VARDHAMAN COLLEGE OF ENGINEERING
(AUTONOMOUS)
Shamshabad – 501 218, Hyderabad

DEPARTMENT OF INFORMATION TECHNOLOGY

MOBILE APPLICATION DEVELOPMENT

Compiled By

B.Ravinder Goud
Asst.Professor-IT
Dr.M.Gopichand
HOD-IT

Course Code: **A3611**

UNIT - I **(10 Lectures)**

A brief history of Mobile, Types of mobile phone generations, The Mobile Ecosystem, Types of Mobile Applications, Mobile Information Architecture

Android Versions, Features of Android, Android Architecture, Installing Android SDK Tools, Configuring Android in Eclipse IDE, Android Development Tools (ADT), Creating Android Virtual Devices (AVD)

A brief history of Mobile

Mobile devices are no longer simple voice communication devices. They have become a medium to create voice, music, text, video, and image communications. Importantly, these various interactions can be created and shared on demand by the mobile user. In addition to communication methods, mobile devices are also a tool used to access the Internet, view television and movies, interact with GPS (Global Positioning System), play games, and read and respond to barcode and augmented reality messages. The reach and functionality of mobile devices depends on their underlying network infrastructure and the capabilities of the mobile device or handset.

A mobile phone is a portable telephone that can make and receive calls over a radio frequency carrier while the user is moving within a telephone service area. The radio frequency link establishes a connection to the switching systems of a mobile phone operator, which provides access to the Public Switched Telephone Network (PSTN). Most modern mobile telephone services use cellular network architecture, and therefore mobile telephones are often also called **cellular telephones or cell phones**. In addition to telephony, 2000s-era mobile phones support a variety of other services, such as text messaging, MMS, email, Internet access, short-range wireless communications (infrared, Bluetooth), business applications, gaming, and digital photography. Mobile phones which offer these and more general computing capabilities are referred to as **smart phones.**

The first handheld mobile phone was demonstrated by John F. Mitchell and Martin Cooper of Motorola in 1973, using a handset weighing c. 4.4 lbs (2 kg). In 1983, the DynaTAC 8000x was the first commercially available handheld mobile phone.

Mobile Technology Networks and Infrastructure

Every mobile phone vendors/proprietary platforms like Android, Apple iPhone OS, RIM Blackberry OS, Bada OS, Symbian OS provide their own SDK (Software Development Kit) to the developer. Developers can create applications using this kit. Developers are also provided a space/place/market where they can publish their creations to the world.

Martin Cooper of Motorola made the first publicized handheld mobile phone call on a prototype DynaTAC model on April 4, 1973.

A handheld mobile radio telephone service was envisioned in the early stages of radio engineering. In 1917, Finnish inventor Eric Tigerstedt filed a patent for a "pocket-size folding telephone with a very thin carbon microphone". Early predecessors of cellular phones included analog radio communications from ships and trains. The Motorola DynaTAC 8000X. First commercially available handheld cellular mobile phone, 1984. The first handheld mobile cell phone was demonstrated by Motorola in 1973. The first commercial automated cellular network was launched in Japan by Nippon Telegraph and Telephone in 1979.

Types of mobile phone generations

First-generation (1G) systems could support far more simultaneous calls, but still used analog technology. In 1991, the second-generation (2G) digital cellular technology was launched in Finland by Radiolinja on the GSM standard.

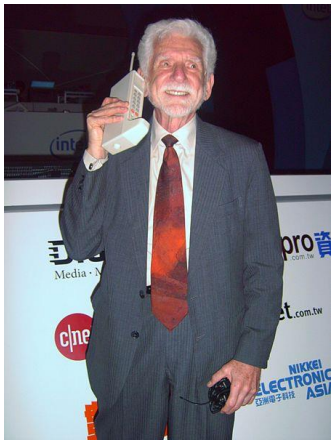
Ten years later, in 2001, the third generation (3G) was launched in Japan by NTT DoCoMo on the WCDMA standard. This was followed by 3.5G, 3G+ or turbo 3G enhancements based on the High Speed Packet Access (HSPA) family, allowing UMTS networks to have higher data transfer speeds and capacity.

By 2009, it had become clear that, at some point, 3G networks would be overwhelmed by the growth of bandwidth-intensive applications, such as streaming media. Consequently, the industry began looking to data-optimized fourth-generation technologies, with the promise of speed improvements up to ten-fold over existing 3G technologies. The first two commercially available technologies billed as 4G were the WiMAX standard, offered in North America by Sprint, and the LTE standard, first offered in Scandinavia by TeliaSonera.

MTS was used in North America until the 1980s, despite AT&T's introduction of the aptly-named Improves Mobile Telephone Service (IMTS) in 1965. The new service introduced

user dialing, removed the need for operator forwarding and used additional radio channels which increased the number of possible subscribers and calls, as well as area coverage.

In 1960 the world's first fully automated mobile telephone was introduced in Sweden. The system allowed for automated connection from a rotary handset (that's the circular dialing knob to me and you) mounted within a car, but required an operator to forward calls. The system was known as Mobile Telephone system A (MTA) and was replaced by MTB two years later.



Dr Martin Cooper, a Motorola researcher and executive made the first phone call from a handheld mobile phone on April 3, 1973.

Analog Cellular Networks or "1G"

The first generation of cellular networks paved the way to the networks we know and use today. Use of multiple cell tower sites, each connected through a network, allowed users to travel and even switch cell towers during a call. It was a revolution built on existing, analog technology with the first being built in Chicago in 1977.

Known as the Analog Mobile Phone System (AMPS), it was built by AT&T and it took the FCC 11 years to approve AT&T's initial proposal in 1971 before they were assigned the 824-894MHz range on which to operate AMPS.

Digital Cellular Networks or "2G"

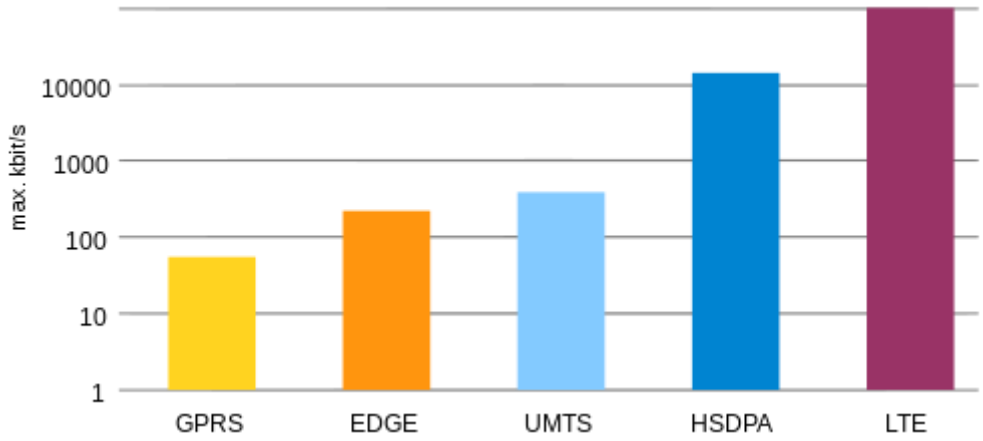
The 1990s saw the arrival of two new, digital technologies – the European GSM standard and the North American CDMA standard. Demand grew and more and more cell tower sites were built. In addition to technological improvements in batteries and internal components, this allowed for much smaller mobile devices.

Another advancement made possible by 2G was the introduction of SMS messaging, with the first computer generated SMS sent in 1992 in the UK. A year later in Finland, the first person-to-person SMS was delivered using GSM technology. As popularity grew, pre-paid mobile phones and plans emerged in the late 1990s which further popularized SMS amongst all ages. The very first download services were also introduced using 2G technology and enabled users to download ringtones.

Mobile Broadband or "3G"

NTT DoCoMo pioneered the first mobile Internet service in Japan in 1999 on existing 2G technologies, but it was soon replaced with their launch of the world's first 3G network in October 2001. Many countries followed suit in the following years including South Korea, the US and the first European 3G networks which sprang up in the UK and Italy in 2003. While 3G was still being developed a number of "2.5G" services appeared in an attempt to bring older technologies up to speed. Unfortunately speed was the lacking factor, and while technologies

like GPRS (General Packet Radio Service) and EDGE (Enhanced Data for GSM Evolution) provided improvements over standard 2G, they did not match the speed of existing 3G technologies.



3G transformed the mobile phone industry and enabled widespread mobile Internet and the transmission services like TV and Radio for the very first time. Handset manufacturers jumped on the bandwagon and smart phone use took off. By around 2005 3G had evolved a step further, leading many to coin the terms "3.5G" "turbo 3G" and "3G+" in reference to HSPDA (High Speed Downlink Packet Access), HSPA and HSPA+.

Native IP or "4G"

The first technology was WiMAX (Worldwide Interoperability for Microwave Access) , offered by Sprint in the US but perhaps the most successful has been LTE, which is popular also in North America but non-existent in some territories such as Australia. 4G marks the switch to native IP networks, bringing mobile Internet more in-line with wired home Internet connections.

The Mobile Ecosystem

Mobile Ecosystem:

Mobile Ecosystem is collection of multiple devices (mobile phones, Tablet, Phablet etc), software (operating system, development tools, testing tools etc.), companies (device manufacturers, carrier, apps stores, development/testing companies, etc.) etc., and the process by which data (sms, bank transactions etc.), is transferred/shared by a user from one device to another device or by the device itself based on some programs (Birthday, Wedding Messages, calendar).

Data (Text, MultiMedia, VOICE) sharing can be done between devices of the same operating system or different operating systems. Examples: IPhone (IOS) to Windows Phone or IPhone(IOS) to Nexus(Android) or Motorola(Android) to Nexus (Android).

Data can be also shared between multiple devices with the same operating system of the same manufacturer. Example: Apples: IOS: Iphone, Ipad, to Ipod, TV, Laptops.

Process:

- Mobile is manufactured with necessary software and applications.
- Users buy phones and subscribe to plans with carriers. If needed, buys/uploads applications for the device.
- From time to time, new applications or features are uploaded or upgraded in the device as and when the need arises.

Mobile Manufacturers:

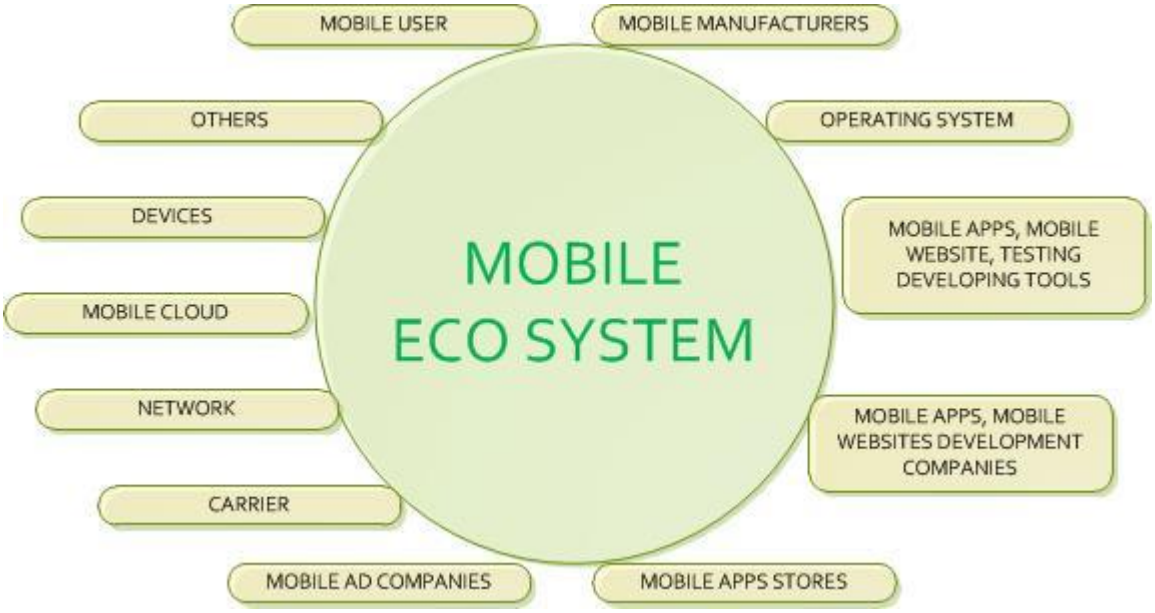
They manufacture mobiles.

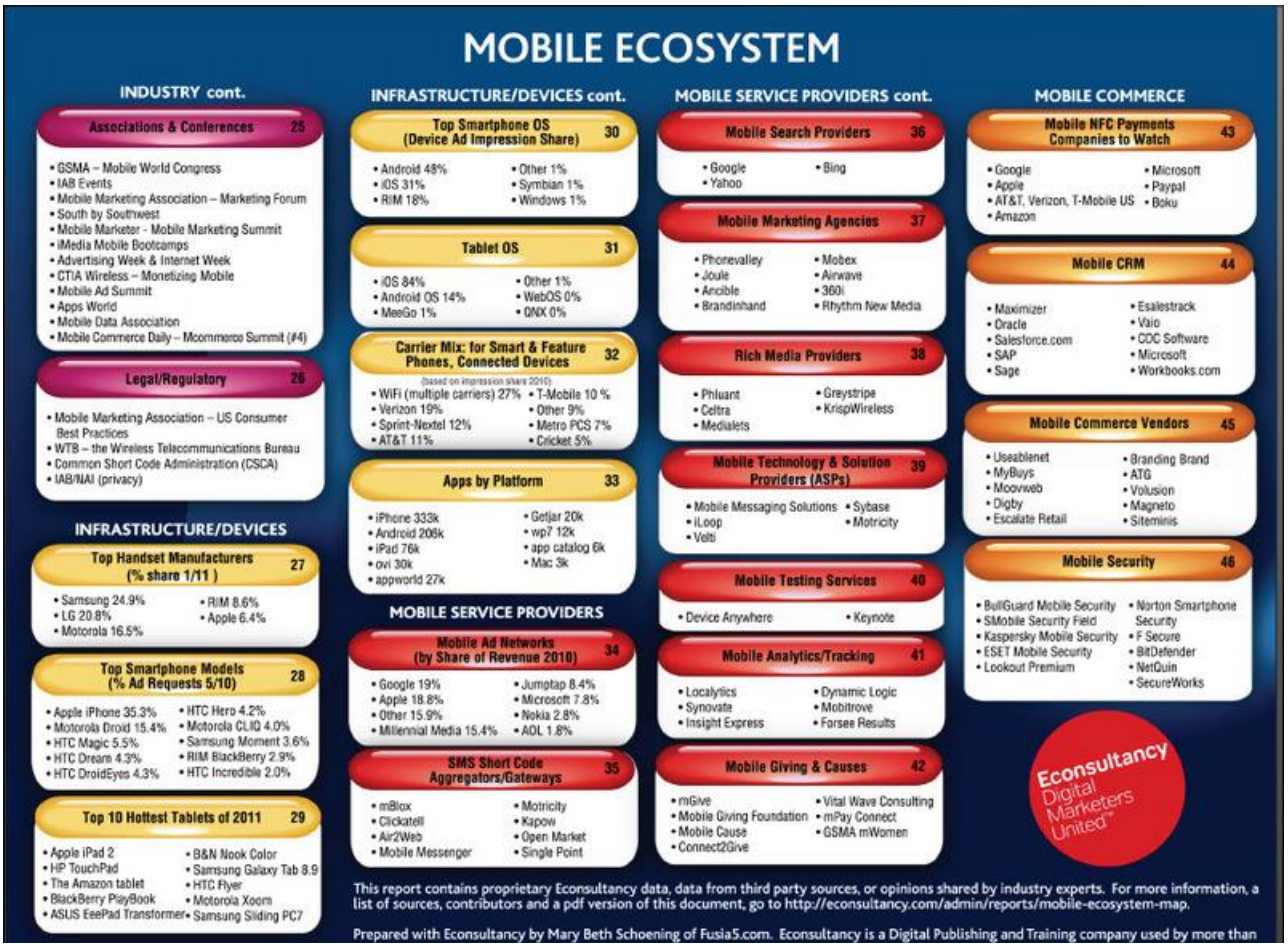
- Example: Samsung, BlackBerry, Sony, Nokia, Motorola, Windows Phone, Nexus

