startTime;

          if((temp-backwardTime)>0){
            startTime = startTime - backwardTime;
            mediaPlayer.seekTo((int) startTime);
            Toast.makeText(getApplicationContext(),"You have Jumped backward 5
              seconds",Toast.LENGTH_SHORT).show();
          }else{
            Toast.makeText(getApplicationContext(),"Cannot jump backward 5
              seconds",Toast.LENGTH_SHORT).show();
          }
        }
      });
    }

    private Runnable UpdateSongTime = new Runnable() {
      public void run() {
        startTime = mediaPlayer.getCurrentPosition();
        tx1.setText(String.format("%d min, %d sec",
          TimeUnit.MILLISECONDS.toMinutes((long) startTime),
          TimeUnit.MILLISECONDS.toSeconds((long) startTime) -
          TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.
          toMinutes((long) startTime)))
        );
        seekbar.setProgress((int)startTime);
        myHandler.postDelayed(this, 100);
      }
    };
}
```

- Following is the modified content of the xml **res/layout/activity_main.xml**.
- In the below code **abc**indicates the logo of tutorial.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">
```

```
<TextView android:text="Music Palyer" 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:layout_below="@+id/textView"
  android:layout_centerHorizontal="true"
  android:src="@drawable/abc"/>

<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/forward"
  android:id="@+id/button"
  android:layout_alignParentBottom="true"
  android:layout_alignParentLeft="true"
  android:layout_alignParentStart="true" />

<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/pause"
  android:id="@+id/button2"
  android:layout_alignParentBottom="true"
  android:layout_alignLeft="@+id/imageView"
  android:layout_alignStart="@+id/imageView" />

<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/back"
  android:id="@+id/button3"
  android:layout_alignTop="@+id/button2"
  android:layout_toRightOf="@+id/button2"
  android:layout_toEndOf="@+id/button2" />

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

```
    android:layout_height="wrap_content"
    android:text="@string/rewind"
    android:id="@+id/button4"
    android:layout_alignTop="@+id/button3"
    android:layout_toRightOf="@+id/button3"
    android:layout_toEndOf="@+id/button3" />

  <SeekBar
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/seekBar"
    android:layout_alignLeft="@+id/textview"
    android:layout_alignStart="@+id/textview"
    android:layout_alignRight="@+id/textview"
    android:layout_alignEnd="@+id/textview"
    android:layout_above="@+id/button" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceSmall"
    android:text="Small Text"
    android:id="@+id/textView2"
    android:layout_above="@+id/seekBar"
    android:layout_toLeftOf="@+id/textView"
    android:layout_toStartOf="@+id/textView" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceSmall"
    android:text="Small Text"
    android:id="@+id/textView3"
    android:layout_above="@+id/seekBar"
    android:layout_alignRight="@+id/button4"
    android:layout_alignEnd="@+id/button4" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Medium Text"
    android:id="@+id/textView4"
    android:layout_alignBaseline="@+id/textView2"
    android:layout_alignBottom="@+id/textView2"
    android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Following is the content of the **res/values/string.xml**.

```
<resources>
  <string name="app_name">My Application</string>
  <string name="back"><![CDATA[<]]></string>
  <string name="rewind"><![CDATA[<<]]></string>
```

```
    <string name="forward"><![CDATA[>>]]></string>
    <string name="pause">||</string>
</resources>
```

- Following is the content of **AndroidManifest.xml** file.

```
<?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="@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 Eclipse, open one of your project's activity files and click Run icon from the toolbar. Before starting your application, Android studio will display following screens

- **recording audio**
- 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) {
        e.printStackTrace();
      } catch (IOException e) {
        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 tutorial

```
<?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.
- Now just press the play button and and recorded audio will just start playing on the device.

- **Camera**
- These are the following two ways, in which you can use camera in your application

- Using existing android camera application in our application

- Directly using Camera API provided by android in our application

- Using existing android camera application in our application
- You will use MediaStore.ACTION_IMAGE_CAPTURE to launch an existing camera application installed on your phone. Its syntax is given below

```
Intent intent = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
```

- Apart from the above, there are other available Intents provided by MediaStore. They are listed as follows

Sr.No	Intent type and description
1	**ACTION_IMAGE_CAPTURE_SECURE** It returns the image captured from the camera , when the device is secured
2	**ACTION_VIDEO_CAPTURE** It calls the existing video application in android to capture video
3	**EXTRA_SCREEN_ORIENTATION** It is used to set the orientation of the screen to vertical or landscape
4	**EXTRA_FULL_SCREEN** It is used to control the user interface of the ViewImage
5	**INTENT_ACTION_VIDEO_CAMERA** This intent is used to launch the camera in the video mode
6	**EXTRA_SIZE_LIMIT**

	It is used to specify the size limit of video or image capture size

- Now you will use the function *startActivityForResult()* to launch this activity and wait for its result. Its syntax is given below

```
startActivityForResult(intent,0)
```

- This method has been defined in the **activity** class. We are calling it from main activity. There are methods defined in the activity class that does the same job , but used when you are not calling from the activity but from somewhere else. They are listed below

Sr.No	Activity function description
1	**startActivityForResult(Intent intent, int requestCode, Bundle options)** It starts an activity , but can take extra bundle of options with it
2	**startActivityFromChild(Activity child, Intent intent, int requestCode)** It launch the activity when your activity is child of any other activity
3	**startActivityFromChild(Activity child, Intent intent, int requestCode, Bundle options)** It work same as above , but it can take extra values in the shape of bundle with it
4	**startActivityFromFragment(Fragment fragment, Intent intent, int requestCode)** It launches activity from the fragment you are currently inside
5	**startActivityFromFragment(Fragment fragment, Intent intent, int requestCode, Bundle options)** It not only launches the activity from the fragment , but can take extra values with it

- No matter which function you used to launch the activity , they all return the result. The result can be obtained by overriding the function *onActivityResult*.
- Example
- Here is an example that shows how to launch the existing camera application to capture an image and display the result in the form of bitmap.
- To experiment with this example , you need to run this on an actual device on which camera is supported.

Steps	Description
1	You will use Android studio IDE to create an Android application and name it as Camera under a com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add intent code to launch the Camera.
3	Modify layout XML file res/layout/activity_main.xml
4	Add the Camera permission and run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/MainActivity.java.**

```
package com.example.sairamkrishna.myapplication;

import android.Manifest;
import android.app.Activity;
import android.app.AlertDialog;

import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
```

```
import android.net.Uri;
import android.os.Bundle;
import android.provider.Settings;

import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;

public class MainActivity extends AppCompatActivity {
  public static final int MY_PERMISSIONS_REQUEST_CAMERA = 100;
  public static final String ALLOW_KEY = "ALLOWED";
  public static final String CAMERA_PREF = "camera_pref";

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

    if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) !=
PackageManager.PERMISSION_GRANTED) {
      if (getFromPref(this, ALLOW_KEY)) {
        showSettingsAlert();
      } else if (ContextCompat.checkSelfPermission(this,
        Manifest.permission.CAMERA)

        != PackageManager.PERMISSION_GRANTED) {

          // Should we show an explanation?
          if (ActivityCompat.shouldShowRequestPermissionRationale(this,
            Manifest.permission.CAMERA)) {
            showAlert();
          } else {
            // No explanation needed, we can request the permission.
            ActivityCompat.requestPermissions(this,
              new String[]{Manifest.permission.CAMERA},
              MY_PERMISSIONS_REQUEST_CAMERA);
          }
        }
    } else {
      openCamera();
    }

  }
  public static void saveToPreferences(Context context, String key, Boolean allowed) {
    SharedPreferences myPrefs = context.getSharedPreferences(CAMERA_PREF,
      Context.MODE_PRIVATE);
    SharedPreferences.Editor prefsEditor = myPrefs.edit();
    prefsEditor.putBoolean(key, allowed);
    prefsEditor.commit();
  }

  public static Boolean getFromPref(Context context, String key) {
    SharedPreferences myPrefs = context.getSharedPreferences(CAMERA_PREF,
```

```
        Context.MODE_PRIVATE);
    return (myPrefs.getBoolean(key, false));
}

private void showAlert() {
    AlertDialog alertDialog = new AlertDialog.Builder(MainActivity.this).create();
    alertDialog.setTitle("Alert");
    alertDialog.setMessage("App needs to access the Camera.");

    alertDialog.setButton(AlertDialog.BUTTON_NEGATIVE, "DONT ALLOW",
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
                finish();
            }
    });

    alertDialog.setButton(AlertDialog.BUTTON_POSITIVE, "ALLOW",
        new DialogInterface.OnClickListener() {

            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
                ActivityCompat.requestPermissions(MainActivity.this,
                new String[]{Manifest.permission.CAMERA},
                MY_PERMISSIONS_REQUEST_CAMERA);
            }
    });
    alertDialog.show();
}

private void showSettingsAlert() {
    AlertDialog alertDialog = new AlertDialog.Builder(MainActivity.this).create();
    alertDialog.setTitle("Alert");
    alertDialog.setMessage("App needs to access the Camera.");

    alertDialog.setButton(AlertDialog.BUTTON_NEGATIVE, "DONT ALLOW",
        new DialogInterface.OnClickListener() {

            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
                //finish();
            }
    });

    alertDialog.setButton(AlertDialog.BUTTON_POSITIVE, "SETTINGS",
        new DialogInterface.OnClickListener() {

            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
                startInstalledAppDetailsActivity(MainActivity.this);
            }
    });

    alertDialog.show();
}
```

```
  @Override
  public void onRequestPermissionsResult(int requestCode, String permissions[], int[]
grantResults) {
    switch (requestCode) {
      case MY_PERMISSIONS_REQUEST_CAMERA: {
        for (int i = 0, len = permissions.length; i < len; i++) {
          String permission = permissions[i];

          if (grantResults[i] == PackageManager.PERMISSION_DENIED) {
            boolean
            showRationale =
              ActivityCompat.shouldShowRequestPermissionRationale(
              this, permission);

            if (showRationale) {
              showAlert();
            } else if (!showRationale) {
              // user denied flagging NEVER ASK AGAIN
              // you can either enable some fall back,
              // disable features of your app
              // or open another dialog explaining
              // again the permission and directing to
              // the app setting
              saveToPreferences(MainActivity.this, ALLOW_KEY, true);
            }
          }
        }
      }

      // other 'case' lines to check for other
      // permissions this app might request
    }
  }

  @Override
  protected void onResume() {
    super.onResume();
  }

  public static void startInstalledAppDetailsActivity(final Activity context) {
    if (context == null) {
      return;
    }

    final Intent i = new Intent();
    i.setAction(Settings.ACTION_APPLICATION_DETAILS_SETTINGS);
    i.addCategory(Intent.CATEGORY_DEFAULT);
    i.setData(Uri.parse("package:" + context.getPackageName()));
    i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    i.addFlags(Intent.FLAG_ACTIVITY_NO_HISTORY);
    i.addFlags(Intent.FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS);
    context.startActivity(i);
  }
```

```
    private void openCamera() {
      Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
      startActivity(intent);
    }
}
```