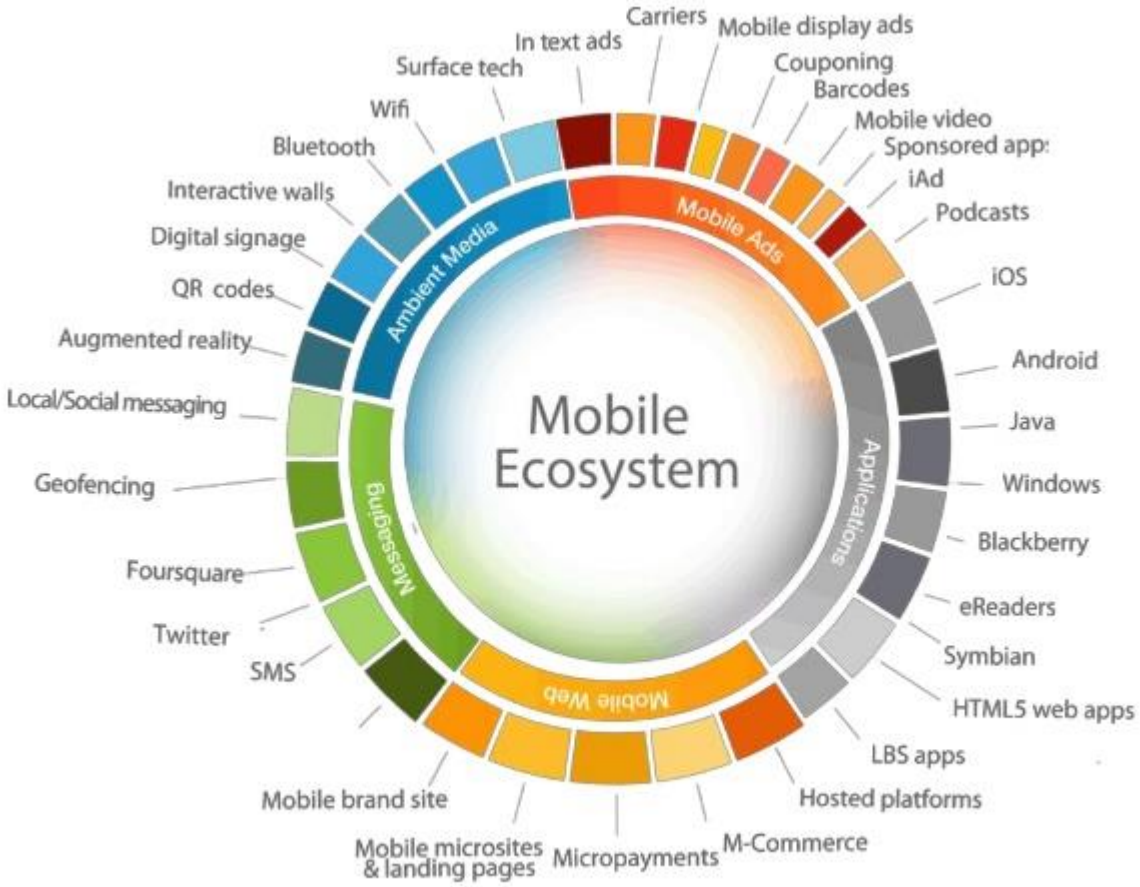
Operating System:

This is the important component of a Mobile, which controls/operates all applications that are residing on the mobile phone. Android is open source and IOS is a closed source.

- Example: IOS, Android, BlackBerry OS, Symbian, Bada etc.







Mobile Apps Development Tools:

- Android Applications are developed with Eclipse, Intellij Idea etc.
- BlackBerry Applications are developed with Eclipse etc.
- IOS Applications are developed with XCode, MonoDevelop, AppCode etc.
- Windows Phone Applications are developed with Microsoft Visual Studio etc.

Please go to relevant websites to get more information on each Tool.

Mobile Website Development Tools:

- HTML5, CSS3, JavaScript etc. are used to create mobile websites.

Mobile Apps Testing Tools:

A. Emulators:

A. Emulators:

Actual device (mobile) operations and functions are simulated on to the computer. Instead of buying several devices (Motorola, Samsung, Micromax etc), we can use emulators for functional testing. Network Connectivity, resolution testing etc cannot be tested 100% with emulators. So for testing to be 100% perfect, devices are needed and again buying several devices will be very costly. For this, you can go to Mobile Cloud environment companies and rent those mobile phone.

B. Mobile Cloud:

Companies rent mobiles and other devices virtually at hourly or weekly basis or monthly basis etc. Applications can be tested by subscribing to those companies.

Example:

- DeviceAnywhere (http://www.deviceanywhere.com/mobile-application-testing-overview.html)
- Perfecto Mobile: www.perfectomobile.com

C. Mobile Testing Tools:

Jamo Solutions, Perfecto Mobile, Device Anywhere Pro.

Mobile Stores:

Application created can be uploaded in stores after getting approval from those stores and applications can be sold. Applications can also be downloaded at free of cost or on paid basis.

- Examples: play.google.com, store.apple.com

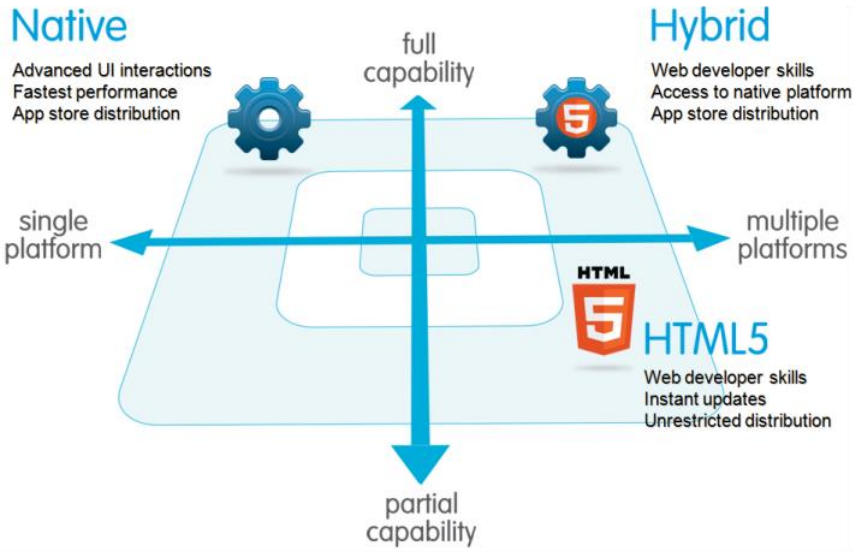
Mobile Ad Companies:

They display advertisements on the mobiles. Example: Google's AdMob.

Types of Mobile Applications

Mobile SDK supports building three types of apps:

- **Native apps** are specific to a given mobile platform (iOS or Android) using the development tools and language that the respective platform supports (e.g., Xcode and Objective-C with iOS, Eclipse and Java with Android). Native apps look and perform the best.
- **HTML5 apps** use standard web technologies—typically HTML5, JavaScript and CSS. This write-once-run-anywhere approach to mobile development creates cross-platform mobile applications that work on multiple devices. While developers can create sophisticated apps with HTML5 and JavaScript alone, some vital limitations remain at the time of this writing, specifically session management, secure offline storage, and access to native device functionality (camera, calendar, geolocation, etc.)
- **Hybrid apps** make it possible to embed HTML5 apps inside a thin native container, combining the best (and worst) elements of native and HTML5 apps.



Native Mobile Applications

In a nutshell, native apps provide the best usability, the best features, and the best overall mobile experience. There are some things you only get with native apps:

- **Multi touch** - double taps, pinch-spread, and other compound UI gestures
- **Fast graphics API** - the native platform gives you the fastest graphics, which may not be a big deal if you're showing a static screen with only a few elements, or a very big deal if you're using a lot of data and require a fast refresh.
- **Fluid animation** - related to the fast graphics API is the ability to have fluid animation. This is especially important in gaming, highly interactive reporting, or intensely computational algorithms for transforming photos and sounds.
- **Built-in components** - The camera, address book, geolocation, and other features native to the device can be seamlessly integrated into mobile apps. Another important built-in components is encrypted storage, but more about that later.
- **Ease of use** - The native platform is what people are accustomed to, and so when you add that familiarity with all of the native features they expect, you have an app that's just plain easier to use.
- **Documentation** - There are over 2500 books alone for iOS and Android development, with many more articles, blog posts, and detailed technical threads on sites like StackOverflow.

Native apps are usually developed using an Integrated Development Environment (IDE). IDEs provide tools for building debugging, project management, version control, and other tools professional developers need. While iOS and Android apps are developed using different IDEs

and languages, there's a lot of parity in the development environments, and there's not much reason to delve into the differences. Simply put, you use the tools required by the device.

You need these tools because native apps are more difficult to develop. Likewise, the level of experience required is higher than other development scenarios, you don't just cut and paste Objective-C and expect it to work. Indeed, the technological know-how of your development team is an important consideration. If you're a professional developer, you don't have to be sold on proven APIs and frameworks, painless special effects through established components, or the benefits of having your code all in one place.

HTML5 Mobile Applications

An HTML5 mobile app is basically a web page, or series of web pages, that are designed to work on a tiny screen. As such, HTML5 apps are device agnostic and can be opened with any modern mobile browser. And because your content is on the web, it's searchable, which can be a huge benefit depending on the app (shopping, for example).

An important part of the "write-once-run-anywhere" HTML5 methodology is that distribution and support is much easier than for native apps. For a native app, there are longer development and testing cycles, after which the consumer typically must log into a store and download a new version to get the latest fix.

In the last year, HTML5 has emerged as a very popular way for building mobile applications. Multiple UI frameworks are available for solving some of the most complex problems that no developer wants to reinvent. iScroll does a phenomenal job of emulating momentum style scrolling. JQuery Mobile and Sencha Touch provide elegant mobile components, with hundreds if not thousands of plugins that offer everything from carousels to super elaborate controls.

So if HTML5 apps are easier to develop, easier to support, and can reach the widest range of devices. The latest batch of browsers support hardware accelerated CSS3 animation properties, providing smooth motion for sliding panels as well transitions between screens, but even that can't match the power and flexibility of native apps. Today, it's simply not possible to capture multi-touch input events (determining when more than one finger is on

the screen) or create path-style elegance with spinout buttons and photos that hover, then drop into the right place.

Hybrid Mobile Applications

Hybrid development combines the best (or worst) of both the native and HTML5 worlds. We define hybrid as a web app, primarily built using HTML5 and JavaScript, that is then wrapped inside a thin native container that provides access to native platform features. PhoneGap is an example of the most popular container for creating hybrid mobile apps.

For the most part, hybrid apps provide the best of both worlds. Existing web developers that have become gurus at optimizing JavaScript, pushing CSS to create beautiful layouts, and writing compliant HTML code that works on any platform can now create sophisticated mobile applications that don't sacrifice the cool native capabilities. In certain circumstances, native developers can write plugins for tasks like image processing, but in cases like this, the devil is in the details.

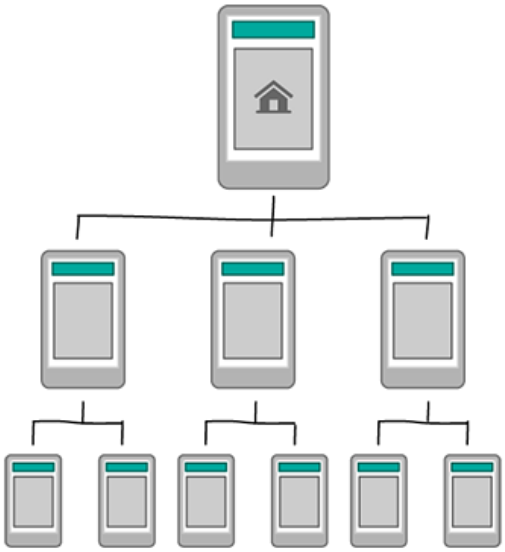
Native apps are installed on the device, while HTML5 apps reside on a Web server.

	Native	**HTML5**	**Hybrid**
App Features			
Graphics	Native APIs	HTML, Canvas, SVG	HTML, Canvas, SVG
Performance	Fast	Slow	Slow
Native look and feel	Native	Emulated	Emulated
Distribution	Appstore	Web	Appstore
Device Access			
Camera	Yes	No	Yes
Notifications	Yes	No	Yes
Contacts, calendar	Yes	No	Yes
Offline storage	Secure file storage	Shared SQL	Secure file system, shared SQL
Geolocation	Yes	Yes	Yes
Gestures			
Swipe	Yes	Yes	Yes
Pinch, spread	Yes	No	Yes
Connectivity	Online and offline	Mostly online	Online and offline
Development skills	ObjectiveC, Java	HTML5, CSS, Javascript	HTML5, CSS, Javascript

Mobile Information Architecture

Mobile devices have their own set of Information Architecture patterns, too. While the structure of a responsive site may follow more "standard" patterns, native apps, for example, often employ navigational structures that are tab-based. Again, there's no "right "way to architect a mobile site or application. Instead, let's take a look at some of the most popular patterns: Hierarchy, Hub & spoke, Nested doll, Tabbed view, Bento box and Filtered view:

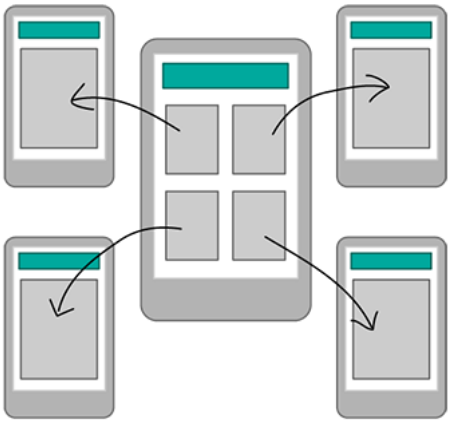
Hierarchy



The hierarchy pattern is a standard site structure with an index page and a series of sub pages. If you are designing a responsive site you may be restricted to this, however introducing additional patterns could allow you to tailor the experience for mobile.

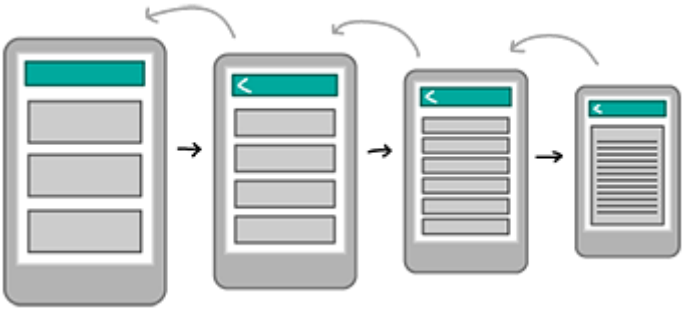
Luke Wroblewski's Mobile First approach helps us focus on the important stuff first: features and user journeys that will help us create great user experiences.

Hub & spoke



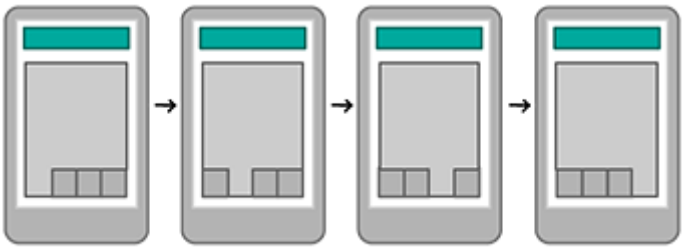
A hub and spoke pattern gives you a central index from which users will navigate out. It's the default pattern on Apple's iPhone. Users can't navigate between spokes but must return to the hub, instead. This has historically been used on desktop where a workflow is restricted (generally due to technical restrictions such as a form or purchasing process) however this is becoming more prevalent within the mobile landscape due to users being focused on one task, as well as the form factor of the device, making a global navigation more difficult to use.

Nested doll



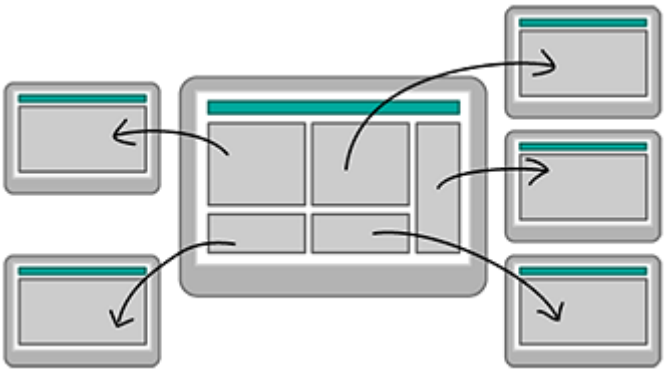
The nested doll pattern leads users in a linear fashion to more detailed content. When users are in difficult conditions this is a quick and easy method of navigation. It also gives the user a strong sense of where they are in the structure of the content due to the perception of moving forward and then back.

Tabbed view



This is a pattern that regular app users will be familiar with. It's a collection of sections tied together by a toolbar menu. This allows the user to quickly scan and understand the complete functionality of the app when it's first opened.

Bento Box/Dashboard

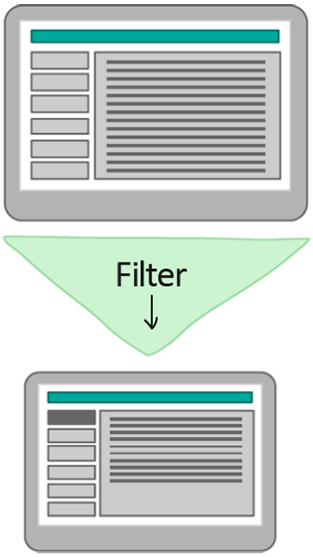


The bento box or dashboard pattern brings more detailed content directly to the index screen by using components to display portions of related tools or content. This pattern is more suited to tablet than mobile due to its complexity. It can be really powerful as it allows the user to comprehend key information at a glance, but does heavily rely on having a well-designed interface with information presented clearly.

Good for

Multi-functional tools and content-based tablet apps that have a similar theme.

Filtered view



Finally, a filtered view pattern allows the user to navigate within a set of data by selecting filter options to create an alternative view. Filtering, as well as using faceted search methods, can be an excellent way to allow users to explore content in a way that suits them.

Good for

Apps or sites with large quantities of content, such as articles, images and videos.

Uses of mobile phones

Digital camera: Point-and-click! Phones capture pictures and let us save them for posterity or transfer them to others and to computers.

Audio recorder: Mobile phones can be used to record conversations or even brief notes to oneself.

Video recorder: Phones are becoming video cameras also -- some of the newest cellphones can record an hour or more of video.

Multimedia messaging: Everything recorded can be shared with others by using MMS.

Email client: The phone can be used to connect to any POP or IMAP server and allow receiving and sending email. While most phones may not have the ease of use that a Blackberry has with email, contacts and calendar, the fact that it is on the phone itself and that there is no need for a separate device can be a big help (along with the lower total cost of ownership).

Web client: Phones can also browse websites, via a WAP and/or HTML browser. Most web sites may not look great on the small screen, but it is still possible to connect to any web site.

Gaming platform: Mobile games have become big business in the past couple years as people seek entertainment in the free time that they have on the device that they always carry with them.

Documents viewer: It is increasingly possible to view documents on the cell phone, in the popular MS-Office file formats.

Music player: The next big thing in 2005 is reckoned to be the combining of music capabilities on the mobile phone. While phones can play MP3s, it will soon also be possible to have music streamed from the Internet. Motorola is expected to introduce a phone this year that marries the mobile with Apple's iPod.

TV: In India, some operators have been promoting many TV channels on the cell phone over next-generation networks like EDGE.

Wallet: The phone can also be used to pay for purchases like a credit or debit card. There is already a billing relationship that exists between the subscriber and the operator, and that can be used to make payments to merchants.

Bar-code readers: Phones will also be able to read bar codes and that can have very interesting applications in commerce.

Introduction to Android

Android is a software package and linux based operating system for mobile devices such as tablet computers and smart phones released first version in 2007. It is developed by Google and later the OHA (Open Handset Alliance). Android is a complete set of software for mobile devices such as tablet computers, notebooks, smart phones, electronic book readers, set-top boxes etc. It contains a **linux-based Operating System**, **middleware** and **key mobile applications**. It can be thought of as a mobile operating system. But it is not limited to mobile only. It is currently used in various devices such as mobiles, tablets, televisions etc.

Java language is mainly used to write the android code even though other languages can be used. The goal of android project is to create a successful real-world product that improves the mobile experience for end users. There are many code names of android such as Lollipop, Kitkat, Jelly Bean, Ice cream Sandwich, Froyo, Ecliar, Donut etc which is covered in next page.

Features of Android

After learning what is android, let's see the features of android. The important features of android are given below:

1) It is open-source.

2) Anyone can customize the Android Platform.

3) There are a lot of mobile applications that can be chosen by the consumer.

4) It provides many interesting features like weather details, opening screen, live RSS (Really Simple Syndication) feeds etc.

It provides support for messaging services(SMS and MMS), web browser, storage (SQLite), connectivity (GSM, CDMA, Blue Tooth, Wi-Fi etc.), media, handset layout etc.

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

Feature	Description
Beautiful UI	Android OS basic screen provides a beautiful and intuitive user interface.
Connectivity	GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.
Storage	SQLite, a lightweight relational database, is used for data storage purposes.
Media support	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

Messaging	SMS and MMS
Web browser	Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.
Multi-touch	Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.
Multi-tasking	User can jump from one task to another and same time various application can run simultaneously.
Resizable widgets	Widgets are resizable, so users can expand them to show more content or shrink them to save space.
Multi-Language	Supports single direction and bi-directional text.
GCM	Google Cloud Messaging (GCM) is a service that lets developers send short message data to their users on Android devices.
Wi-Fi Direct	A technology that lets apps discover and pair directly, over a high-bandwidth peer-to-peer connection.
Android Beam	A popular NFC-based technology that lets users instantly share, just by touching two NFC-enabled phones together.

Android Applications

Android applications are usually developed in the Java language using the Android Software Development Kit.

Once developed, Android applications can be packaged easily and sold out either through a store such as **Google Play**, **SlideME**, **Opera Mobile Store**, **Mobango**, **F-droid** and the **Amazon Appstore**.

Android powers hundreds of millions of mobile devices in more than 190 countries around the world. It's the largest installed base of any mobile platform and growing fast. Every day more than 1 million new Android devices are activated worldwide.

Categories of Android applications

There are many android applications in the market. The top categories are:

- Entertainment
- Tools
- Communication
- Productivity
- Personalization
- Music and Audio

- Social
- Media and Video
- Travel and Local etc.

Versions of Android

The code names of android ranges from A to L currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwitch, Jelly Bean, KitKat and Lollipop. Let's understand the android history in a sequence.

What is API (Application Programming Interface) level?

API Level is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform.

Platform Version	API	VERSION_CODE
Android 5.1	22	LOLLIPOP_MR1
Android 5.0	21	LOLLIPOP

Android 4.4W	20	KITKAT_WATCH
Android 4.4	19	KITKAT
Android 4.3	18	JELLY_BEAN_MR2
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1
Android 4.1, 4.1.1	16	JELLY_BEAN
Android 4.0.3,	15	ICE_CREAM_SANDWICH_MR1
Android 4.0, 4.0.1,	14	ICE_CREAM_SANDWICH
Android 3.2	13	HONEYCOMB_MR2
Android 3.1.x	12	HONEYCOMB_MR1
Android 3.0.x	11	HONEYCOMB
Android 2.3.4	10	GINGERBREAD_MR1
Android 2.3.2	9	GINGERBREAD
Android 2.2.x	8	FROYO
Android 2.1.x	7	ECLAIR_MR1
Android 2.0.1	6	ECLAIR_0_1
Android 2.0	5	ECLAIR
Android 1.6	4	DONUT
Android 1.5	3	CUPCAKE
Android 1.1	2	BASE_1_1
Android 1.0	1	BASE

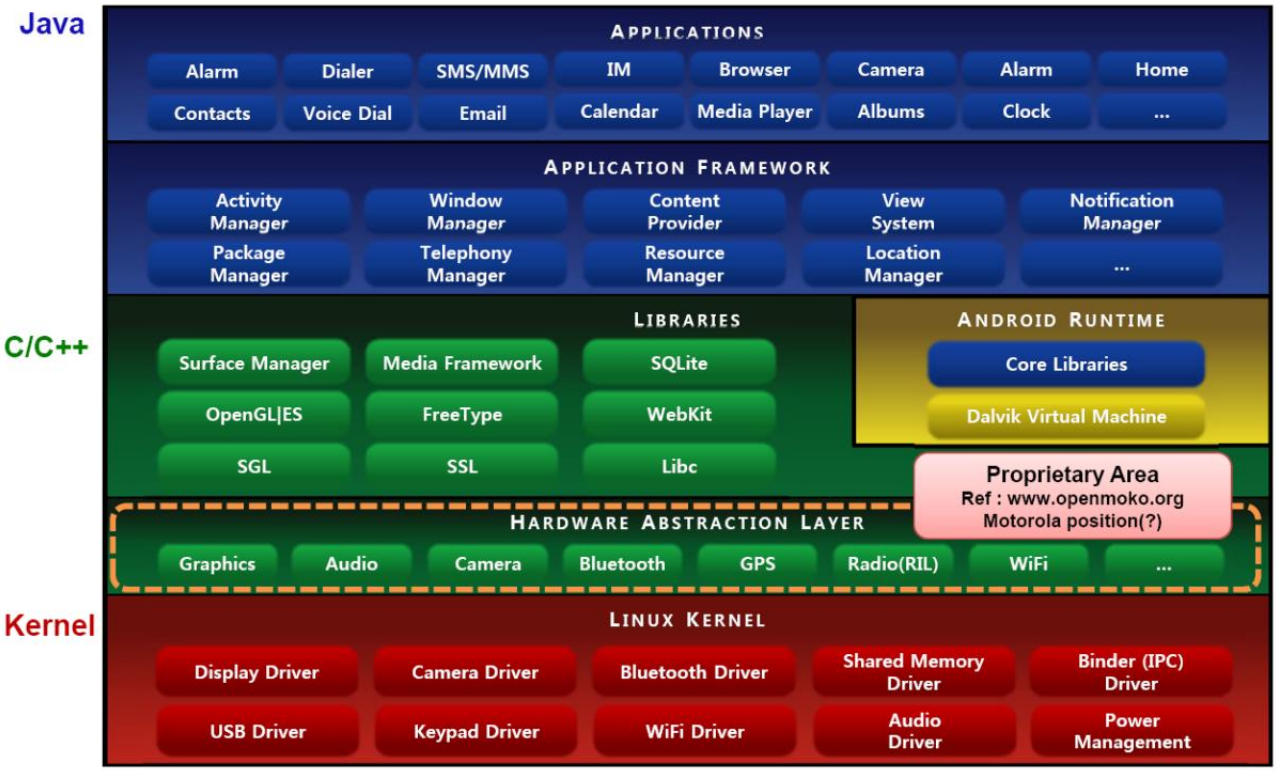
Android IDEs

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

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 (DVM)** 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.

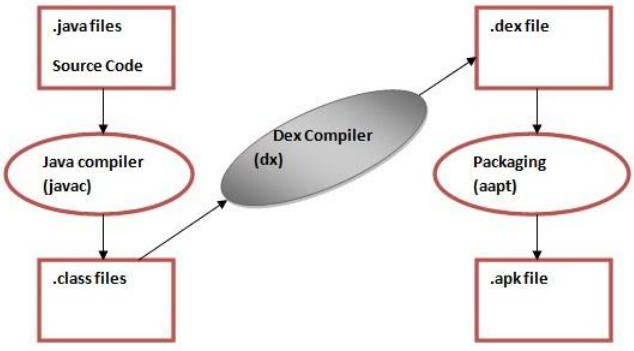
Applications

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.

Dalvik Virtual Machine | DVM

As we know the modern JVM is high performance and provides excellent memory management. But it needs to be optimized for low-powered handheld devices as well. The **Dalvik Virtual Machine (DVM)** is an android virtual machine optimized for mobile devices. It optimizes the virtual machine for *memory*, *battery life* and *performance*. Dalvik is a name of a town in Iceland. The Dalvik VM was written by Dan Bornstein. The Dex compiler converts the class files into the .dex file that run on the Dalvik VM. Multiple class files are converted into one dex file.

Let's see the compiling and packaging process from the source file:



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

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

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

Installing and using Eclipse with ADT plug – in

You will be glad to know that you 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 SDK
- Java Runtime Environment (JRE) 6
- Android Studio
- Eclipse IDE for Java Developers (optional)
- Android Development Tools (ADT) Eclipse Plug-in (optional)

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:\jdk1.6.0_15, you would have to put the following line in your C:\autoexec.bat file.

```
set PATH=C:\jdk1.7.0_75\bin;%PATH%
set JAVA_HOME=C:\jdk1.7.0_75
```

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/jdk1.6.0_15 and you use the C shell, you would put the following code into your **.cshrc** file.

```
setenv PATH /usr/local/jdk1.7.0_75/bin:$PATH
```

```
setenv JAVA_HOME /usr/local/jdk1.7.0_75
```

Alternatively, if you use an Integrated Development Environment (IDE) Eclipse, 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

Android supports java, c++, c# etc. language to develop android applications. Java is the officially supported language for android. All the android examples of this site is developed using Java language and Eclipse IDE.

Here, we are going to tell you, the required software to develop android applications using Eclipse IDE.

There are two ways to install android.

1. By ADT Bundle
2. By Setup Eclipse Manually

---

1) By ADT Bundle

It is the simplest technique to install required software for android application. It includes:

- Eclipse IDE
- Android SDK
- Eclipse Plugin

If you download the ADT from android site, you don't need to have eclipse IDE, android SDK and eclipse Plugin because it is already included in adt bundle.

If you have downloaded the ADT bundle, unjar it, go to eclipse IDE and start the eclipse by clicking on the eclipse icon. You don't need to do any extra steps here.

Note: If eclipse is not started, paste the JRE directory inside the eclipse directory.

2) By set up eclipse manually

How to setup Android for Eclipse IDE

In this page, you will learn what software are required for running an android application on eclipse IDE. Here, you will be able to learn how to install the android SDK and ADT plugin for Eclipse IDE. Let's see the list of software required to **setup android for eclipse** IDE manually.

1. Install the JDK

2. Download and install the Eclipse for developing android application
3. Download and Install the android SDK
4. Install the ADT plugin for eclipse
5. Configure the ADT plugin
6. Create the AVD
7. Create the hello android application

1) Install the Java Development Kit (JDK)

For creating android application, JDK must be installed if you are developing the android application with Java language.

2) Download and install the Eclipse IDE

For developing the android application using eclipse IDE, you need to install the Eclipse.

3) Download and install the android SDK

First of all, download the android SDK. In this example we have installed the android SDK for windows (.exe version).

Now double click on the exe file, it will be installed. I am using the android 2.2 version here.