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

```
<?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">
</RelativeLayout>
```

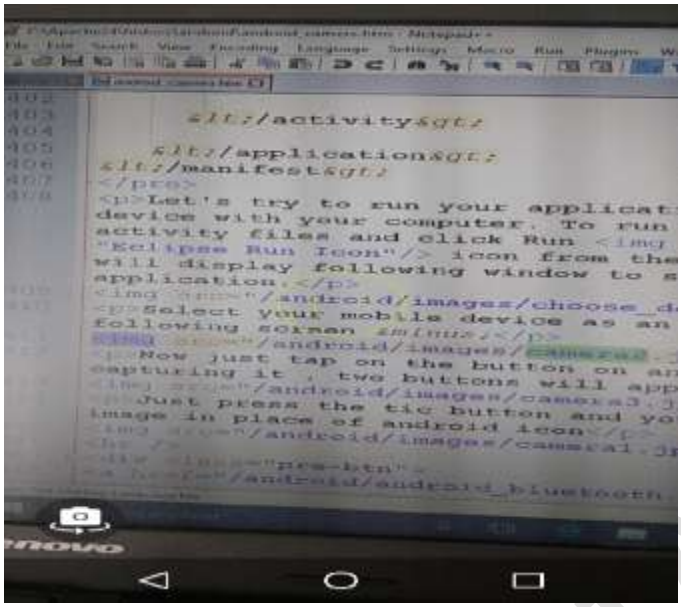
- Following will be the content of **res/values/strings.xml** to define one new constants

```
<resources>
  <string name="app_name">My Application</string>
</resources>
```

- Following is the default 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.CAMERA" />
  <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 tool bar.



- **Handling Telephony**
- Android provides Built-in applications for phone calls, in some occasions we may need to make a phone call through our application. This could easily be done by using implicit Intent with appropriate actions. Also, we can use PhoneStateListener and TelephonyManager classes, in order to monitor the changes in some telephony states on the device.
- This chapter lists down all the simple steps to create an application which can be used to make a Phone Call. You can use Android Intent to make phone call by calling built-in Phone Call functionality of the Android. Following section explains different parts of our Intent object required to make a call.
- Intent Object - Action to make Phone Call
- You will use **ACTION_CALL** action to trigger built-in phone call functionality available in Android device. Following is simple syntax to create an intent with ACTION_CALL action

```
Intent phoneIntent = new Intent(Intent.ACTION_CALL);
```

You can use **ACTION_DIAL** action instead of ACTION_CALL, in that case you will have option to modify hardcoded phone number before making a call instead of making a direct call.

Intent Object - Data/Type to make Phone Call

To make a phone call at a given number 91-000-000-0000, you need to specify **tel:** as URI using setData() method as follows −

```
phoneIntent.setData(Uri.parse("tel:91-000-000-0000"));
```

- The interesting point is that, to make a phone call, you do not need to specify any extra data or data type.
- Example
- Following example shows you in practical how to use Android Intent to make phone call to the given mobile number.

- To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *My Application* under a package *com.example.saira_000.myapplication*.
2	Modify *src/MainActivity.java* file and add required code to take care of making a call.
3	Modify layout XML file *res/layout/activity_main.xml* add any GUI component if required. I'm adding a simple button to Call 91-000-000-0000 number
4	No need to define default string constants.Android studio takes care of default constants.
5	Modify *AndroidManifest.xml* as shown below
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

- Following is the content of the modified main activity file **src/MainActivity.java**.

```
package com.example.saira_000.myapplication;

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

public class MainActivity extends AppCompatActivity {
  private Button button;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    button = (Button) findViewById(R.id.buttonCall);

    button.setOnClickListener(new View.OnClickListener() {
      public void onClick(View arg0) {
        Intent callIntent = new Intent(Intent.ACTION_CALL);
        callIntent.setData(Uri.parse("tel:0377778888"));

        if (ActivityCompat.checkSelfPermission(MainActivity.this,
          Manifest.permission.CALL_PHONE) !=
PackageManager.PERMISSION_GRANTED) {
            return;
          }
          startActivity(callIntent);
      }
    });

  }
}
```

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

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

  <Button
    android:id="@+id/buttonCall"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="call 0377778888" />

</LinearLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">My Application</string>
</resources>
```

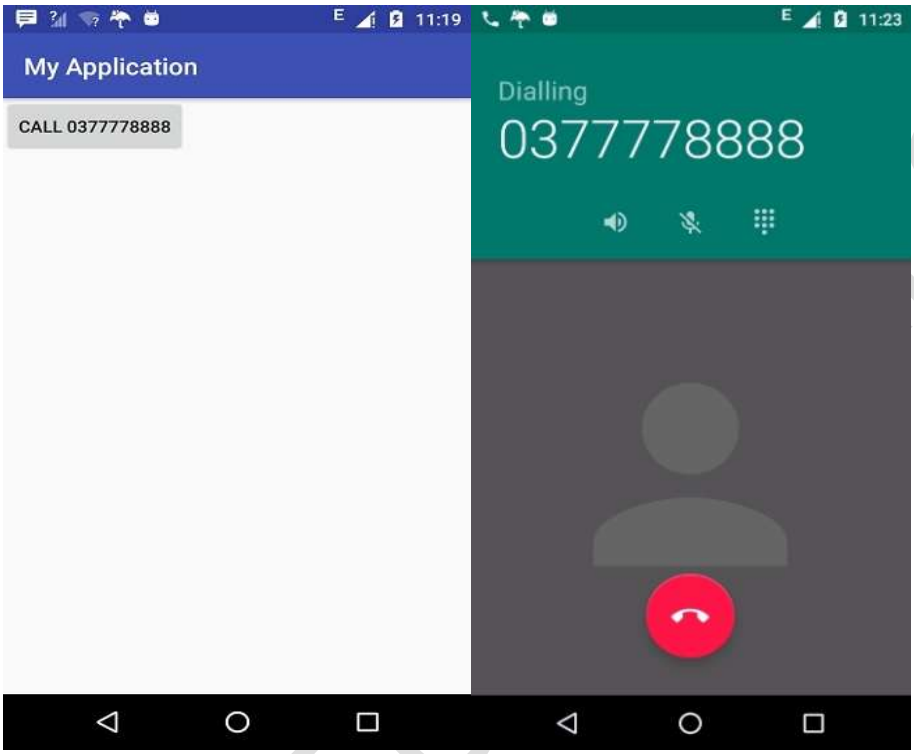
- Following is the default content of **AndroidManifest.xml** −

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.saira_000.myapplication" >

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

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.saira_000.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 **My Application** 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.Select your mobile device as an option and then check your mobile device which will display following screen –

- Now use **Call** button to make phone call as shown below −



- **Handling SMS,**
- In Android, you can use SmsManager API or devices Built-in SMS application to send SMS's. In this tutorial, we shows you two basic examples to send SMS message −
- **SmsManager API**

```
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage("phoneNo", null, "sms message", null, null);
```

- **Built-in SMS application**

```
Intent sendIntent = new Intent(Intent.ACTION_VIEW);
sendIntent.putExtra("sms_body", "default content");
sendIntent.setType("vnd.android-dir/mms-sms");
startActivity(sendIntent);
```

- Of course, both need **SEND_SMS permission**.

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

- Apart from the above method, there are few other important functions available in SmsManager class. These methods are listed below −

Sr.No.	Method & Description
1	**ArrayList<String> divideMessage(String text)** This method divides a message text into several fragments, none bigger than the maximum SMS message size.
2	**static SmsManager getDefault()** This method is used to get the default instance of the SmsManager
3	**void sendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent)** This method is used to send a data based SMS to a specific application port.
4	**void sendMultipartTextMessage(String destinationAddress, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntents, ArrayList<PendingIntent> deliveryIntents)** Send a multi-part text based SMS.
5	**void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)** Send a text based SMS.

- **Example**
- Following example shows you in practical how to use SmsManager object to send an SMS to the given mobile number.
- To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

Step	Description
1	You will use Android Studio IDE to create an Android application and name it as *tutorial* under a package *com.example.tutorial*.
2	Modify *src/MainActivity.java* file and add required code to take care of sending sms.
3	Modify layout XML file *res/layout/activity_main.xml* add any GUI component if required. I'm adding a simple GUI to take mobile number and SMS text to be sent and a simple button to send SMS.
4	No need to define default string constants at res/values/strings.xml. Android studio takes care of default constants.
5	Modify *AndroidManifest.xml* as shown below
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

- Following is the content of the modified main activity file **src/com.example.tutorial/MainActivity.java**.