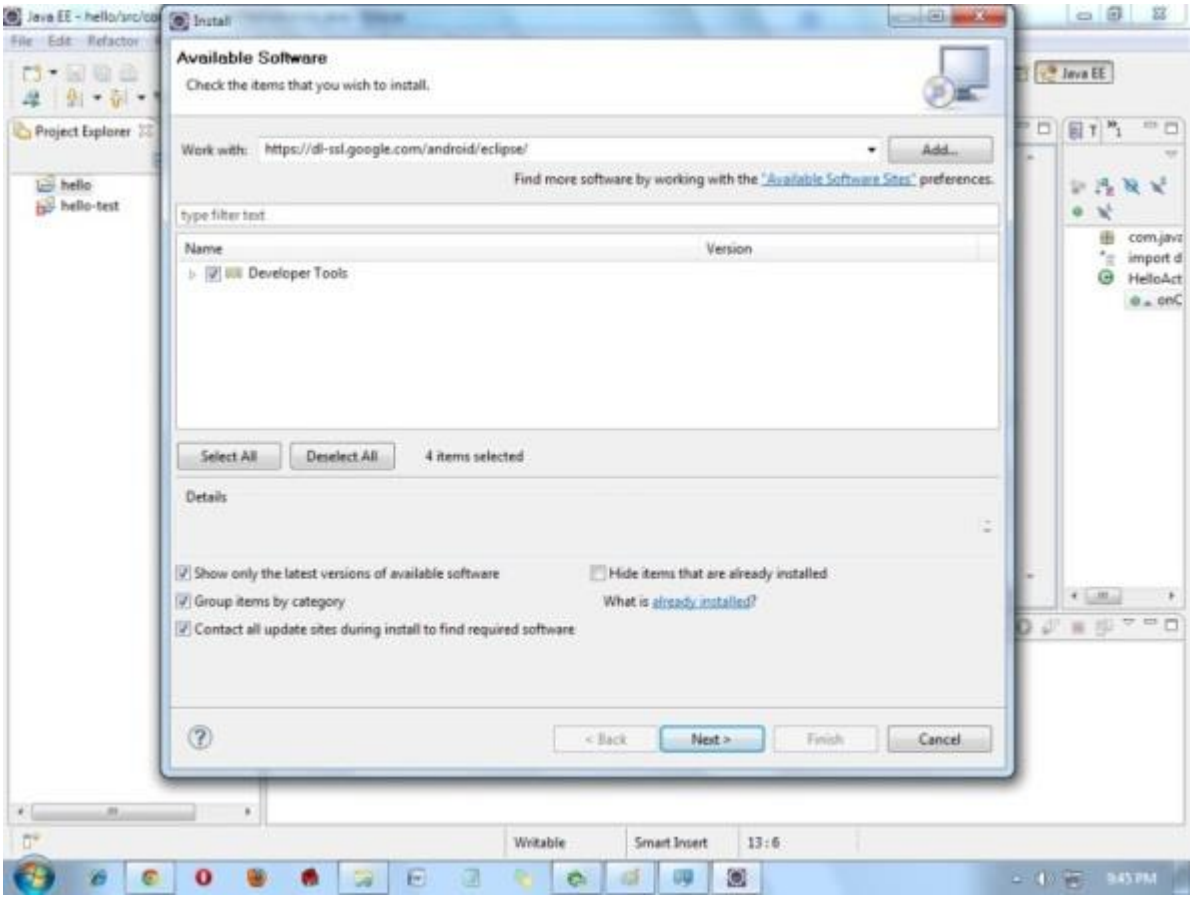
4) Download the ADT plugin for eclipse

ADT (Android Development Tools) is required for developing the android application in the eclipse IDE. It is the plugin for Eclipse IDE that is designed to provide the integrated environment.

For downloading the ADT, you need to follow these steps:

1) Start the eclipse IDE, then select **Help** > **Install new software...**

2) In the **work with** combo box, write **https://dl-ssl.google.com/android/eclipse/**



3) **select the checkbox** next to Developer Tools and **click next**

4) You will see, a list of tools to be downloaded here, **click next**

5) **click finish**

6) After completing the installation, restart the eclipse IDE

---

5) Configuring the ADT plugin

After the installing ADT plugin, now tell the eclipse IDE for your android SDK location. To do so:

1. Select the **Window menu > preferences**
2. Now select the android from the left panel. Here you may see a dialog box asking if you want to send the statistics to the google. Click **proceed**.
3. Click on the browse button and locate your SDK directory e.g. my SDK location is C:\Program Files\Android\android-sdk .
4. Click the apply button then OK.

---

6) Create an Android Virtual Device (AVD)

For running the android application in the Android Emulator, you need to create and AVD. For creating the AVD:

1. Select the **Window menu > AVD Manager**
2. Click on the **new** button, to create the AVD
3. Now a dialog appears, write the AVD name e.g. myavd. Now choose the target android version e.g. android2.2.
4. click the **create AVD**

---

7) create and run the simple android example

Visit the next page to create first android application.

How to make android apps

In this page, you will know how to create the simple hello android application. We are creating the simple example of android using the Eclipse IDE. For creating the simple example:

1. Create the new android project
2. Write the message (optional)
3. Run the android application

UNIT - II **(10 Lectures)**

Creating first android application, Anatomy of android application, Deploying Android app on USB connected Android device, Android application components, Activity life cycle, Understanding activities, Exploring Intent objects, Intent Types, Linking activities using intents

Creating first android application

Visit the next page to create first android application.

How to make android apps

In this page, you will know how to create the simple hello android application. We are creating the simple example of android using the Eclipse IDE. For creating the simple example:

4. Create the new android project
5. Write the message (optional)
6. Run the android application

Hello Android Example

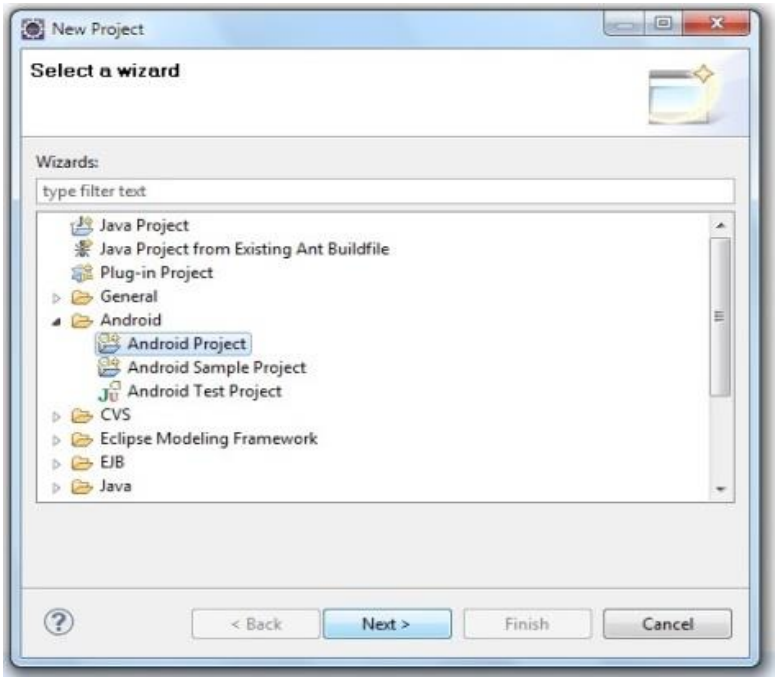
You need to follow the 3 steps mentioned above for creating the Hello android application.

1) Create the New Android project

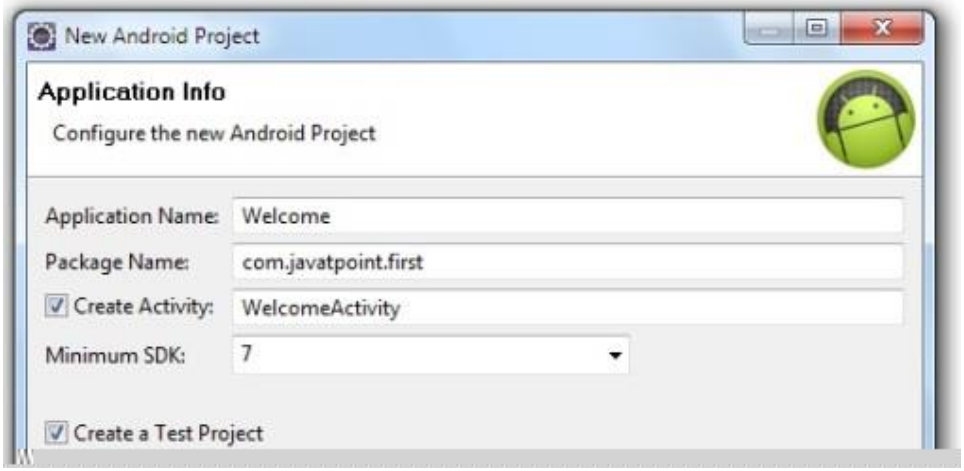
For creating the new android project:

1) Select **File > New > Project...**

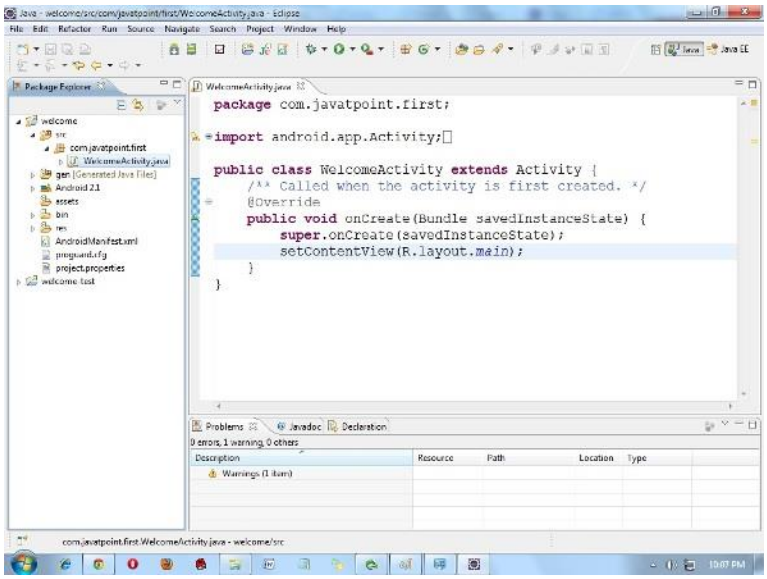
2) Select the android project and click **next**



3) Fill the Details in this dialog box and click **finish**



Now an android project have been created. You can explore the android project and see the simple program, it looks like this:



2) Write the message

For writing the message we are using the TextView class. Change the onCreate method as:

```
TextView textview=new TextView(this);
textview.setText("Hello Android!");
setContentView(textview);
```

Let's see the full code of MainActivity.java file.

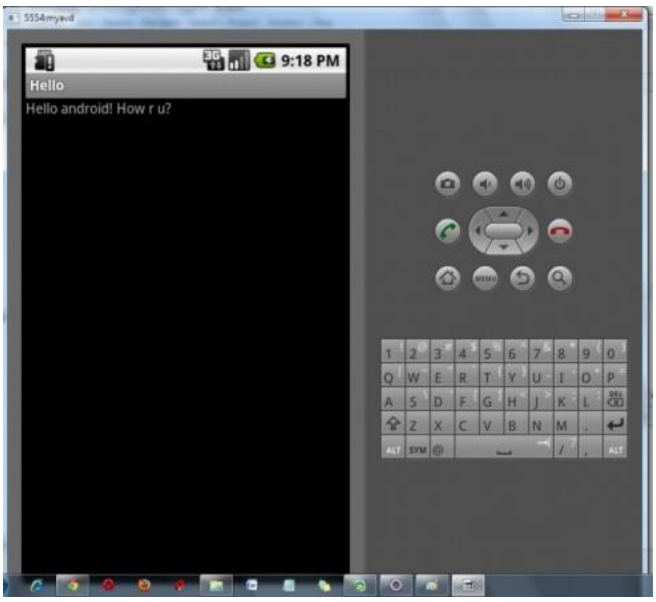
```
package com.example.helloandroid;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.TextView;
```