```
package com.example.tutorial;

import android.Manifest;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.app.Activity;

import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.telephony.SmsManager;

import android.util.Log;
import android.view.Menu;
import android.view.View;
```

```
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {
  private static final int MY_PERMISSIONS_REQUEST_SEND_SMS =0 ;
  Button sendBtn;
  EditText txtphoneNo;
  EditText txtMessage;
  String phoneNo;
  String message;

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

    sendBtn = (Button) findViewById(R.id.btnSendSMS);
    txtphoneNo = (EditText) findViewById(R.id.editText);
    txtMessage = (EditText) findViewById(R.id.editText2);

    sendBtn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        sendSMSMessage();
      }
    });
  }

  protected void sendSMSMessage() {
    phoneNo = txtphoneNo.getText().toString();
    message = txtMessage.getText().toString();

    if (ContextCompat.checkSelfPermission(this,
      Manifest.permission.SEND_SMS)
      != PackageManager.PERMISSION_GRANTED) {
        if (ActivityCompat.shouldShowRequestPermissionRationale(this,
          Manifest.permission.SEND_SMS)) {
        } else {
          ActivityCompat.requestPermissions(this,
            new String[]{Manifest.permission.SEND_SMS},
            MY_PERMISSIONS_REQUEST_SEND_SMS);
      }
    }
  }

  @Override
  public void onRequestPermissionsResult(int requestCode,String permissions[], int[]
grantResults) {
    switch (requestCode) {
      case MY_PERMISSIONS_REQUEST_SEND_SMS: {
        if (grantResults.length > 0
          && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            SmsManager smsManager = SmsManager.getDefault();
            smsManager.sendTextMessage(phoneNo, null, message, null, null);
```

```
            Toast.makeText(getApplicationContext(), "SMS sent.",
              Toast.LENGTH_LONG).show();
          } else {
            Toast.makeText(getApplicationContext(),
              "SMS faild, please try again.", Toast.LENGTH_LONG).show();
            return;
          }
        }
      }

    }
}
```

- Following will be the content of **res/layout/activity_main.xml**file −
- Here abc indicates about tutorial logo

```
<?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"
  tools:context="MainActivity">

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Sending SMS Example"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="30dp" />

  <TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials point "
    android:textColor="#ff87ff09"
    android:textSize="30dp"
    android:layout_below="@+id/textView1"
    android:layout_alignRight="@+id/imageButton"
    android:layout_alignEnd="@+id/imageButton" />

  <ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_below="@+id/textView2"
    android:layout_centerHorizontal="true" />
```

```
<EditText
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/editText"
  android:hint="Enter Phone Number"
  android:phoneNumber="true"
  android:textColorHint="@color/abc_primary_text_material_dark"
  android:layout_below="@+id/imageButton"
  android:layout_centerHorizontal="true" />

<EditText
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/editText2"
  android:layout_below="@+id/editText"
  android:layout_alignLeft="@+id/editText"
  android:layout_alignStart="@+id/editText"
  android:textColorHint="@color/abc_primary_text_material_dark"
  android:layout_alignRight="@+id/imageButton"
  android:layout_alignEnd="@+id/imageButton"
  android:hint="Enter SMS" />

<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Send Sms"
  android:id="@+id/btnSendSMS"
  android:layout_below="@+id/editText2"
  android:layout_centerHorizontal="true"
  android:layout_marginTop="48dp" />

</RelativeLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">tutorial</string>
</resources>
```

- Following is the default content of **AndroidManifest.xml** −

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

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

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
```

```
    <activity
      android:name="com.example.tutorial.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 **tutorial** 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 installer will display following window to select an option where you want to run your Android application.

- Now you can enter a desired mobile number and a text message to be sent on that number. Finally click on **Send SMS** button to send your SMS. Make sure your GSM/CDMA connection is working fine to deliver your SMS to its recipient.
- You can take a number of SMS separated by comma and then inside your program you will have to parse them into an array string and finally you can use a loop to send message

to all the given numbers. That's how you can write your own SMS client. Next section will show you how to use existing SMS client to send SMS.

- Using Built-in Intent to send SMS
- You can use Android Intent to send SMS by calling built-in SMS functionality of the Android. Following section explains different parts of our Intent object required to send an SMS.
- Intent Object - Action to send SMS
- You will use **ACTION_VIEW** action to launch an SMS client installed on your Android device. Following is simple syntax to create an intent with ACTION_VIEW action.

```
Intent smsIntent = new Intent(Intent.ACTION_VIEW);
```

- Intent Object - Data/Type to send SMS
- To send an SMS you need to specify **smsto:** as URI using setData() method and data type will be to **vnd.android-dir/mms-sms** using setType() method as follows −

```
smsIntent.setData(Uri.parse("smsto:"));
smsIntent.setType("vnd.android-dir/mms-sms");
```

- Intent Object - Extra to send SMS

Android has built-in support to add phone number and text message to send an SMS as follows −

```
smsIntent.putExtra("address"  , new String("0123456789;3393993300"));
smsIntent.putExtra("sms_body"  , "Test SMS to Angilla");
```

- Here address and sms_body are case sensitive and should be specified in small characters only. You can specify more than one number in single string but separated by semi-colon (;).

- **Example**
- Following example shows you in practical how to use Intent object to launch SMS client to send an SMS to the given recipients.
- To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *tutorial* under a package *com.example.tutorial*.
2	Modify *src/MainActivity.java* file and add required code to take care of sending SMS.
3	Modify layout XML file *res/layout/activity_main.xml* add any GUI component if required. I'm adding a simple button to launch SMS Client.
4	No need to define default constants.Android studio takes care of default constants.
5	Modify *AndroidManifest.xml* as shown below
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

- Following is the content of the modified main activity file **src/com.example.tutorial/MainActivity.java**.

```
package com.example.tutorial;
```

```
import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Button startBtn = (Button) findViewById(R.id.button);
    startBtn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        sendSMS();
      }
    });
  }

  protected void sendSMS() {
    Log.i("Send SMS", "");
    Intent smsIntent = new Intent(Intent.ACTION_VIEW);

    smsIntent.setData(Uri.parse("smsto:"));
    smsIntent.setType("vnd.android-dir/mms-sms");
    smsIntent.putExtra("address"  , new String ("01234"));
    smsIntent.putExtra("sms_body"  , "Test ");

    try {
      startActivity(smsIntent);
      finish();
      Log.i("Finished sending SMS...", "");
    } catch (android.content.ActivityNotFoundException ex) {
      Toast.makeText(MainActivity.this,
      "SMS faild, please try again later.", Toast.LENGTH_SHORT).show();
    }
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
  }
}
```

- Following will be the content of **res/layout/activity_main.xml**file −
- Here abc indicates about tutorial logo

```
<?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">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Drag and Drop Example"
    android:id="@+id/textView"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="30dp" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials Point "
    android:id="@+id/textView2"
    android:layout_below="@+id/textView"
    android:layout_centerHorizontal="true"
    android:textSize="30dp"
    android:textColor="#ff14be3c" />

  <ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageView"
    android:src="@drawable/abc"
    android:layout_marginTop="48dp"
    android:layout_below="@+id/textView2"
    android:layout_centerHorizontal="true" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Compose SMS"
    android:id="@+id/button"
    android:layout_below="@+id/imageView"
    android:layout_alignRight="@+id/textView2"
    android:layout_alignEnd="@+id/textView2"
    android:layout_marginTop="54dp"
    android:layout_alignLeft="@+id/imageView"
    android:layout_alignStart="@+id/imageView" />

</RelativeLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">tutorial</string>
</resources>
```

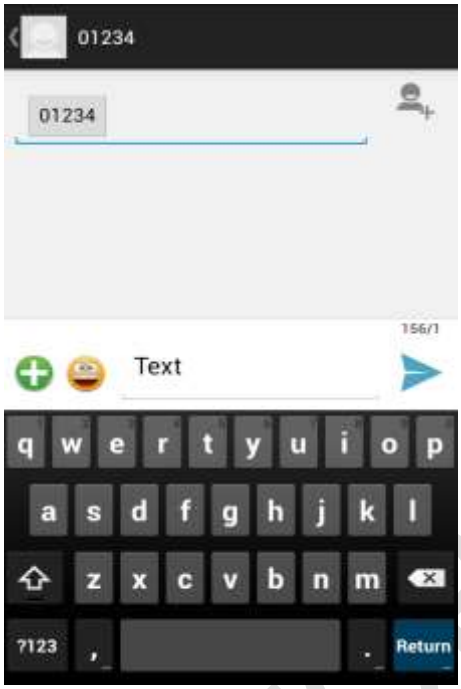
Following is the default content of **AndroidManifest.xml** −

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

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >

      <activity
         android:name="com.example.tutorial.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.

Select your mobile device as an option and then check your mobile device which will display following screen −

- Now use **Compose SMS** button to launch Android built-in SMS clients which is shown below –



UNIT 4

- **Working with Bluetooth and Wi-Fi –**
- **BluetoothAdapter and Managing discoverability mode**
- Bluetooth is a communications protocol designed for short-range, low-bandwidth peer-to-peer communications.
  Using the Bluetooth APIs, you can search for, and connect to, other Bluetooth devices within range. By initiating a communications link using Bluetooth Sockets, you can then transmit and receive streams of data between devices from within your applications.
- At the time of writing, only encrypted communication is supported between devices, meaning that you can form connections only between devices that have been paired.

- **Managing the Local Bluetooth Device Adapter**
- The local Bluetooth device is controlled via the BluetoothAdapter class, which represents the host Android device on which your application is running.
- To access the default Bluetooth Adapter, call getDefaultAdapter. Some Android devices feature multiple Bluetooth adapters, though it is currently only possible to access the default device.

- **Accessing the default Bluetooth Adapter**
- BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();
- To read any of the local Bluetooth Adapter properties, initiate discovery, or fi nd bonded devices, you need to include the BLUETOOTH permission in your application manifest. To modify any of the local device properties, the BLUETOOTH_ADMIN permission is also required:

  <uses-permission android:name=”android.permission.BLUETOOTH”/>
  <uses-permission android:name=”android.permission.BLUETOOTH_ADMIN”/>

- The Bluetooth Adapter offers methods for reading and setting properties of the local Bluetooth hardware.
- The Bluetooth Adapter properties can be read and changed only if the Bluetooth Adapter is currently turned on — that is, if its device state is enabled. If the device is off, these methods will return null.
- Use the isEnabled method to confirm the device is enabled, after which you can access the Bluetooth Adapter’s friendly name (an arbitrary string that users can set to identify a particular device) and hardware address, using the getName and getAddress methods, respectively:

```
if (bluetooth.isEnabled()) {
    String address = bluetooth.getAddress();
 String name = bluetooth.getName();
}
```

  If you have the BLUETOOTH_ADMIN permission, you can change the friendly name of the Bluetooth Adapter using the setName method:

```
bluetooth.setName(“Blackfang”);
```

- To find a more detailed description of the current Bluetooth Adapter state, use the getState method, which will return one of the following BluetoothAdapter constants:
- STATE_TURNING_ON

- STATE_ON
- STATE_TURNING_OFF
- STATE_OFF
- To conserve battery life and optimize security, most users will keep Bluetooth disabled until they plan to use it.
  To enable the Bluetooth Adapter, you can start a system Preference Activity using the BluetoothAdapter.ACTION_REQUEST_ENABLE static constant as a startActivityForResult action string:

```
startActivityForResult(
new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE), 0);
```

Figure 1 shows the resulting Preference Activity.

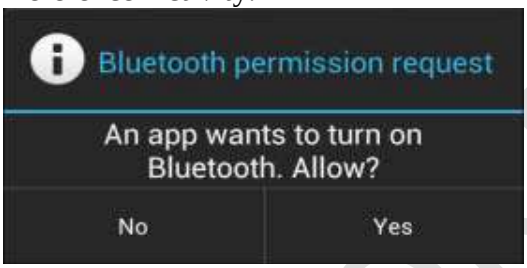


Figure 1

- It prompts the user to turn on Bluetooth and asks for confirmation. If the user agrees, the sub-Activity will close and return to the calling Activity when the Bluetooth Adapter has turned on (or has encountered an error). If the user selects no, the sub-Activity will close and return immediately. Use the result code parameter returned in the on Activity Result handler to determine the success of this operation.

- **Enabling Bluetooth**

```
private static final int ENABLE_BLUETOOTH = 1;
private void init Bluetooth() {
        if (!bluetooth.isEnabled()) {
        // Bluetooth isn't enabled, prompt the user to turn it on.
        Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(intent, ENABLE_BLUETOOTH);
        } else {
        // Bluetooth is enabled, initialize the UI.
        initBluetoothUI();
        }
}
protected void onActivityResult(int requestCode,
        int resultCode, Intent data) {
        if (requestCode == ENABLE_BLUETOOTH)
                if (resultCode == RESULT_OK) {
                // Bluetooth has been enabled, initialize the UI.
                initBluetoothUI();
        }
}
```

- Enabling and disabling the Bluetooth Adapter are somewhat time-consuming, asynchronous operations. Rather than polling the Bluetooth Adapter, your application

should register a Broadcast Receiver that listens for ACTION_STATE_CHANGED. The Broadcast Intent will include two extras, EXTRA_STATE and EXTRA_PREVIOUS_STATE, which indicate the current and previous Bluetooth Adapter states, respectively:

```
BroadcastReceiver bluetoothState = new BroadcastReceiver() {
        @Override
         public void onReceive(Context context, Intent intent) {
                String prevStateExtra = BluetoothAdapter.EXTRA_PREVIOUS_STATE;
                String stateExtra = BluetoothAdapter.EXTRA_STATE;
                int state = intent.getIntExtra(stateExtra, −1);
                int previousState = intent.getIntExtra(prevStateExtra, −1);
                String tt = "";
                switch (state) {
                        case (BluetoothAdapter.STATE_TURNING_ON) :
                        tt = "Bluetooth turning on"; break;
                        case (BluetoothAdapter.STATE_ON)
                       tt = "Bluetooth on"; break;
                        case (BluetoothAdapter.STATE_TURNING_OFF) :
                        tt = "Bluetooth turning off"; break;
                        case (BluetoothAdapter.STATE_OFF) :
                        tt = "Bluetooth off"; break;
                        default: break;
                        }
                Log.d(TAG, tt);
        }
};

String actionStateChanged = BluetoothAdapter.ACTION_STATE_CHANGED;
registerReceiver (bluetoothState,
new  IntentFilter(actionStateChanged));
```

- You can also turn the Bluetooth Adapter on and off directly, using the enable and disable methods, respectively, if you include the BLUETOOTH_ADMIN permission in your manifest.

- This should be done only when absolutely necessary, and the user should always be notified if you are manually changing the Bluetooth Adapter status on the user's behalf. In most cases you should use the Intent mechanism described earlier.

- **Being Discoverable and Remote Device Discovery**
- The process of two devices finding each other to connect is called discovery. Before you can establish a Bluetooth Socket for communications, the local Bluetooth Adapter must bond with the remote device. Before two devices can bond and connect, they first need to discover each other.
  Although the Bluetooth protocol supports ad-hoc connections for data transfer, this mechanism is not currently available in Android. Android Bluetooth communication is currently supported only between bonded devices.

- **Managing Device Discoverability**

- For an Android device to find your local Bluetooth Adapter during a discovery scan, you need to ensure that it's discoverable. The Bluetooth Adapter's discoverability is indicated by its scan mode, found using the getScanMode method on the BluetoothAdapter object.

- It will return one of the following BluetoothAdapter constants:

- SCAN_MODE_CONNECTABLE_DISCOVERABLE — Inquiry scan and page scan are both enabled, meaning that the device is discoverable from any Bluetooth device performing a discovery scan.
- SCAN_MODE_CONNECTABLE — Page scan is enabled but inquiry scan is not. This means that devices that have previously connected and bonded to the local device can find it during discovery, but new devices can't.
- SCAN_MODE_NONE — Discoverability is turned off. No remote devices can find the local Bluetooth Adapter during discovery.

- For privacy reasons, Android devices will default to having discoverability disabled. To turn on discovery, you need to obtain explicit permission from the user; you do this by starting a new Activity using the ACTION_REQUEST_DISCOVERABLE action.

3. Enabling discoverability

```
startActivityForResult(
    new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE),
        DISCOVERY_REQUEST);
```

- By default, discoverability will be enabled for 2 minutes. You can modify this setting by adding an EXTRA_DISCOVERABLE_DURATION extra to the launch Intent, specifying the number of seconds you want discoverability to last.
- When the Intent is broadcast, the user will be prompted by the dialog, as shown in Figure .2, to turn on discoverability for the specified duration.

Figure.2

- **Monitoring discoverability request approval**

```
@Override
protected void onActivityResult(int requestCode,
        int resultCode, Intent data) {
        if (requestCode == DISCOVERY_REQUEST) {
                if (resultCode == RESULT_CANCELED) {
                 Log.d(TAG, "Discovery canceled by user");
                 }
```

```
        }
    }
```

- Alternatively, you can monitor changes in discoverability by receiving the ACTION_SCAN_MODE_CHANGED broadcast action. The Broadcast Intent ncludes the current and previous scan modes as extras:

```
registerReceiver(new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
        String prevScanMode =
BluetoothAdapter.EXTRA_PREVIOUS_SCAN_MODE ;
        String scanMode = BluetoothAdapter.EXTRA_SCAN_MODE;
        int currentScanMode = intent.getIntExtra(scanMode, −1);
        int prevMode = intent.getIntExtra(prevScanMode, −1);
        Log.d(TAG, “Scan Mode: “ + currentScanMode +
        “. Previous: “ + prevMode);
       }
  },
 new IntentFilter(BluetoothAdapter.ACTION_SCAN_MODE_CHANGED));
```

- **Discovering Remote Devices**

- In this section you’ll learn how to initiate discovery from your local Bluetooth Adapter to find discoverable devices nearby. The discovery process can take some time to complete (up to 12 seconds). During this time, performance of your Bluetooth Adapter communications will be seriously degraded. Use the techniques in this section to check and monitor the discovery status of the Bluetooth Adapter, and avoid doing high-bandwidth operations (including connecting to a new remote Bluetooth Device) while discovery is in progress.
- You can check if the local Bluetooth Adapter is already performing a discovery scan by using the isDiscovering method.

- To initiate the discovery process, call startDiscovery on the Bluetooth Adapter:

```
if (bluetooth.isEnabled())
  bluetooth.startDiscovery();
```

- To cancel a discovery in progress, call cancelDiscovery.
- of discovery as well as remote devices discovered during the scan.

The discovery process is asynchronous. Android uses broadcast Intents to notify you of the start and end You can monitor changes in the discovery process by creating Broadcast Receivers to listen for the ACTION_DISCOVERY_STARTED and ACTION_DISCOVERY_FINISHED Broadcast Intents:

```
BroadcastReceiver discoveryMonitor = new BroadcastReceiver() {
    String dStarted = BluetoothAdapter.ACTION_DISCOVERY_STARTED;
    String dFinished = BluetoothAdapter.ACTION_DISCOVERY_FINISHED;
        @Override
        public void onReceive(Context context, Intent intent) {
        if (dStarted.equals(intent.getAction())) {
        // Discovery has started.
```

```
                Log.d(TAG, "Discovery Started...");
                }
                else if (dFinished.equals(intent.getAction())) {
                // Discovery has completed.
                Log.d(TAG, "Discovery Complete.");
                }
               }
        };
        registerReceiver(discoveryMonitor,
                        new IntentFilter(dStarted));
        registerReceiver(discoveryMonitor,
                        new IntentFilter(dFinished));
```

- Discovered Bluetooth Devices are returned via Broadcast Intents by means of the ACTION_FOUND broadcast action.
- **Discovering remote Bluetooth Devices**

```
private ArrayList<BluetoothDevice> deviceList =
        new ArrayList<BluetoothDevice>();

private void startDiscovery() {
        registerReceiver(discoveryResult,
                new IntentFilter(BluetoothDevice.ACTION_FOUND));
        if (bluetooth.isEnabled() && !bluetooth.isDiscovering())
             deviceList.clear();
            bluetooth.startDiscovery();
}
BroadcastReceiver discoveryResult = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String remoteDeviceName =
               intent.getStringExtra(BluetoothDevice.EXTRA_NAME);
        BluetoothDevice remoteDevice =
               intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
        deviceList.add(remoteDevice);

        Log.d(TAG, "Discovered " + remoteDeviceName);
        }
};
```

- Each Bluetooth Device object returned through the discovery broadcasts represents a remote Bluetooth Device discovered.

- **Managing Wi-Fi connectivity using WiFiManager**
- **WifiManager-**
- The WifiManager, which represents the Android Wi-Fi Connectivity Service, can be used to configure Wi-Fi network connections, manage the current Wi-Fi connection, scan for access points, and monitor changes in Wi-Fi connectivity.
- To use the Wi-Fi Manager, your application must have uses-permissions for accessing and changing the Wi-Fi state included in its manifest:

<uses-permission android:name=”android.permission.ACCESS_WIFI_STATE”/>
<uses-permission android:name=”android.permission.CHANGE_WIFI_STATE”/>
Access the Wi-Fi Manager using the getSystemService method, passing in the Context.WIFI_
SERVICE constant.

- Accessing the Wi-Fi Manager
  String service = Context.WIFI_SERVICE;
  WifiManager wifi = (WifiManager)getSystemService(service);

- You can use the Wi-Fi Manager to enable or disable your Wi-Fi hardware using the setWifiEnabled method, or to request the current Wi-Fi state using the getWifiState or isWifiEnabled methods.

- Wi-Fi Manager

- Monitoring and changing Wi-Fi state

```
if (!wifi.isWifiEnabled())
        if (wifi.getWifiState() != WifiManager.WIFI_STATE_ENABLING)
         wifi.setWifiEnabled(true);
```

- The following sections begin with tracking the current Wi-Fi connection status and monitoring changes in signal strength. Later you’ll also learn how to scan for and connect to specific access points.

- **Monitoring Wi-Fi Connectivity**
- In most cases it’s best practice to use the Connectivity Manager to monitor changes in Wi-Fi connectivity; however, the Wifi Manager does broadcast Intents whenever the connectivity status of the Wi-Fi network changes, using an action from one of the following constants defined in the WifiManager class:

- **WIFI_STATE_CHANGED_ACTION** — Indicates that the Wi-Fi hardware status has changed, moving between enabling, enabled, disabling, disabled, and unknown. It includes two extra values keyed on EXTRA_WIFI_STATE and EXTRA_PREVIOUS_STATE that provide the new and previous Wi-Fi states, respectively.
- **SUPPLICANT_CONNECTION_CHANGE_ACTION** — This Intent is broadcast whenever the connection state with the active supplicant (access point) changes. It is fired when a new connection is established or an existing connection is lost, using the EXTRA_NEW_STATE Boolean extra, which returns true in the former case.
- **NETWORK_STATE_CHANGED_ACTION** — Fired whenever the Wi-Fi connectivity state changes. This Intent includes two extras: the first, EXTRA_NETWORK_INFO, includes a NetworkInfo object that details the current network state, whereas the second, EXTRA_BSSID, includes the BSSID of the access point you're connected to.

- **RSSI_CHANGED_ACTION** — You can monitor the current signal strength of the connected Wi-Fi network by listening for the RSSI_CHANGED_ACTION Intent. This Broadcast Intent includes an integer extra, EXTRA_NEW_RSSI, that holds the current signal strength. To use this signal strength, you should use the calculateSignalLevel static method on the Wi-Fi Manager to convert it to an integer value on a scale you specify.

- **Managing Wi-Fi Configurations**
- You can use the Wi-Fi Manager to manage the configured network settings and control which networks to connect to. When connected, you can interrogate the active network connection to get additional details of its configuration and settings.
- Get a list of the current network configurations using getConfiguredNetworks. The list of Wi-Fi Configuration objects returned includes the network ID, SSID, and other details for each configuration.
- To use a particular network configuration, use the enableNetwork method, passing in the network ID to use and specifying true for the disableAllOthers parameter:

```
// Get a list of available configurations
List<WifiConfiguration> configurations = wifi.getConfiguredNetworks();
// Get the network ID for the first one.
if (configurations.size() > 0) {
int netID = configurations.get(0).networkId;
// Enable that network.
boolean disableAllOthers = true;
wifi.enableNetwork(netID, disableAllOthers);
}
```

- **Creating Wi-Fi Network Configurations**
- To connect to a Wi-Fi network, you need to create and register a configuration. Normally, your users would do this using the native Wi-Fi configuration settings, but there's no reason you can't expose the same functionality within your own applications or, for that matter, replace the native Wi-Fi configuration Activity entirely.

- Network configurations are stored as Wi-Fi Configuration objects. The following is a nonexhaustive
- list of some of the public fields available for each Wi-Fi configuration:

- BSSID — The BSSID for an access point.
- SSID — The SSID for a particular network.
- networkId — A unique identifier used to identify this network configuration on the current device.
- priority — The network configuration's priority to use when ordering the list of potential access points to connect to.
- status — The current status of this network connection, which will be one of the following:
    - WifiConfiguration.Status.ENABLED, WifiConfiguration.Status.DISABLED, or
    - WifiConfiguration.Status.CURRENT

- The Wi-Fi Configuration object also contains the supported authentication techniques, as well as the keys used previously to authenticate with this access point.

- **Threads and Thread Handlers**
- **Introduction to Threads**
- Threads are the cornerstone of any multitasking operating system and can be thought of as mini-processes running within a main process, the purpose of which is to enable at least the appearance of parallel execution paths within applications.

- **The Application Main Thread**

- When an Android application is first started, the runtime system creates a single thread in which all application components will run by default. This thread is generally referred to as the main thread.
- The primary role of the main thread is to handle the user interface in terms of event handling and interaction with views in the user interface. Any additional components that are started within the application will, by default, also run on the main thread.
- Any component within an application that performs a time consuming task using the main thread will cause the entire application to appear to lock up until the task is completed.
- This will typically result in the operating system displaying an "Application is unresponsive" warning to the user. Clearly, this is far from the desired behavior for any application. In such a situation, this can be avoided simply by launching the task to be performed in a separate thread, allowing the main thread to continue unhindered with other tasks.

- **Thread Handlers**

- Clearly one of the key rules of application development is never to perform time-consuming operations on the main thread of an application.
- The second, equally important rule is that the code within a separate thread must never, under any circumstances, directly update any aspect of the user interface. Any changes to the user interface must always be performed from within the main thread.
- The reason for this is that the Android UI toolkit is not thread-safe. Attempts to work with non thread-safe code from within multiple threads will typically result in intermittent problems and unpredictable application behavior.
- In the event that the code executing in a thread needs to interact with the user interface, it must do so by synchronizing with the main UI thread. This is achieved by creating a handler within the main thread, which, in turn, receives messages from another thread and updates the user interface accordingly.

- **A Basic Threading Example**

- The first step will be to highlight the risks involved in not performing time-consuming tasks in a separate thread from the main thread. Begin, therefore, by creating a new Android project named ThreadExample containing a single blank activity named ThreadExampleActivity with the Layout and Fragment names set to activity_thread_example and fragment_thread_example.
- Load the fragment_thread_example.xml file for the project into the Graphical Layout tool. Right-click on the TextView component, select the Assign ID… menu option and change the ID for the view to myTextView. Next, click and drag the TextView so that it is positioned in the center of the display canvas.
- Add a Button view to the user interface, positioned directly beneath the existing TextView object as illustrated in Figure 1:

Figure 1.

- Right-click on the button view and select the Edit Text… menu option. Create a new string resource named button_text containing the text "Press Me". Right click on the button view a second time, this time selecting the Other Properties -> All by Name -> onClick… menu option. In the resulting dialog, enter buttonClick as the method name.
- Once completed, the XML file content should be similar to the following listing:

```
<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=".ThreadExampleActivity" >

<TextView
   android:id="@+id/myTextView"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_centerHorizontal="true"
   android:layout_centerVertical="true"
   android:text="@string/hello_world" />

<Button
   android:id="@+id/button1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_below="@+id/myTextView"
   android:layout_centerHorizontal="true"
   android:layout_marginTop="48dp"
   android:onClick="buttonClick"
   android:text="@string/button_text" />

</RelativeLayout>
```

- Save the user interface design file before proceeding.
- Next, load the ThreadExampleActivity.java file into an editing panel and add code to implement the buttonClick() method which will be called when the Button view is touched by the user. Since the goal here is to demonstrate the problem of performing lengthy tasks on the main thread, the code will simply pause for 20 seconds before displaying different text on the TextView object:

```
package com.example.threadexample;

import android.app.Activity;
import android.app.ActionBar;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.os.Build;
public class ThreadExample extends Activity {
.
.
.
   public void buttonClick(View view)
   {
         long endTime = System.currentTimeMillis() + 20*1000;
```