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView textview=new TextView(this);
        textview.setText("Hello Android!");
        setContentView(textview);
    }
    @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;
    }
}
```

3) Run the android application

To run the android application: **Right click on your project > Run As.. > Android Application**

The android emulator might take 2 or 3 minutes to boot. So please have patience. After booting the emulator, the eclipse plugin installs the application and launches the activity. You will see something like this:



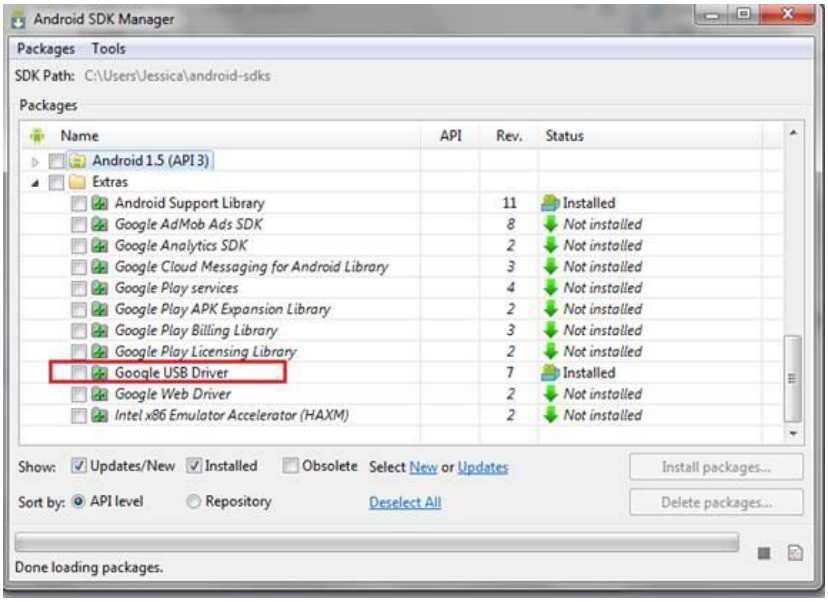
Deploy it on USB connected Android device

How to Connect Your Device

1. Before we start, you must enable debugging mode on your Android device. Open the 'Applications' folder and select 'Development.' On the subsequent screen, ensure the 'USB Debugging' checkbox is ticked.

2. If you're developing on Windows, you'll need to install the USB driver for your device. Your first stop should be the list of OEM USB Drivers maintained by Google.

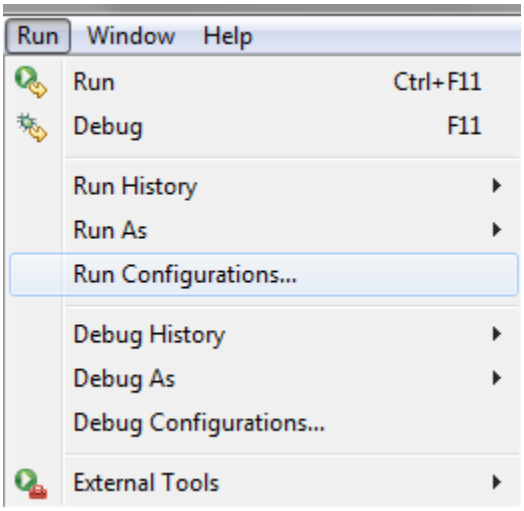
If you're working with a Google device (such as the Nexus line) you can download the necessary drivers through the Android SDK Manager. Open the 'Windows' menu and select 'Android SDK Manager' to get started. The Android SDK Manager will open, listing all of the installed packages, along with available updates and additional packages that can be installed. Locate the 'Google USB Driver' package under the 'Extras' folder. If it isn't installed, tick the checkbox and click the 'Install Package' button. This can take some time to download, so be prepared to wait a few minutes.



Android SDK Manager

3. Connect the device to your computer using the appropriate USB cable.

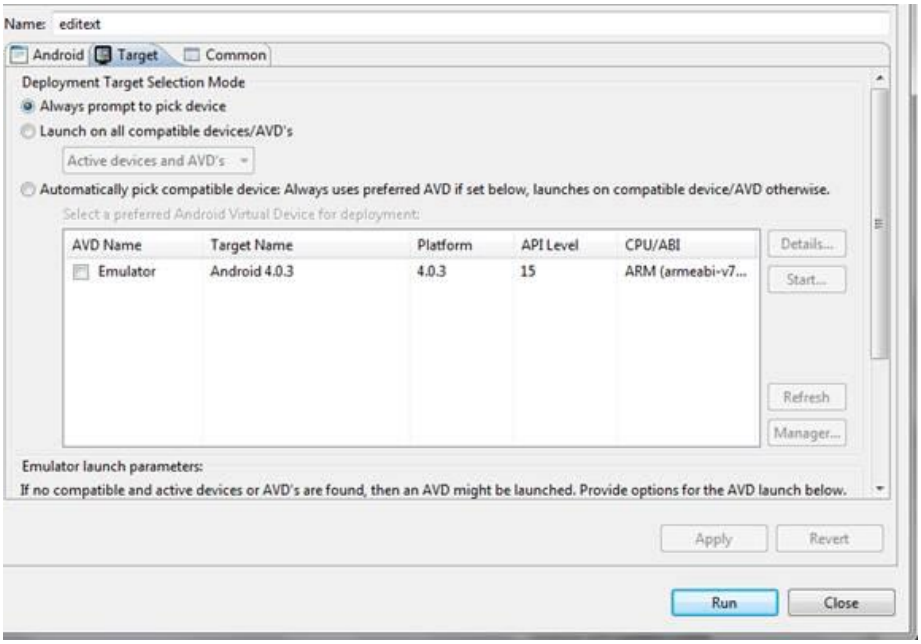
4. Inside your Eclipse installation, open the 'Run' menu and select 'Run Configurations.'



Run Configurations

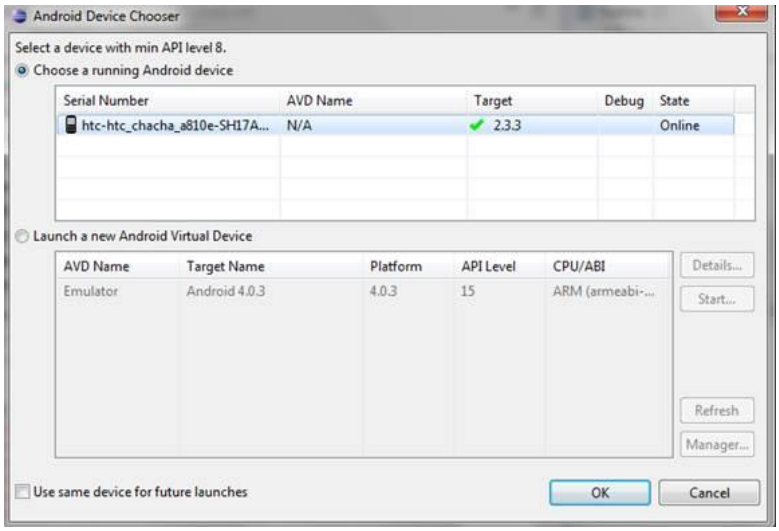
5. Select the Android project you wish to run/debug and open the 'Target' tab.

6. Ensure 'Always prompt to pick device' is selected, click 'Apply' and 'Run.'



Target tab >> Always prompt to pick device

7. The 'Android Device Chooser' will open. If you've connected your device successfully, it will be listed under 'Choose a running Android device.' Select your device and click 'OK.'



Android Device Chooser

8. Check your device - your app will have made the leap from Eclipse onto your screen!

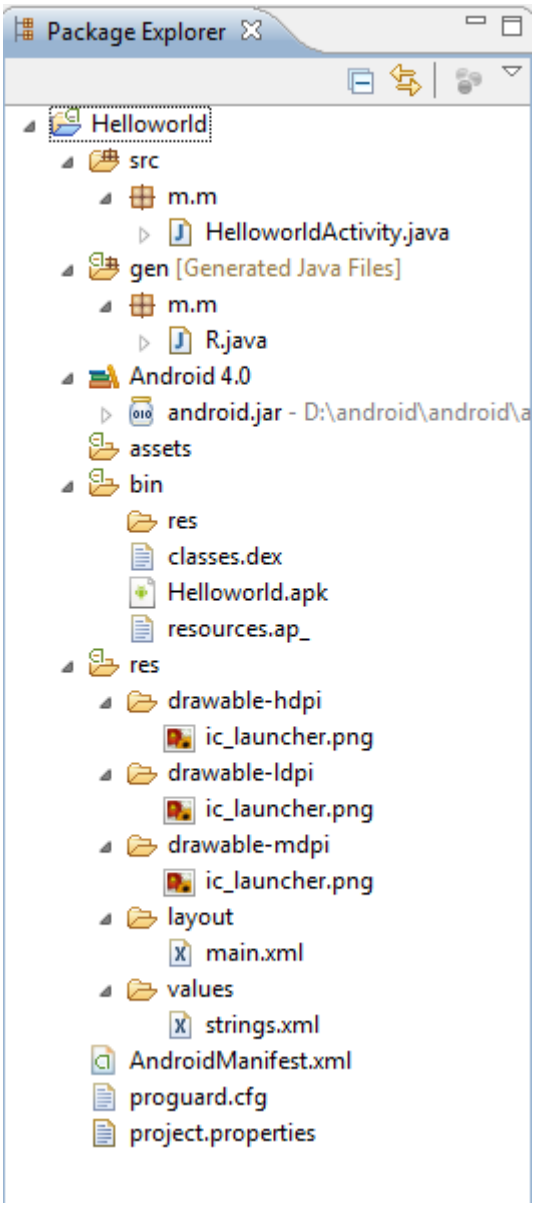
Steps:

1. **Connect** your **android phone** with the pc on which you are running **eclipse/your map project.**
2. Let it install all the necessary **drivers.** When done, open your smart phone, go to: **Settings > Applications > Development > USB debugging** and enable it on by clicking on the check button at the right side.
3. Also, enable **Settings > Unknown resources**

4. Come back to **eclipse** on your pc. Right click on the project/application, **Run As > Run configurations... >Choose Device>Target Select your device Run.**
5. Click on the **Target** tab from top. By default it is on the first tab **Android**
6. Choose the second radio button which says **Launch on all compatible devices/AVDs**. Then click **Apply** at the bottom and afterwards, click **Run**.
7. Here you go, it will automatically install your application's **.apk** file into your smart phone and make it run over it., just like on emulator.

Anatomy of android application

First, note the various files that make up an Android project in the Package Explorer in Eclipse.



The various folders and their files are as follows:

➤ src — Contains the .java source files for your project. In this example, there is one file, HelloworldActivity.java. The HelloworldActivity.java file is the source file for your activity. You will write the code for your application in this file.

➤ Android 2.3 library — This item contains one file, android.jar, which contains all the class libraries needed for an Android application.

➤ gen — Contains the R.java file, a compiler-generated file that references all the resources found in your project. You should not modify this file.

➤ assets — This folder contains all the assets used by your application, such as HTML, text files, databases, etc.

➤ res — This folder contains all the resources used in your application. It also contains a few other subfolders: drawable-<resolution>, layout, and values.

➤ AndroidManifest.xml — This is the manifest file for your Android application. Here you specify the permissions needed by your application, as well as other features
(such as intent-filters, receivers, etc.).

MAIN.XML

The main.xml file defines the user interface for your activity. Observe the following in red color:

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

The @string in this case refers to the strings.xml file located in the res/values folder.
Hence, @string/hello refers to the hello string defined in the strings.xml file, which is “Hello World, MainActivity!”:

```
<?xml version=”1.0” encoding=”utf-8”?>
<resources>
<string name=”hello”>Hello World, MainActivity!</string>
<string name=”app_name”>HelloWorld</string>
</resources>
```

It is recommended that you store all the string constants in your application in this strings.xml file and reference these strings using the @string identifier. That way, if you ever need to localize your application to another language, all you need to do is replace the strings stored in the strings.xml file with the targeted language and recompile your application.

Observe the content of the AndroidManifest.xml file:

```
<?xml version=”1.0” encoding=”utf-8”?>
<manifest xmlns:android=”http://schemas.android.com/apk/res/android”
package=”m.m”
android:versionCode=”1”
android:versionName=”1.0”>
<application android:icon=”@drawable/icon” android:label=”@string/app_name”>
<activity android:name=”.HelloworldActivity”
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>
<uses-sdk android:minSdkVersion=”9” />
</manifest>
```

The AndroidManifest.xml file contains detailed information about the application:

- ➤ It defines the package name of the application as m.m.
- ➤ The version code of the application is 1. This value is used to identify the version number of your application. It can be used to programmatically determine whether an application needs to be upgraded.
- ➤ The version name of the application is 1.0. This string value is mainly used for display to the user. You should use the format: <major>.<minor>.<point> for this value.
- ➤ The application uses the image named icon.png located in the drawable folder.
- ➤ The name of this application is the string named app_name defined in the strings.xml file.
- ➤ There is one activity in the application represented by the HelloworldActivity.java file. The label displayed for this activity is the same as the application name.

➤ Within the definition for this activity, there is an element named <intent-filter>:

➤ The action for the intent fi lter is named android.intent.action.MAIN to indicate that this activity serves as the entry point for the application.

➤ The category for the intent-fi lter is named android.intent.category.LAUNCHER to indicate that the application can be launched from the device's Launcher icon.

➤ Finally, the android:minSdkVersion attribute of the <uses-sdk> element specifies the minimum version of the OS on which the application will run.

As you add more fi les and folders to your project, Eclipse will automatically generate the content of R.java, which at the moment contains the following:

```
package m.m;
public final class R {
public static final class attr {
}
public static final class drawable {
public static final int icon=0x7f020000;
}
public static final class layout {
public static final int main=0x7f030000;
}
public static final class string {
public static final int app_name=0x7f040001;
public static final int hello=0x7f040000;
}
}
```

You are not supposed to modify the content of the R.java file; Eclipse automatically generates the content for you when you modify your project. Finally, the code that connects the activity to the UI (main.xml) is the setContentView() method, which is in the MainActivity.java file:

```
package m.m;
import android.app.Activity;
import android.os.Bundle;
```

```
public class HelloworldActivity extends Activity
{
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
}
}
```

Here, R.layout.main refers to the main.xml file located in the res/layout folder. As you add additional XML files to the res/layout folder, the filenames will automatically be generated in the R.java file. The onCreate() method is one of many methods that are fired when an activity is loaded.

AndroidManifest.xml file in android

The **AndroidManifest.xml file** *contains information of your package*, including components of the application such as activities, services, broadcast receivers, content providers etc.

It performs some other tasks also:

- It is **responsible to protect the application** to access any protected parts by providing the permissions.
- It also **declares the android api** that the application is going to use.
- It **lists the instrumentation classes**. The instrumentation classes provides profiling and other informations. These informations are removed just before the application is published etc.

This is the required xml file for all the android application and located inside the root directory.

A simple AndroidManifest.xml file looks like this:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.javatpoint.hello"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
```

```
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

Elements of the AndroidManifest.xml file

The elements used in the above xml file are described below.

<manifest>

manifest is the root element of the AndroidManifest.xml file. It has **package** attribute that describes the package name of the activity class.

<application>

application is the subelement of the manifest. It includes the namespace declaration. This element contains several subelements that declares the application component such as activity etc.

The commonly used attributes are of this element are **icon**, **label**, **theme** etc.

android:icon represents the icon for all the android application components.

android:label works as the default label for all the application components.

android:theme represents a common theme for all the android activities.

<activity>

activity is the subelement of application and represents an activity that must be defined in the AndroidManifest.xml file. It has many attributes such as label, name, theme, launchMode etc.

android:label represents a label i.e. displayed on the screen.

android:name represents a name for the activity class. It is required attribute.

<intent-filter>

intent-filter is the sub-element of activity that describes the type of intent to which activity, service or broadcast receiver can respond to.

<action>

It adds an action for the intent-filter. The intent-filter must have at least one action element.

<category>

It adds a category name to an intent-filter.

Android R.java file

Android R.java is *an auto-generated file by aapt* (Android Asset Packaging Tool) that contains resource IDs for all the resources of res/ directory.

If you create any component in the activity_main.xml file, id for the corresponding component is automatically created in this file. This id can be used in the activity source file to perform any action on the component.

Android Core Building Blocks

An android **component** is simply a piece of code that has a well defined life cycle e.g. Activity, Receiver, Service etc.

The **core building blocks** or **fundamental components** of android are activities, views, intents, services, content providers, fragments and AndroidManifest.xml.

Activity

An activity is a class that represents a single screen. It is like a Frame in AWT.

An activity is implemented as a subclass of **Activity** class as follows –

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

View

A view is the UI element such as button, label, text field etc. Anything that you see is a view.

Intent

Intent is used to invoke components. It is mainly used to:

- Start the service
- Launch an activity
- Display a web page
- Display a list of contacts
- Broadcast a message
- Dial a phone call etc.

Service

Service is a background process that can run for a long time.

There are two types of services local and remote. Local service is accessed from within the application whereas remote service is accessed remotely from other applications running on the same device.

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 {
}
```

Content Provider

Content Providers are used to share data between the applications.

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(){}
}
```

Fragment

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

AndroidManifest.xml

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

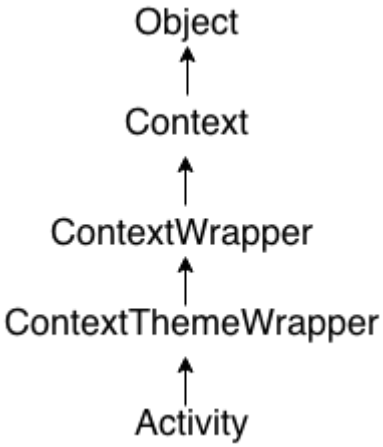
Broadcast Receivers

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){}
}
```

Activity life cycle



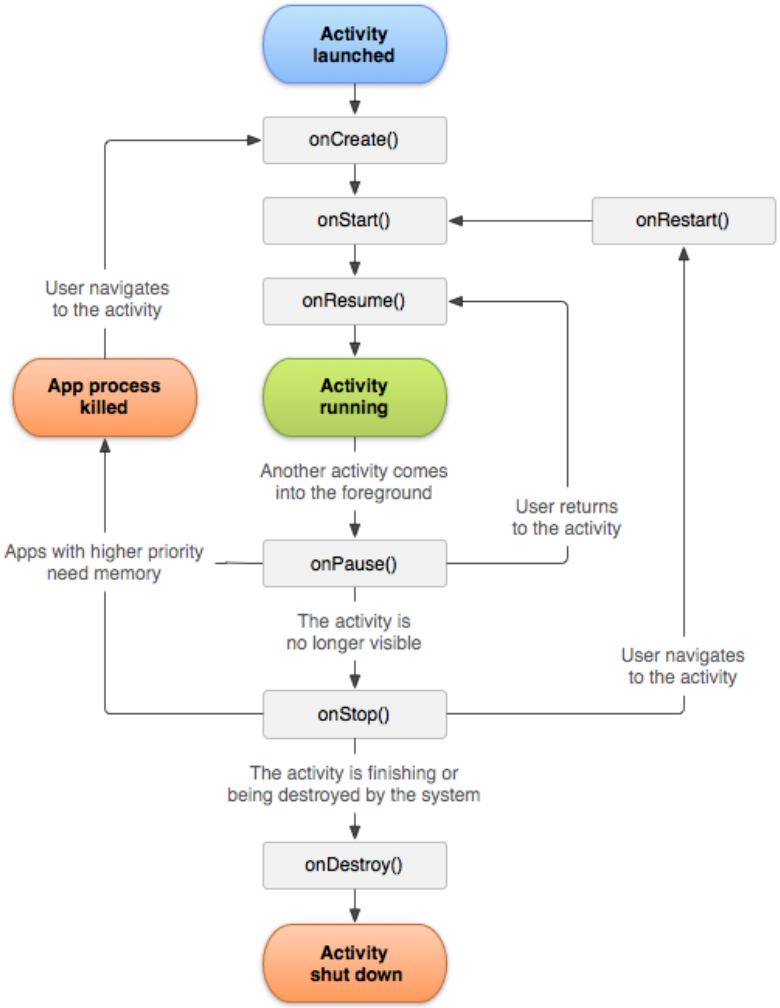
Android Activity Lifecycle is controlled by 7 methods of android.app.Activity class. The android Activity is the subclass of ContextThemeWrapper class.

An activity is the single screen in android. It is like window or frame of Java.
By the help of activity, you can place all your UI components or widgets in a single screen.
The 7 lifecycle method of Activity describes how activity will behave at different states.

Android Activity Lifecycle methods

Let's see the 7 lifecycle methods of android activity.

Method	Description
onCreate	called when activity is first created.
onStart	called when activity is becoming visible to the user.
onResume	called when activity will start interacting with the user.
onPause	called when activity is not visible to the user.
onStop	called when activity is no longer visible to the user.
onRestart	called after your activity is stopped, prior to start.
onDestroy	called before the activity is destroyed.