```
                while (System.currentTimeMillis() < endTime) {
                    synchronized (this) {
                    try {
                            wait(endTime - System.currentTimeMillis());
                        } catch (Exception e) {
                            }
                    }
                }
                TextView myTextView =
(TextView)findViewById(R.id.myTextView);
                myTextView.setText("Button Pressed");
    }
}
```

- With the code changes complete, run the application on either a physical device or an emulator. Once the application is running, touch the Button, at which point the application will appear to freeze. It will, for example, not be possible to touch the button a second time and in some situations the operating system will, as demonstrated in Figure 2, report the application as being unresponsive:

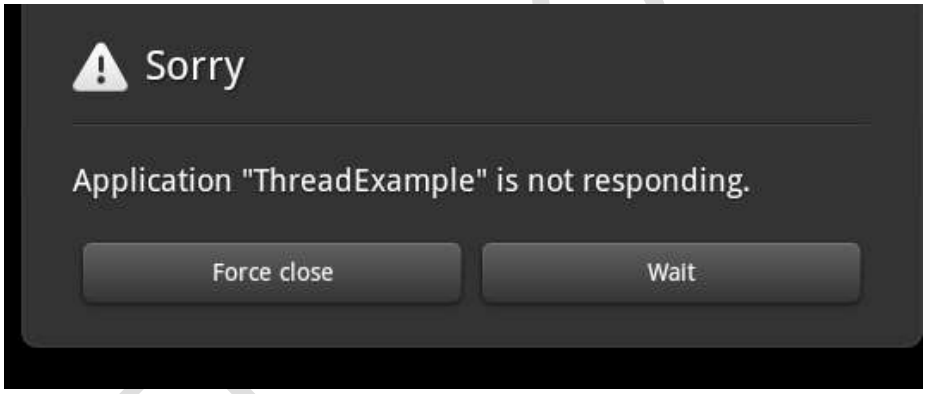


Figure 2.

- Clearly, anything that is going to take time to complete within the buttonClick() method needs to be performed within a separate thread.

- Additionally, the Android UI toolkit is *not* thread-safe. So, you must not manipulate your UI from a worker thread—you must do all manipulation to your user interface from the UI thread. Thus, there are simply two rules to Android's single thread model:

  1. Do not block the UI thread
  2. Do not access the Android UI toolkit from outside the UI thread.

- **Worker threads**

  - Because of the single threaded model described above, it's vital to the responsiveness of your application's UI that you do not block the UI thread. If you have operations to perform that are not instantaneous, you should make sure to do them in separate threads ("background" or "worker" threads).

- However, note that you cannot update the UI from any thread other than the UI thread or the "main" thread.
- To fix this problem, Android offers several ways to access the UI thread from other threads. Here is a list of methods that can help:

➢ Activity.runOnUiThread(Runnable)

➢ View.post(Runnable)

➢ View.postDelayed(Runnable, long)

```
public void onClick(View v) {
   new Thread(new Runnable() {
      public void run() {
         // a potentially  time consuming task
         final Bitmap bitmap =
               processBitMap("image.png");
         mImageView.post(new Runnable() {
            public void run() {
               mImageView.setImageBitmap(bitmap);
            }
         });
      }
   }).start();
}
```

- This implementation is thread-safe: the background operation is done from a separate thread while the ImageView is always manipulated from the UI thread.
- However, as the complexity of the operation grows, this kind of code can get complicated and difficult to maintain. To handle more complex interactions with a worker thread, you might consider using a Handler in your worker thread, to process messages delivered from the UI thread. Perhaps the best solution, is to extend the AsyncTask class, which simplifies the execution of worker thread tasks that need to interact with the UI.
- **Using AsyncTask**
- AsyncTask allows you to perform asynchronous work on your user interface. It performs the blocking operations in a worker thread and then publishes the results on the UI thread, without requiring you to handle threads and/or handlers yourself.
- To use it, you must subclass AsyncTask and implement the doInBackground() callback method, which runs in a pool of background threads. To update your UI, you should implement onPostExecute(), which delivers the result from doInBackground() and runs in the UI thread, so you can safely update your UI. You can then run the task by calling execute() from the UI thread
- **Interprocess Communication**
- Android offers a mechanism for interprocess communication (IPC) using remote procedure calls (RPCs), in which a method is called by an activity or other application component, but executed remotely (in another process), with any result returned back to the caller.
- This entails decomposing a method call and its data to a level the operating system can understand, transmitting it from the local process and address space to the remote process

and address space, then reassembling and reenacting the call there. Return values are then transmitted in the opposite direction.

- Android provides all the code to perform these IPC transactions, so you can focus on defining and implementing the RPC programming interface.
- To perform IPC, your application must bind to a service, using bindService().

- **Working with Animation and Graphics**
- Android provides a variety of powerful APIs for applying animation to UI elements and drawing custom 2D and 3D graphics**.**

- **Animation**
- The Android framework provides two animation systems:
  - property animation
  - view animation.
- Both animation systems are viable options, but the property animation system, in general, is the preferred method to use, because it is more flexible and offers more features. In addition to these two systems, you can utilize Drawable animation, which allows you to load drawable resources and display them one frame after another.

- **Property Animation**
- Introduced in Android 3.0 (API level 11), the property animation system lets you animate properties of any object, including ones that are not rendered to the screen. The system is extensible and lets you animate properties of custom types as well.

- **View Animation**
- View Animation is the older system and can only be used for Views. It is relatively easy to setup and offers enough capabilities to meet many application's needs.

- **Drawable Animation**
- Drawable animation involves displaying Drawable resources one after another, like a roll of film. This method of animation is useful if you want to animate things that are easier to represent with Drawable resources, such as a progression of bitmaps.

- **2D and 3D Graphics**
- When writing an application, it's important to consider exactly what your graphical demands will be.
- Varying graphical tasks are best accomplished with varying techniques.
- For example, graphics and animations for a rather static application should be implemented much differently than graphics and animations for an interactive game. Here, we'll discuss a few of the options you have for drawing graphics on Android and which tasks they're best suited for.

- **Canvas and Drawables**
- Android provides a set of View widgets that provide general functionality for a wide array of user interfaces. You can also extend these widgets to modify the way they look or behave. In addition, you can do your own custom 2D rendering using the various drawing methods contained in the Canvas class or create Drawable objects for things such as textured buttons or frame-by-frame animations.
- **ShapeDrawable**
- This is a Drawable object that draws a primitive geometric shape and applies a limited set of graphical effects on that shape. They are very useful for things such as customizing Buttons or setting the background of TextViews.
- **ShapeDrawable**
- A Drawable object that draws primitive shapes. A ShapeDrawable takes a Shape object and manages its presence on the screen. If no Shape is given, then the ShapeDrawable will default to a RectShape.
- This object can be defined in an XML file with the `<shape>` element.
- **Hardware Acceleration**
- Beginning in Android 3.0 (API level 11), the Android 2D rendering pipeline supports hardware acceleration, meaning that all drawing operations that are performed on a View's canvas use the GPU. Because of the increased resources required to enable hardware acceleration, your app will consume more RAM.
- Hardware acceleration is enabled by default if your Target API level is >=14, but can also be explicitly enabled. If your application uses only standard views and Drawables, turning it on globally should not cause any adverse drawing effects.
- However, because hardware acceleration is not supported for all of the 2D drawing operations, turning it on might affect some of your custom views or drawing calls.
- Problems usually manifest themselves as invisible elements, exceptions, or wrongly rendered pixels. To remedy this, Android gives you the option to enable or disable hardware acceleration at multiple levels.
- You can control hardware acceleration at the following levels:
- Application
- Activity
- Window
- View
    - **Application level**

In your Android manifest file, add the following attribute to the `<application>` tag to enable hardware acceleration for your entire application:

<application android:hardwareAccelerated="true" ...>

- Activity level
- If your application does not behave properly with hardware acceleration turned on globally, you can control it for individual activities as well. To enable or disable hardware acceleration at the activity level, you can use the `android:hardwareAccelerated` attribute for the `<activity>` element. The following example enables hardware acceleration for the entire application but disables it for one activity:

```
<application android:hardwareAccelerated="true">
   <activity ... />
   <activity android:hardwareAccelerated="false" />
</application>
```

- **Window level**
- If you need even more fine-grained control, you can enable hardware acceleration for a given window with the following code:

```
getWindow().setFlags(
   WindowManager.LayoutParams.FLAG_HARDWARE_ACCELERATED,
   WindowManager.LayoutParams.FLAG_HARDWARE_ACCELERATED);
```

- **View level**
- You can disable hardware acceleration for an individual view at runtime with the following code:

```
myView.setLayerType(View.LAYER_TYPE_SOFTWARE, null);
```

- **OpenGL**

- Android supports OpenGL ES 1.0 and 2.0, with Android framework APIs as well as natively with the Native Development Kit (NDK).
- Using the framework APIs is desireable when you want to add a few graphical enhancements to your application that are not supported with the Canvas APIs, or if you desire platform independence and don't demand high performance.
- There is a performance hit in using the framework APIs compared to the NDK, so for many graphic intensive applications such as games, using the NDK is beneficial (It is important to note though that you can still get adequate performance using the framework APIs. For example, the Google Body app is developed entirely using the framework APIs).
- OpenGL with the NDK is also useful if you have a lot of native code that you want to port over to Android.

• INTRODUCING CLOUD TO DEVICE MESSAGING

- The Cloud to Device Messaging (C2DM) service provides an alternative to regularly polling a server for updates; instead, your server can "push" messages to a specifi c client.
- The frequency of your application's background polling can have a dramatic impact on the host device's battery life, so you always need to compromise between data freshness and the resulting power drain.

- Introduced in Android 2.2 (API level 8), C2DM allows you to eliminate background polling, and instead have your server notify a particular device when new data is available for it.
- On the client side, C2DM is implemented using Intents and Broadcast Receivers. As a result, your application does not need to be active in order to receive C2DM messages. On the server side, C2DM messages area transmitted from your server to each target device by way of the C2DM service.
- The C2DM service maintains an open TCP/IP connection with each device, allowing it to transmit information instantly whenever required. The C2DM service takes care of maintaining and restoring that connection, queuing messages, and retrying failed deliveries.

- In the following sections you'll learn how to:

- Register each device on which your application is running with the Android C2DM server.
- Notify your server of the C2DM address of your application running on a particular device.
- Transmit messages from your server to the C2DM service.
- Receive your server messages within your application once they're relayed through the

- **C2DM service**.

- C2DM is a Google service, so its documentation is available at http://code.google.com/android/c2dm/.

- **C2DM Restrictions**

- C2DM is not designed as a blanket replacement for background polling. It is best used in situa-tions where only one device (or a small, distinct group of devices) requires updates at any given time — such as email or voicemail services.
- The real-time nature of each push makes C2DM an ideal alternative for situations where the updates are unlikely to be at predictable intervals; however, successful message delivery, latency, and delivery order are not guaranteed. As a result, you should not rely on C2DM for critical messages or where timeliness is important. It's also good practice to implement a traditional polling mechanism at long intervals as a fail-safe.
- The transmitted messages should be lightweight and are limited to 1024 bytes. They should carry very little payload, instead containing only the information required for the client application to efficiently query your server for the data directly.
- C2DM is based around existing Google services and requires Google Play to be installed on the device, and for the user to have a Google account configured.
- At the time of writing, new C2DM accounts receive a development quota of up to 200,000 messages per day. If your production requirements demand more, you can request an increase — details on that process will be emailed to you after you sign up.

- **Signing Up to Use C2DM**

- The first step is to view and agree to the terms of the C2DM service at http://code.google.com/android/c2dm/signup.html.

- As part of the registration process, you will be asked for your application's package name, an estimate of the total number of daily messages you plan to send, and the estimated peak queries per second (QPS). The C2DM team uses this information to help identify applications that may need to be granted larger quotas.
- You will also be asked to supply three email addresses: your contact details, an escalation email address for urgent issues, and a role account that will be used to authenticate with the C2DM service and send messages from your server.
- The role account should be a Google account used specifically for use with the C2DM service.
- Because you will be providing a server with authentication details for this account, it's good practice to create a new account rather than use a personal Gmail or Google Play account.

- After receiving confi rmation that your account has been enabled for sending C2DM messages, you can update your application to register itself, and each device it's running on, with the C2DM service.

- **Registering Devices with a C2DM Server**

- In order for your application to receive C2DM messages, it must fi rst register each installed instance of itself with the C2DM service. Start by adding a com.google.android.c2dm.permission.RECEIVE uses-permission node to your manifest:

  <uses-permission android:name=”com.google.android.c2dm.permission.RECEIVE” />
- You should also define (and request) a signature-level permission that restricts the receipt of C2DM messages targeted at your application to applications signed with the same key:

```
<permission android:name=”com.example.myapp.permission.C2D_MESSAGE”
 android:protectionLevel=”signature” />
<uses-permission android:name=”com.example.myapp.permission.C2D_MESSAGE” />
```

- You should also defi ne (and request) a signature-level permission that restricts the receipt of C2DM messages targeted at your application to applications signed with the same key:

```
<permission android:name=”com.example.myapp.permission.C2D_MESSAGE”
android:protectionLevel=”signature” />
<uses-permission android:name=”com.example.myapp.permission.C2D_MESSAGE” />
```

Registering an application for C2DM is a three-step process, as shown in Figure 1.

- The process of registering your application on each device with the C2DM service associates each installed instance of your application with the device on which it is installed. Once registered, the C2DM service returns a registration ID that uniquely identifi es that particular installation. Your application should send that ID, along with a way to identify each installation (typically a username or anonymous UUID) to your server.

- The process of registering your application on each device with the C2DM service associates each installed instance of your application with the device on which it is installed. Once registered, the C2DM service returns a registration ID that uniquely identifi es that particular installation. Your application should send that ID, along with a way to identify each installation (typically a username or anonymous UUID) to your server.

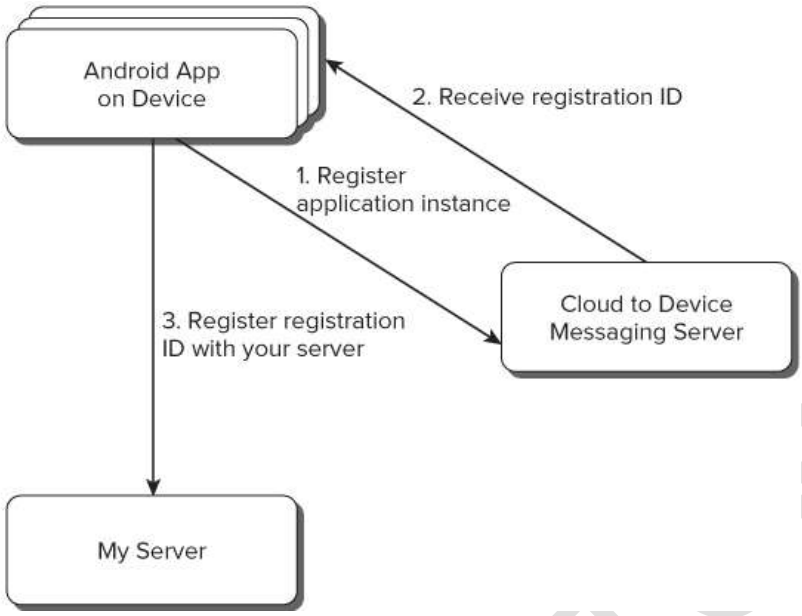


Figure 1.

- Begin by starting a Service using an Intent that includes the com.google.android.c2dm.intent.REGISTER action. It must include extras to identify your application and specify your sender account. As shown in 1.

1. Registering an application instance with the C2DM server

```
Intent registrationIntent =
new Intent("com.google.android.c2dm.intent.REGISTER");
registrationIntent.putExtra("app",
PendingIntent.getBroadcast(this, 0, new Intent(), 0));
registrationIntent.putExtra("sender",
        "myC2DMaccount@gmail.com");
startService(registrationIntent);
```

- Your application is identifi ed using the app extra key and a Pending Broadcast Intent that will be populated by the C2DM service to send your application messages when they are received.
- The sender extra is used to specify the role account you registered when signing up for C2DM, and will be used by your server to transmit messages.
- The platform will transmit this information to the C2DM server, which will return a registration ID. To receive this, you need to register a Broadcast Receiver that listens for the

com.google.android.c2dm.intent.REGISTRATION action, requires the com.google.android.c2dm.permission.SEND permission, and includes your application package name as a category, as shown in point 2.

2. Listening for C2DM registration IDs

```
<receiver
android:name=".MyC2DMReceiver"
android:permission="com.google.android.c2dm.permission.SEND">

<intent-filter>
<action
android:name="com.google.android.c2dm.intent.REGISTRATION"
/>
<category android:name="com.mypackage.myc2dmAppName"/>
</intent-filter>
</receiver>
```

- The registration ID for each application/device pair may be changed at any time, so it's important that your application continue listening for new REGISTRATION Broadcast Intents.
- The registration ID itself is included in the registration_id extra, as shown in point 3. If the registration process fails, the error code will be included as an error extra, and successful deregistration requests will be signaled using the unregistered extra.

3. Extracting the C2DM registration ID

```
public void onReceive(Context context, Intent intent) {
if (intent.getAction().equals(
 "com.google.android.c2dm.intent.REGISTRATION")) {
String registrationId = intent.getStringExtra("registration_id");
String error = intent.getStringExtra("error");
String unregistered = intent.getStringExtra("unregistered");
if (error != null) {
// Registration failed.
if (error.equals("SERVICE_NOT_AVAILABLE")) {
Log.e(TAG, "Service not available.");
// Retry using exponential back off.
}
else if (error.equals("ACCOUNT_MISSING")) {
Log.e(TAG, "No Google account on device.");
// Ask the user to create / add a Google account
}
else if (error.equals("AUTHENTICATION_FAILED")) {
Log.e(TAG, "Incorrect password.");
// Ask the user to re-enter their Google account password.
}
else if (error.equals("TOO_MANY_REGISTRATIONS")) {
Log.e(TAG, "Too many applications registered.");
// Ask the user to unregister / uninstall some applications.
```

```
}
else if (error.equals("INVALID_SENDER")) {
Log.e(TAG, "Invalid sender account.");
// The sender account specified has not been registered
// with the C2DM server.
}
else if (error.equals("PHONE_REGISTRATION_ERROR")) {
Log.e(TAG, "Phone registration failed.");
// The phone doesn't currently support C2DM.
}
} else if (unregistered != null) {
// Unregistration complete. The application should stop
// processing any further received messages.
Log.d(TAG, "Phone deregistration completed successfully.");
} else if (registrationId != null) {
Log.d(TAG, "C2DM registration ID received.");
// Send the registration ID to your server.
}
}
}
```

- The received registration ID becomes the address your server uses to target a message at this particular device/application instance. Accordingly, you need transmit this ID to your server, along with an identifi er it can use to identify the user associated with this installation. This will allow you to look up the device address based on a particular user in order to transmit data to him or her. In the case of email, this might be the username; for voicemail, the phone number; or for a game, a generated UUID.
- It's good practice to create a server-side hash to simplify the lookup. Keep in mind that a single user may have multiple devices, so you may need to include a collision algorithm that determines which device should receive the message (or if multiple devices should receive them).
- Also remember that the registration ID may subsequently change, so be sure to retransmit the identifi er/ID pair should that happen.You can unregister a device by calling startService, passing in an Intent that uses the com.google.android.c2dm.intent.UNREGISTER action, and including an app extra that uses a Pending Intent to identify your application:

```
PendingIntent pi =
PendingIntent.getBroadcast(this, 0, new Intent(), 0);
Intent unregister =
            new Intent("com.google.android.c2dm.intent.UNREGISTER");
unregister.putExtra("app", pi);
startService(unregister);
```

- **Sending C2DM Messages to Devices**
- Once you've recorded a particular device's registration ID on your server, it's possible for it to transmit messages to that device. Sending messages is a two-step process, as shown in Figure 2

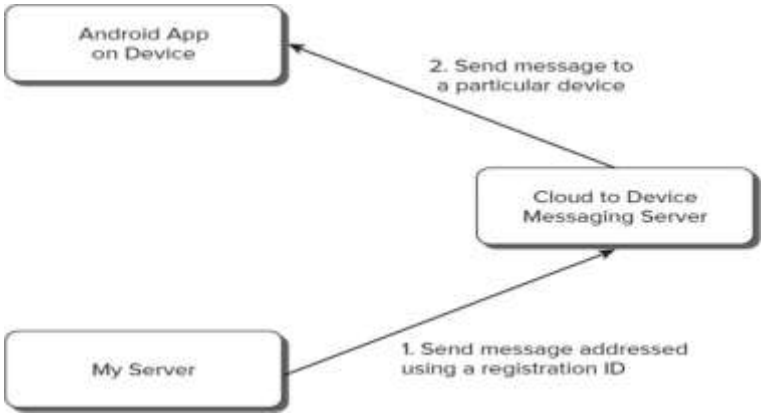


Figure 2.

- Creating the server-side implementation of C2DM is beyond the scope of this book. If your server is an AppEngine application, the Chrome 2 Phone project (http://code.google.com/p/chrometophone/) includes a server-side implementation that can be used to greatly simplify the process of authentication and message transmission to the C2DM service.
- Your server transmits a message to the C2DM service using POST requests to https://android.apis.google.com/c2dm/send that include the following parameters:
  - registration_id — The address of the target device/application pair.
  - collapse_key — When the target device is offl ine, messages transmitted to it will be queued. By specifying a collapse key, you can effectively collapse that queue, causing each message with the same key to override the previous so that only the last message gets sent to the target device.
  - data.[key] — Payload data in the form of key/value pairs. They will be passed in to your application as extras within the C2DM message Intent, using the keys you specify. Each C2DM message is limited to 1024 bytes, so payload data should be kept to the bare minimum — typically only the information required for the client to perform an efficient lookup.

- **Receiving C2DM Messages**
- After your server transmits messages to the C2DM service, they are, in turn, sent to the device to which they are addressed. The target device then delivers each message to its recipient application as a Broadcast Intent.
- To receive these Intents, you must register a Broadcast Receiver that includes the com.google.android.c2dm.permission.SEND permission, a fi lter for the com.google.android.c2dm.intent.RECEIVE action, and the category set to the application's package name, as shown in point 4.

4.Registering to receive C2DM messages

```
<receiver
android:name=".C2DMMessageReceiver"
android:permission="com.google.android.c2dm.permission.SEND">
<intent-filter>
<action
android:name="com.google.android.c2dm.intent.RECEIVE"/>
<category android:name="com.mypackage.myc2dmAppName"/>
</intent-filter>
</receiver>
```

- Within the associated Broadcast Receiver implementation, you can extract any extras using the keys you specifi ed when sending the associated server message, as shown in point 5.

5. Extracting C2DM message details

```
public void onReceive(Context context, Intent intent) {
if (intent.getAction().equals(
 "com.google.android.c2dm.intent.RECEIVE")) {
Bundle extras = intent.getExtras();
// Extract any extras included in the server messages.
int newVoicemailCount = extras.getInt("VOICEMAIL_COUNT", 0);
}
}
```

- Due to the payload data limit, it's generally considered good practice to include as little payload data as possible and to use an incoming C2DM message as a tickle to indicate that the application should perform a server update.

- **Publishing Applications**

- After creating your Android Developer Profi le, you are ready to upload your application. Select the Upload Application button on the Android Developer Console. You'll be prompted to upload your signed release package.
- The package name (not the file name) must be unique. Google Play uses application package names as unique identifi ers and will not allow you to upload a duplicate package name.
- After the application is uploaded, you'll be asked to enter your application's assets and listing details, as shown in Figure 1 and 2.

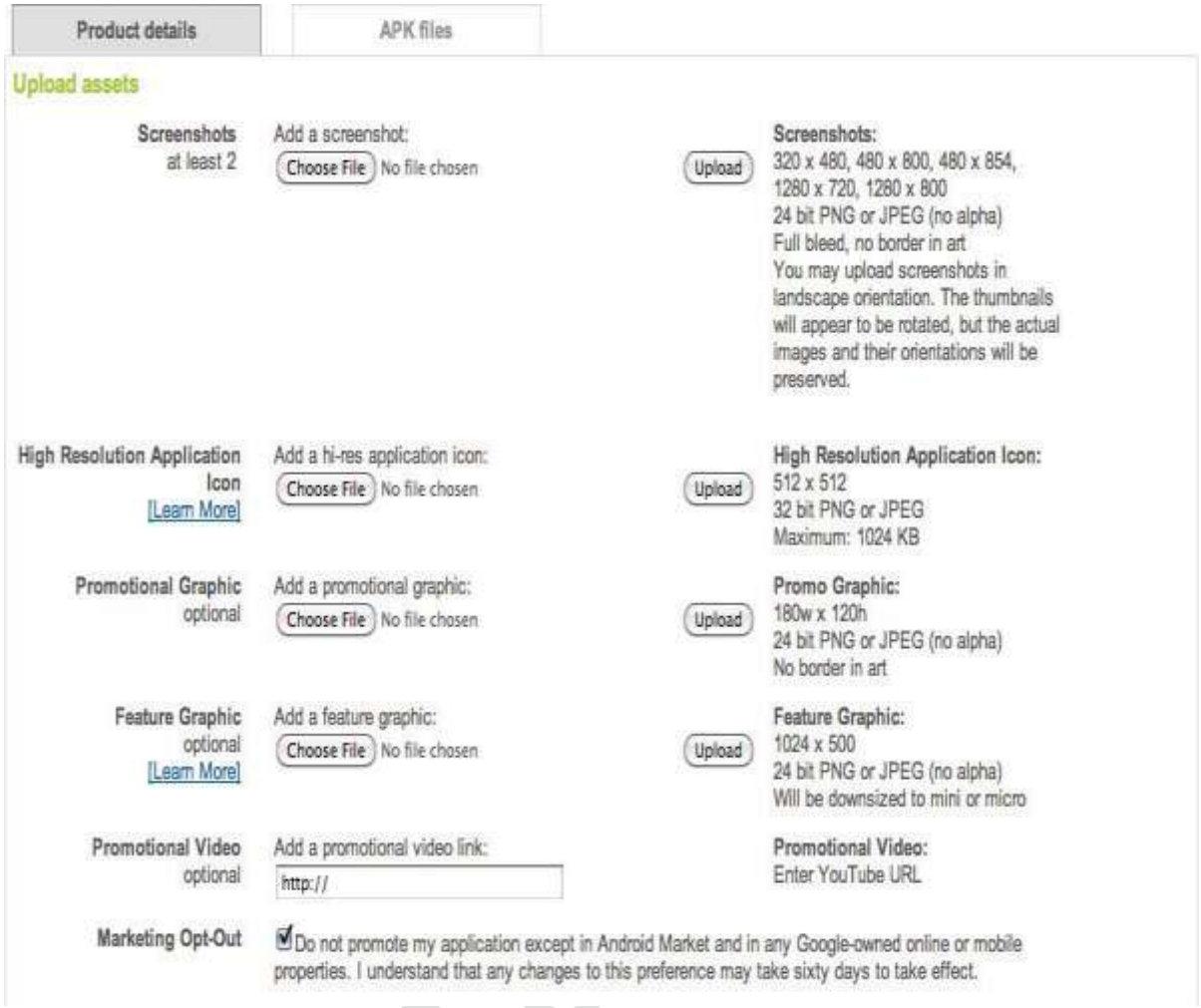


Figure 1.

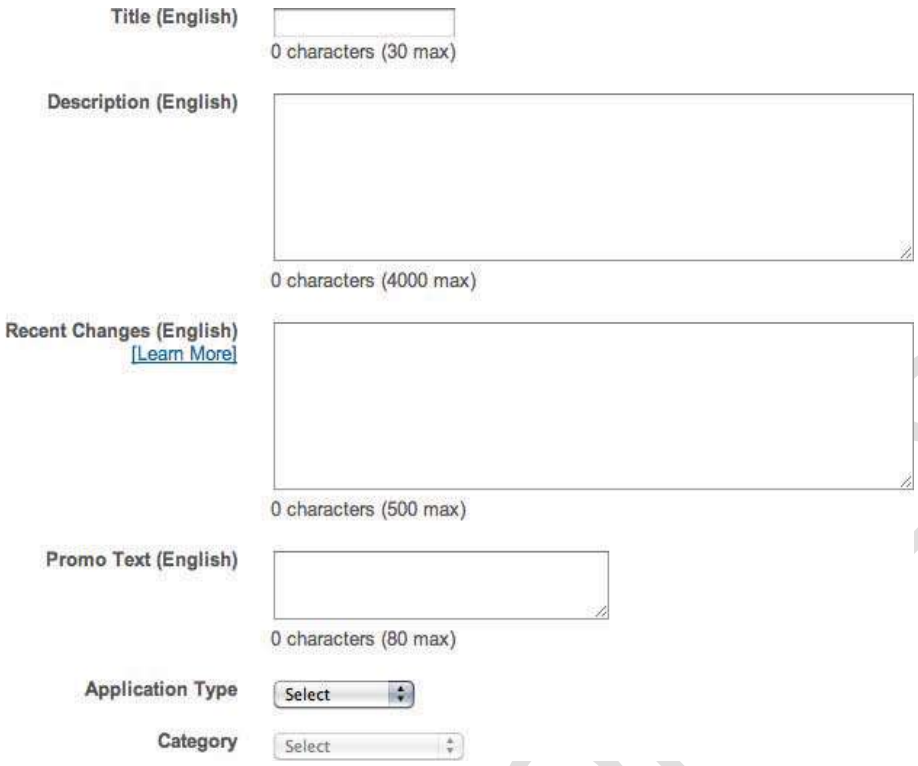


Figure 2.

- It's important that you supply all the assets available — even those that may be listed as optional. Each asset is used throughout Google Play, including the website, Google Play Store clients, and promotional campaigns. Not including some assets may prevent your application from being featured or promoted.
- The title, description, and category determine how and where your application will be displayed within Google Play. It's important to provide high quality, descriptive titles and application descriptions in this section to make it easier for users to discover your application and make an informed choice on its suitability.
- Do not engage in keyword stuffi ng or other SEO spam in your title or description, as doing so will likely result in your application being suspended.
- It's also possible to supply the title and description in multiple languages.
- The listing page also lets you specify the availability of your application, providing mechanisms to set its maturity level and the countries in which you want it to be available.
- Finally, you can supply application-specifi c contact details for users of your applications, as shown in Figure 3.
- These details will be published alongside your application's listing in Google Play, so the email and phone number provided should point to a managed support queue rather than to your personal email address.

- When your listing details are complete, click the Publish button. Your application will be live and available for download almost immediately.

Figure 3.