Android Activity Lifecycle Example

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

File: MainActivity.java

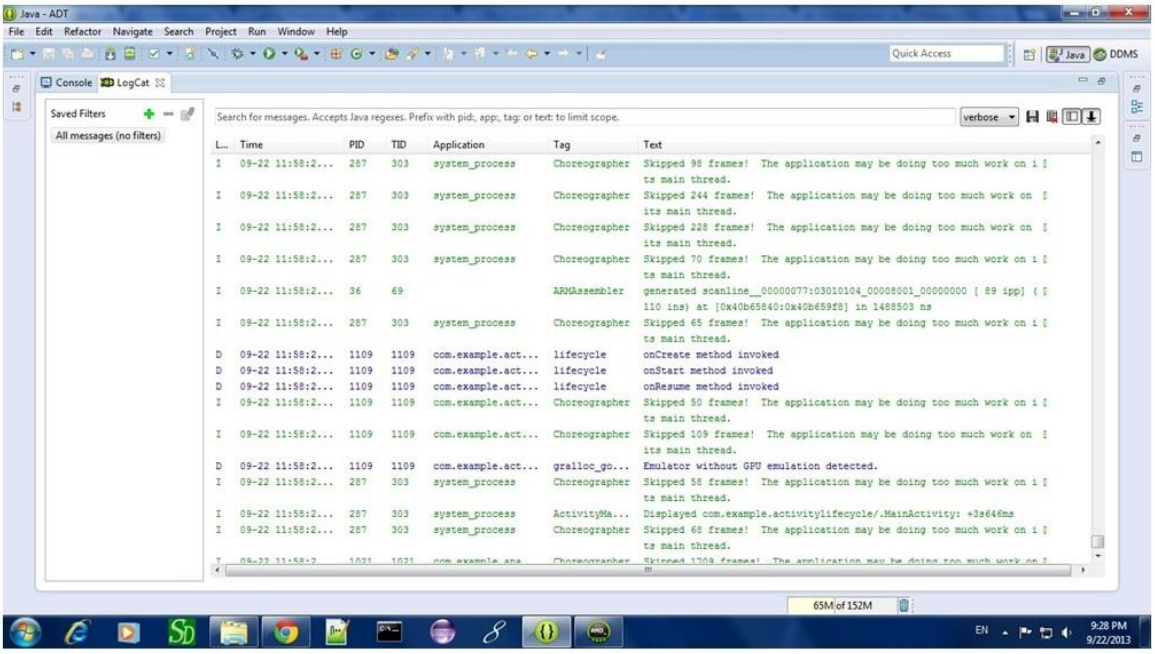
```
package com.example.activitylifecycle;
import android.os.Bundle;
import android.app.Activity;
import android.util.Log;
import android.view.Menu;
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("lifecycle","onCreate invoked");
    }
    @Override
    protected void onStart() {
        super.onStart();
         Log.d("lifecycle","onStart invoked");
    }
    @Override
    protected void onResume() {
        super.onResume();
         Log.d("lifecycle","onResume invoked");
    }
    @Override
    protected void onPause() {
        super.onPause();
```

```
        Log.d("lifecycle","onPause invoked");
    }
    @Override
    protected void onStop() {
        super.onStop();
        Log.d("lifecycle","onStop invoked");
    }
      @Override
    protected void onRestart() {
        super.onRestart();
        Log.d("lifecycle","onRestart invoked");
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d("lifecycle","onDestroy invoked");
    }
}
```

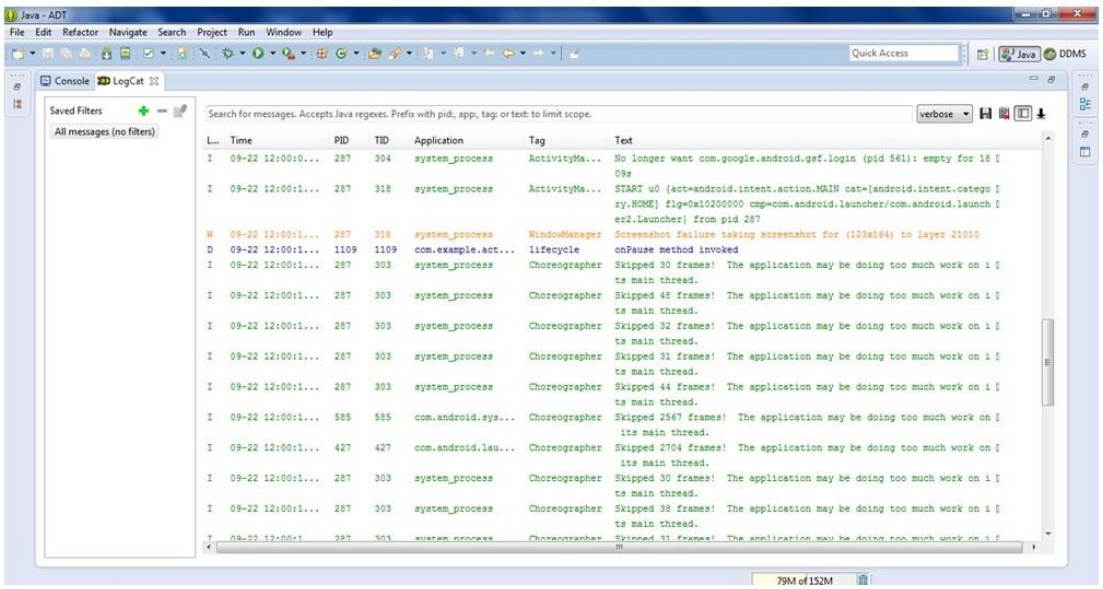
Output:

You will not see any output on the emulator or device. You need to open logcat.

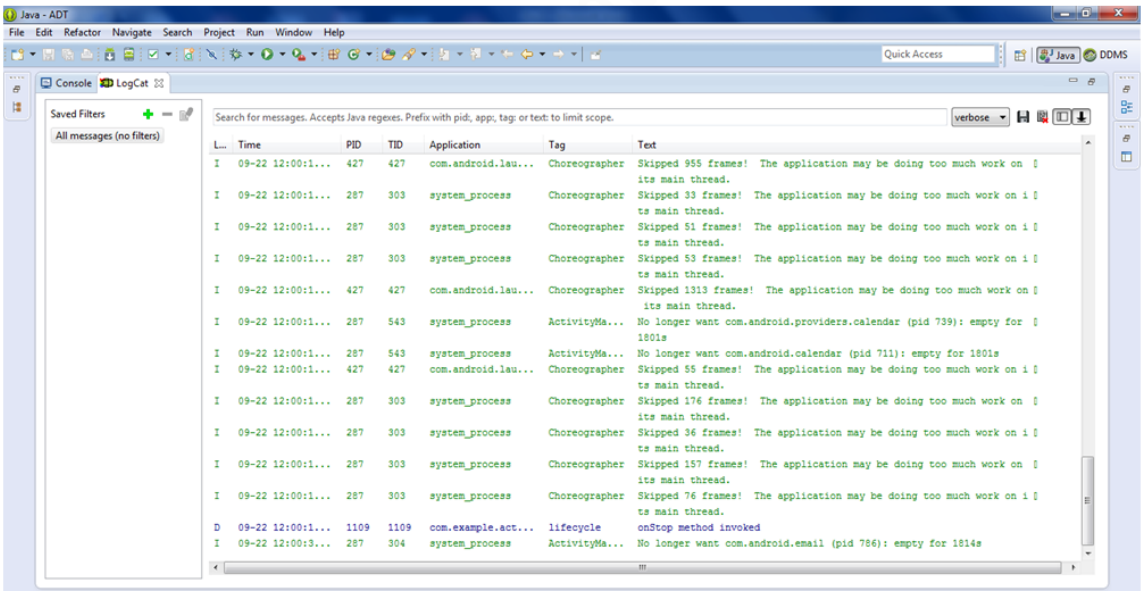
Now see on the logcat: onCreate, onStart and onResume methods are invoked.



Now click on the HOME Button. You will see onPause method is invoked.



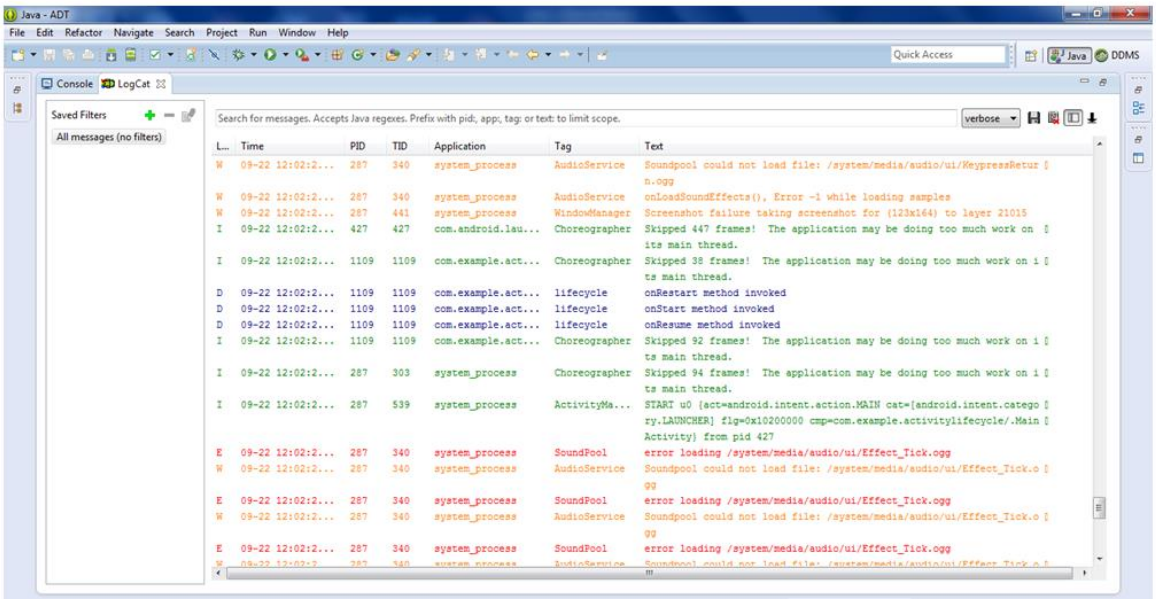
After a while, you will see onStop method is invoked.



Now see on the emulator. It is on the home. Now click on the center button to launch the app again.

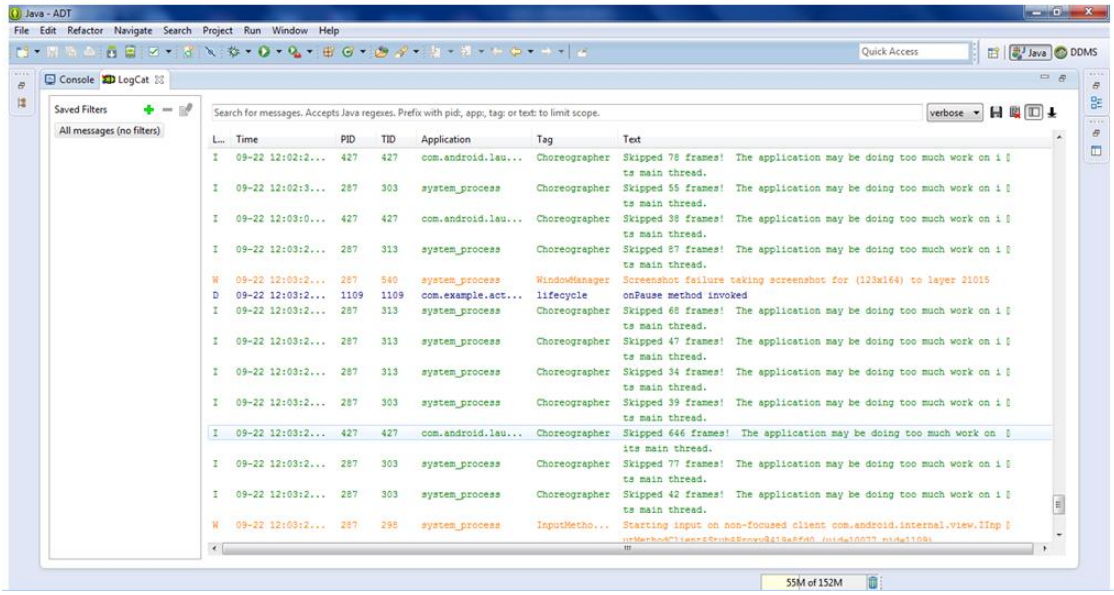
Now click on the lifecycleactivity icon.

Now see on the logcat: onRestart, onStart and onResume methods are invoked.

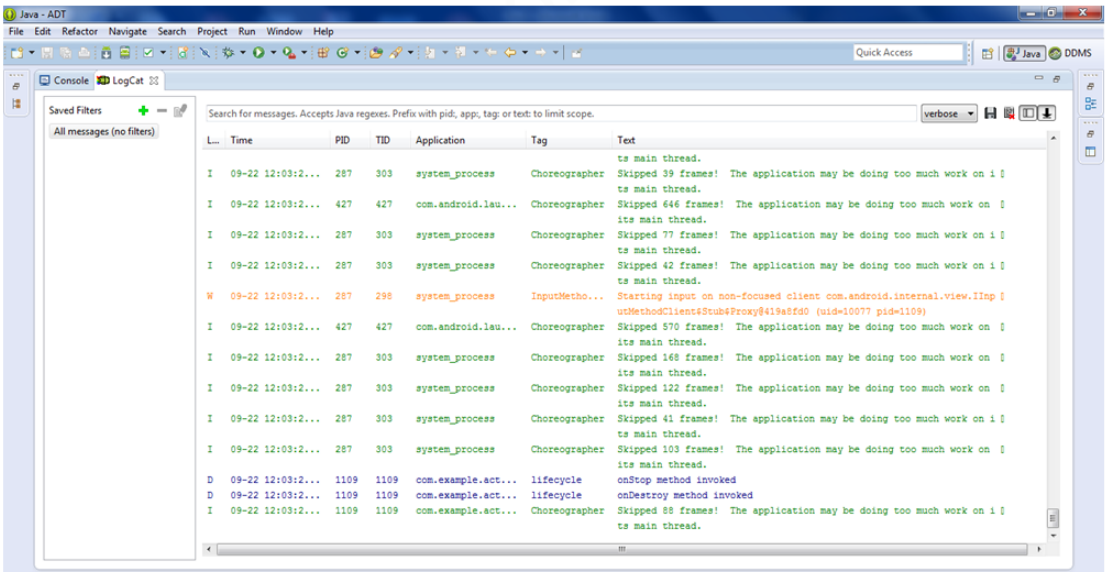


If you see the emulator, application is started again.

Now click on the back button. Now you will see onPause methods is invoked.

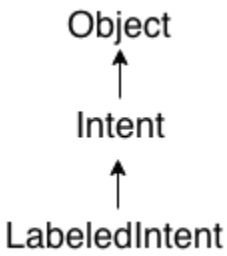


After a while, you will see onStop and onDestroy methods are invoked.



The onCreate() and onDestroy() methods are called only once throughout the activity lifecycle.

Exploring Intent objects



Android Intent is the *message* that is passed between components such as activities, content providers, broadcast receivers, services etc.

It is generally used with startActivity() method to invoke activity, broadcast receivers etc.

The **dictionary meaning** of intent is *intention or purpose*. So, it can be described as the intention to do action.

The LabeledIntent is the subclass of android.content.Intent class.

Android intents are mainly used to:

- Start the service
- Launch an activity
- Display a web page
- Display a list of contacts
- Broadcast a message

- Dial a phone call etc.

Intent Types

There are two types of intents in android: implicit and explicit.

1) Implicit Intent

Implicit Intent doesn't specifiy the component. In such case, intent provides information of available components provided by the system that is to be invoked.

For example, you may write the following code to view the webpage.

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

2) Explicit Intent

Explicit Intent specifies the component. In such case, intent provides the external class to be invoked.

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

To get the full code of explicit intent, visit the next page.

Android Implicit Intent Example

Let's see the simple example of implicit intent that displays a web page.

activity_main.xml

File: activity_main.xml

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >
    <EditText
      android:id="@+id/editText1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="44dp"
```

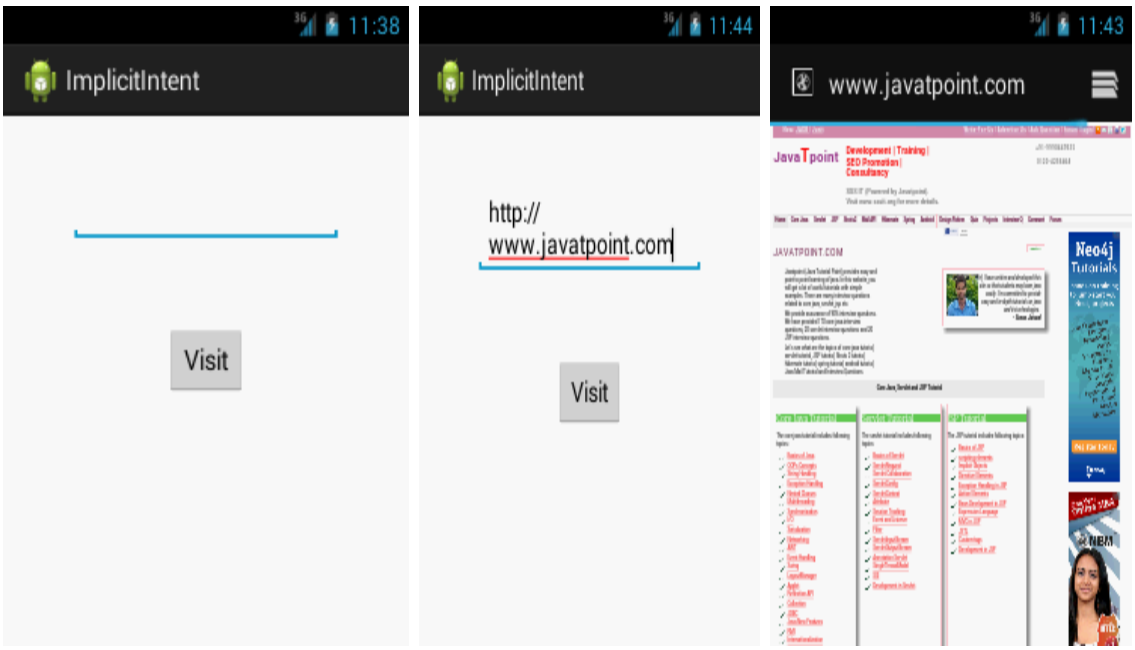
```
        android:ems="10" />
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editText1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="54dp"
        android:text="Visit" />
</RelativeLayout>
```

Activity class

File: MainActivity.java

```
package org.sssit.implicitintent;
import android.net.Uri;
import android.widget.EditText;
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final EditText editText1=(EditText)findViewById(R.id.editText1);
        Button button1=(Button)findViewById(R.id.button1);
        button1.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View arg0) {
                String url=editText1.getText().toString();
                Intent intent=new Intent(Intent.ACTION_VIEW,Uri.parse(url));
                startActivity(intent);
            } });  } }
```

Output:



Android Explicit Intent Example

Android Explicit intent specifies the component to be invoked from activity. In other words, we can call another activity in android by explicit intent. We can also pass the information from one activity to another using explicit intent. Here, we are going to see an example to call one activity from another and vice-versa.

Android calling one activity from another activity example

Let's see the simple example of android explicit example that calls one activity from another and vice versa.

activity_main.xml

File: activity_main.xml

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <Button
      android:id="@+id/Button01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
```

```
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/TextView01"
        android:layout_marginLeft="65dp"
        android:layout_marginTop="38dp"
        android:onClick="onClick"
        android:text="Call second activity" />

    <TextView
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/Button01"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="18dp"
        android:layout_marginTop="27dp"
        android:minHeight="60dip"
        android:text="First Activity"
        android:textSize="20sp" />

</RelativeLayout>
```

activitytwo_main.xml

File: activitytwo_main.xml

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <Button
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/TextView01"
```

```
        android:layout_marginLeft="65dp"
        android:layout_marginTop="38dp"
        android:onClick="onClick"
        android:text="Call First activity" />

    <TextView
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/Button01"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="18dp"
        android:layout_marginTop="27dp"
        android:minHeight="60dip"
        android:text="Second Activity"
        android:textSize="20sp" />
</RelativeLayout>
```

ActivityOne class

File: MainActivityOne.java

```
package com.example.explicitintent2;
import android.widget.Toast;
public class ActivityOne extends Activity {
  /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Button button1=(Button)findViewById(R.id.Button01);

    button1.setOnClickListener(new OnClickListener(){
     public void onClick(View view) {
      Intent i = new Intent(getApplicationContext(), ActivityTwo.class);
      i.putExtra("Value1", "Android By Javatpoint");
```

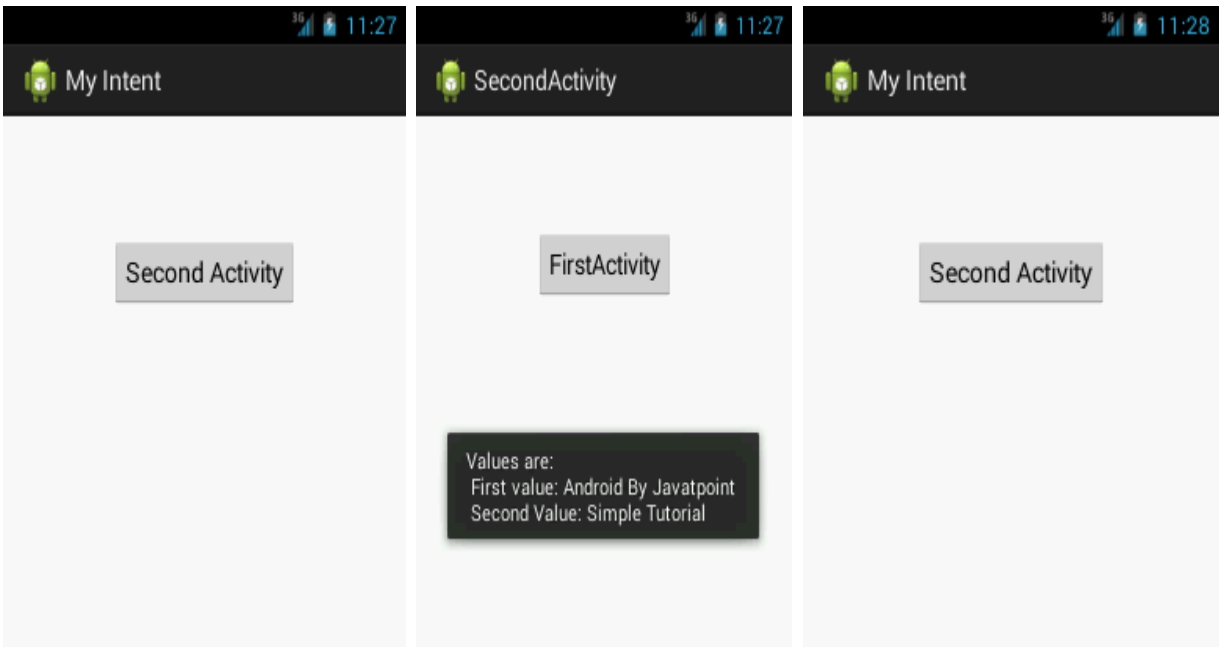
```
        i.putExtra("Value2", "Simple Tutorial");
        // Set the request code to any code you like, you can identify the
        // callback via this code
        startActivity(i);
       }
       });
     } }
```

ActivityTwo class

File: MainActivityTwo.java

```
package com.example.explicitintent2;
import android.app.Activity;
import android.widget.Toast;
public class ActivityTwo extends Activity {
/** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle bundle) {
  super.onCreate(bundle);
  TextView tv=new TextView(this);
  tv.setText("second activity");
  setContentView(R.layout.activity_two);
  Bundle extras = getIntent().getExtras();
  String value1 = extras.getString("Value1");
  String value2 = extras.getString("Value2");
  Toast.makeText(getApplicationContext(),"Values are:\n First value: "+value1+
     "\n Second Value: "+value2,Toast.LENGTH_LONG).show();
  Button button1=(Button)findViewById(R.id.Button01);
  button1.setOnClickListener(new OnClickListener(){
    public void onClick(View view) {
      Intent i = new Intent(getApplicationContext(), ActivityOne.class);
      startActivity(i);
     }
 });
} }
```

Output:


11:27
My Intent
Second Activity
11:27
SecondActivity
FirstActivity
Values are:
First value: Android By Javatpoint
Second Value: Simple Tutorial
11:28
My Intent
Second Activity

UNIT - III **(12 Lectures)**

Fragments life cycle, Interaction between fragments, Understanding the components of a screen (Layouts), Adapting to display orientation, Managing changes to screen orientation, Utilizing the Action Bar, Working with Views(UI Widgets)-Button, Toast, ToggleButton, CheckBox, RadioButton, Spinner, WebView, EditText, DatePicker, TimePicker, ListView, ProgressBar, Analog and Digital clock, Handling UI events, List fragment, Dialog fragment

Android - 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**. There can be more than one fragment in an activity. Fragments represent multiple screen inside one activity. Android fragment lifecycle is affected by activity lifecycle because fragments are included in activity. Each fragment has its own life cycle methods that is affected by activity life cycle because fragments are embedded in activity.

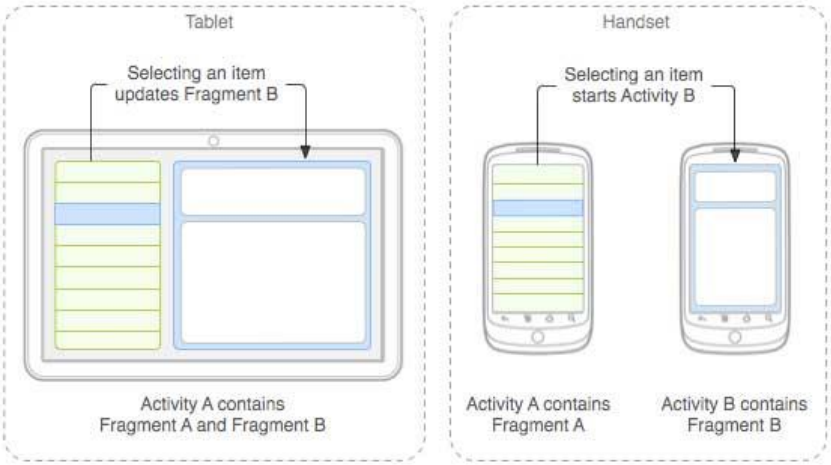
Following are important points about fragment –

- 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.

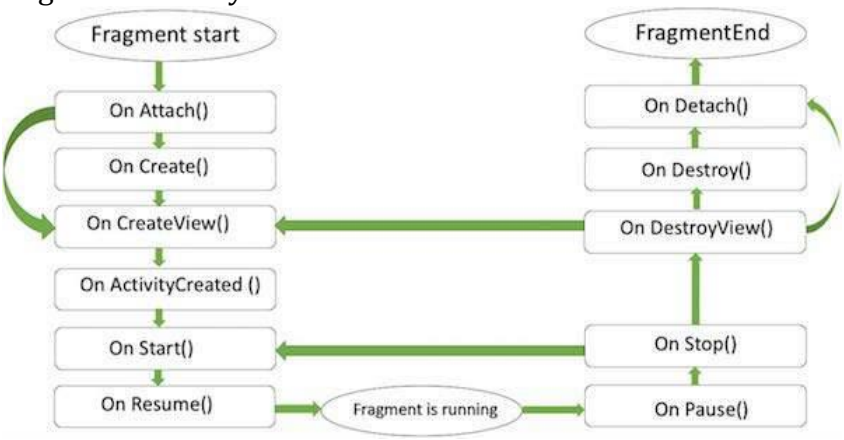
Following is a typical example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.



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.

Android Fragment Example

Let's have a look at the simple example of android fragment.

activity_main.xml

File: activity_main.xml

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

  <fragment
    android:id="@+id/fragment2"
    android:name="com.example.fragmentexample.Fragment2"
    android:layout_width="0px"
    android:layout_height="match_parent"
    android:layout_weight="1" />
   <fragment
    android:id="@+id/fragment1"
    android:name="com.example.fragmentexample.Fragment1"
    android:layout_width="0px"
```

```
        android:layout_height="match_parent"
        android:layout_weight="1"       />
    </LinearLayout>
```
File: fragment1.xml
```
    <?xml version="1.0" encoding="utf-8"?>
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
      android:background="#00ff00"      >
       <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="fragment frist"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    </LinearLayout>
```
File: fragment2.xml
```
    <?xml version="1.0" encoding="utf-8"?>
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
      android:background="#0000ff"
       >
       <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Second Fragment"
        android:textAppearance="?android:attr/textAppearanceLarge" />
    </LinearLayout>
```

MainActivity class

File: MainActivity.java
```
    package com.example.fragmentexample;
     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);
      }
    }
```

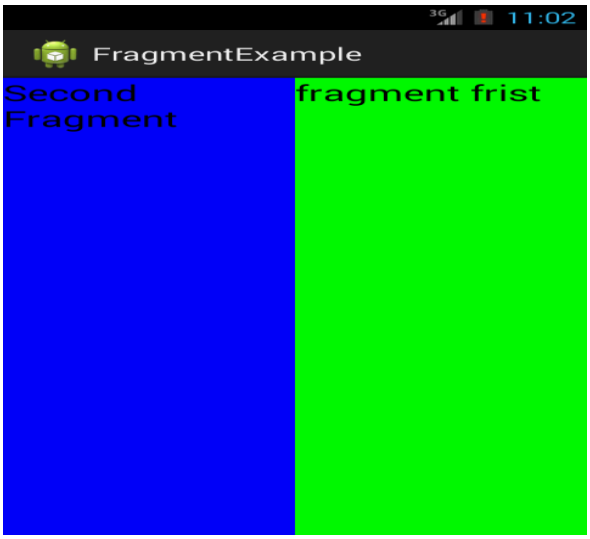
File: Fragment1.java
```
    package com.example.fragmentexample;
```

```
 import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class Fragment1 extends Fragment {
  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    // TODO Auto-generated method stub
    return inflater.inflate(R.layout.fragment1,container, false);
  }
 }
```

File: Fragment2.java

```
package com.example.fragmentexample;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class Fragment2 extends Fragment {
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    // TODO Auto-generated method stub
    return inflater.inflate(R.layout.fragment2,container, false);
  }
 }
```

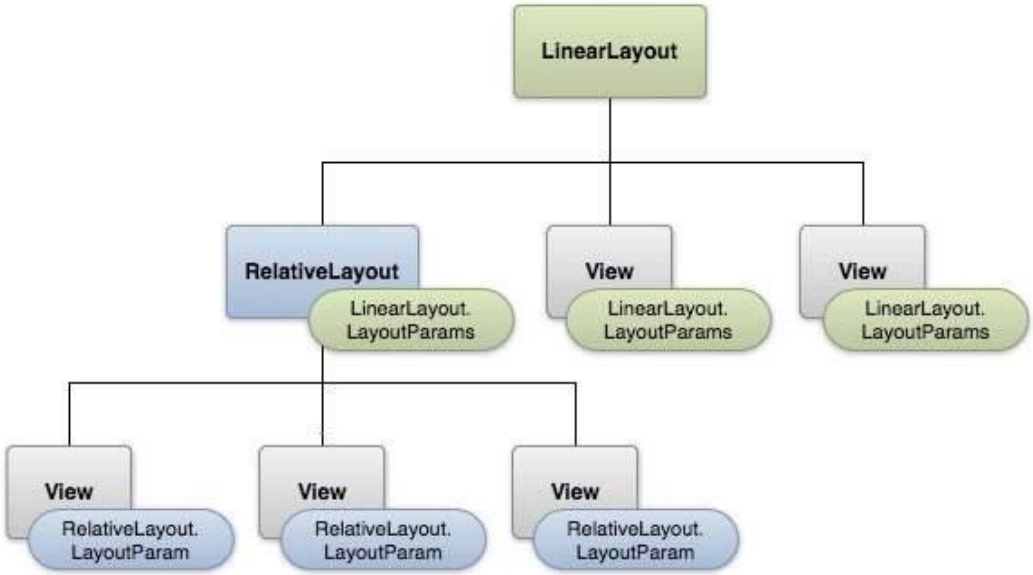
Output:



Android - UI Layouts

The basic building block for user interface is a **View** object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc.

The **ViewGroup** is a subclass of **View** and provides invisible container that hold other Views or other ViewGroups and define their layout properties.
At third level we have different layouts which are subclasses of ViewGroup class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using **View/ViewGroup** objects or you can declare your layout using simple XML file **main_layout.xml** which is located in the res/layout folder of your project.



LAYOUT PARAMS

This tutorial is more about creating your GUI based on layouts defined in XML file. A layout may contain any type of widgets such as buttons, labels, textboxes, and so on. Following is a simple example of XML file having LinearLayout:

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

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

  <Button android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="This is a Button" />
      <!-- More GUI components go here  -->
  </LinearLayout>
```

Once your layout has created, you can load the layout resource from your application code, in your *Activity.onCreate()* callback implementation as shown below –

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

Android Layout Types

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

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

Layout Attributes

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

Attribute	Description
android:id	This is the ID which uniquely identifies the view.
android:layout_width	This is the width of the layout.
android:layout_height	This is the height of the layout

android:layout_marginTop	This is the extra space on the top side of the layout.
android:layout_marginBottom	This is the extra space on the bottom side of the
android:layout_marginLeft	This is the extra space on the left side of the layout.
android:layout_marginRight	This is the extra space on the right side of the layout.
android:layout_gravity	This specifies how child Views are positioned.
android:layout_weight	This specifies how much of the extra space in the
android:layout_x	This specifies the x-coordinate of the layout.
android:layout_y	This specifies the y-coordinate of the layout.
android:layout_width	This is the width of the layout.
android:layout_width	This is the width of the layout.
android:paddingLeft	This is the left padding filled for the layout.
android:paddingRight	This is the right padding filled for the layout.
android:paddingTop	This is the top padding filled for the layout.
android:paddingBottom	This is the bottom padding filled for the layout.

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

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

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

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

Constant	Value	Description
top	0x30	Push object to the top of its container, not changing its size.
bottom	0x50	Push object to the bottom of its container, not changing its size.
left	0x03	Push object to the left of its container, not changing its size.
right	0x05	Push object to the right of its container, not

		changing its size.
center_vertical	0x10	Place object in the vertical center of its container, not changing its size.
fill_vertical	0x70	Grow the vertical size of the object if needed so it completely fills its container.
center_horizontal	0x01	Place object in the horizontal center of its container, not changing its size.
fill_horizontal	0x07	Grow the horizontal size of the object if needed so it completely fills its container.
center	0x11	Place the object in the center of its container in both the vertical and horizontal axis, not changing its size.
fill	0x77	Grow the horizontal and vertical size of the object if needed so it completely fills its container.
clip_vertical	0x80	Additional option that can be set to have the top and/or bottom edges of the child clipped to its container's bounds. The clip will be based on the vertical gravity: a top gravity will clip the bottom edge, a bottom gravity will clip the top edge, and neither will clip both edges.
clip_horizontal	0x08	Additional option that can be set to have the left and/or right edges of the child clipped to its container's bounds. The clip will be based on the horizontal gravity: a left gravity will clip the right edge, a right gravity will clip the left edge, and neither will clip both edges.
start	0x00800003	Push object to the beginning of its container, not changing its size.
end	0x00800005	Push object to the end of its container, not changing its size.

View Identification

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

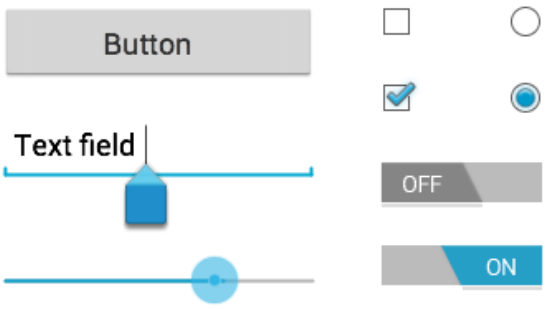
```
android:id="@+id/my_button"
```

Following is a brief description of @ and + signs −

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

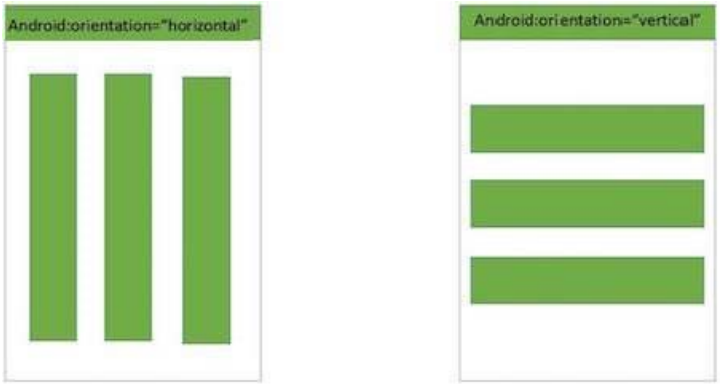
```
Button myButton = (Button) findViewById(R.id.my_button);
```

nput controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, check box, zoom buttons, toggle buttons, and many more.



Android Linear Layout

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"/>
```

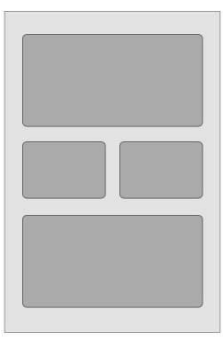
</LinearLayout>
Following will be the content of **res/values/strings.xml** to define two new constants −

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

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

Android Relative Layout

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.



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>
```

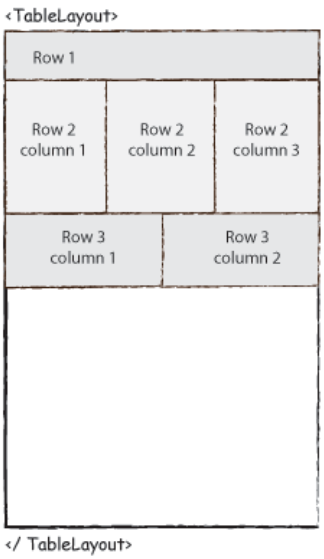
Following will be the content of **res/values/strings.xml** to define two new constants –

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

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"
```

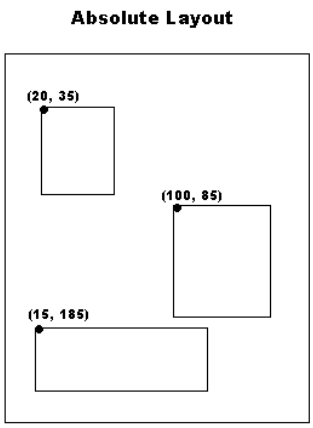
```
    android:layout_height="fill_parent">

    <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>
```

Android Absolute Layout

An Absolute Layout lets you specify exact locations (x/y coordinates) of its children. Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning.



Absolute Layout

AbsoluteLayout Attributes

Following are the important attributes specific to AbsoluteLayout –

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:layout_x** This specifies the x-coordinate of the view.
3	**android:layout_y** This specifies the y-coordinate of the view.

Public Constructors

AbsoluteLayout(Context context)

AbsoluteLayout(Context context, AttributeSet attrs)

AbsoluteLayout(Context context, AttributeSet attrs, int defStyleAttr)

AbsoluteLayout(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes)

Example

This example will take you through simple steps to show how to create your own Android application using absolute 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 absolute 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;

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

}
```

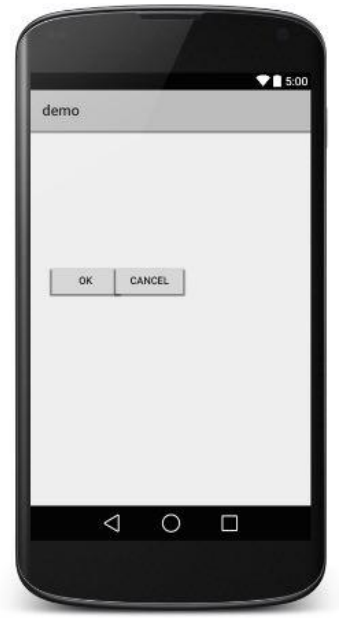
Following will be the content of **res/layout/activity_main.xml** file –

```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
    <Button
    android:layout_width="100dp"
    android:layout_height="wrap_content"
    android:text="OK"
```

```
    android:layout_x="50px"
    android:layout_y="361px" />
  <Button
    android:layout_width="100dp"
    android:layout_height="wrap_content"
    android:text="Cancel"
    android:layout_x="225px"
    android:layout_y="361px" />
</AbsoluteLayout>
```

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>
```



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.

You can, however, add multiple children to a FrameLayout and control their position within the FrameLayout by assigning gravity to each child, using the android:layout_gravity attribute.

Frame Layout

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;
```

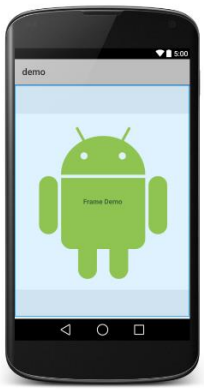
```
public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
}
```

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>
```

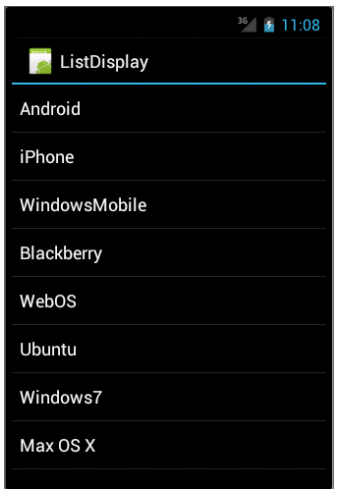
Following will be the content of **res/values/strings.xml** to define two new constants –

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



Android List View

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.



List View

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 –

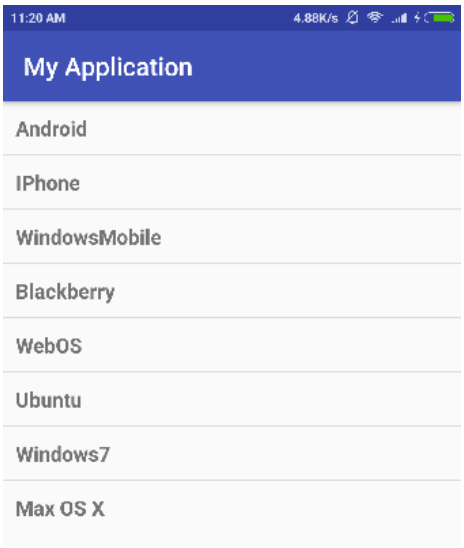
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  tools:context=".ListActivity" >

  <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>
```

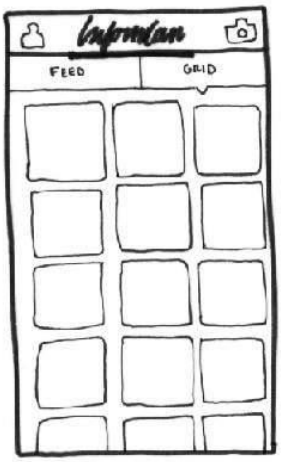
Following will be the content of **res/layout/activity_listview.xml** file –

```
<?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>
```



Android Grid View

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**



Grid view

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
  };
}
```

Multiple screen sizes

One important aspect to consider is how to support multiple screens in Android. As we know, Android is an operating system installed on many devices. One of the most important consequences is that the developer has to face with different screen resolution and densities. There are, on the market, many devices with different capabilities in terms of resolution and density and as developer we have to take it into account in order to create an application that can have a good usability in all these devices. You can think that an android application can be executed on a smart phone and on a tablet and it should be clear the screen size differences.

Before digging into the details how to support multiple screen sizes and densities it is important that we clarify some terms:

– **Resolution:** is the number of pixels on the device screen. It can be for example 480×800 etc.

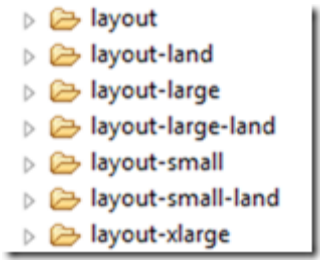
– Density: is the number of pixel in a specific area. Usually we consider dot per inch (dpi). This is a measure of the screen quality

– **Orientation:** is how the screen is oriented. It can assume two different values: **Landscape** and **Portrait**

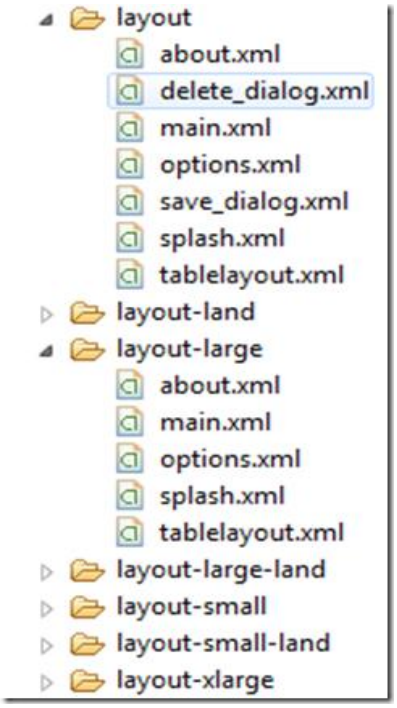
Using these concepts we can create apps that support multiple screens. There are two different aspects that we have to consider when we want to support multiple screens in Android: one is how we place object into the screen and we define it as layout and the second is the images. The first aspect has to do with resolution because we can have more or less space to place our components the second has to do with image resolution. Android has a smart way to handle different layout for different screen size and different image resolutions for different screen density.

From (inch)	To (inch)	Android name
2	4	***small***
around 4	around 4	***normal***
from around 4	7	***large***
7	10	***xlarge***

If we want to support all these screen sizes we have to map these android names into a directory structure inside the res directory. The convention used by android is the add these name at the end of *layout* word. For example *layout-small*, *layout-normal* etc. So inside ***res*** directory we will have a structure like this one:



As you can see all names are mapped in this structure except normal. Normal is the default layout so it isn't mapped with its name but just with ***layout***. Now let's see what is inside each directory.



As you can see each directory contains the same elements with all the same names. The only thing that changes is the content of each element.

For example, if we open one of these files (main.xml) in these two directories you will have:

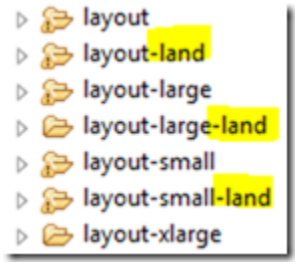
Layout

```
<ImageView android:id="@+id/title" android:src="@drawable/title"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_gravity="center_horizontal"
            android:layout_marginTop="10dp"/>
```

layout -large

```
<ImageView android:id="@+id/title" android:src="@drawable/title"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_gravity="center_horizontal"
           android:layout_marginTop="40dp"
           />
```

Doing so you take into the account only the portrait screen orientation. What happen if the user rotates the screen or the device as a landscape screen? No problem we can support this kind of screen including the magic word ***–land.*** So the full structure becomes:



In these way we can support multiple screen in Android!!. I guess you are wondering what inside each file in land layout. Well in the landscape layout we have to place components in a different way. Let's see how i solved this problem in a real application published on the market.

Landscape Orientation:

```
<LinearLayout android:id="@+id/linearLayout1"
    android:orientation="horizontal" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:layout_gravity="center_horizontal">
```

Portrait Orientation:

```
<LinearLayout android:id="@+id/linearLayout1"
              android:orientation="vertical"
              android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:layout_gravity="center_vertical" android:layout_marginTop="30dp">
```

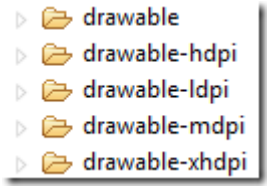
Android multiple screen: Screen Density

By now we considered only the screen size but as we said earlier there are different densities too. Density affects primary the images so an image could be the right size one a screen device and looks too small in another device with more density.

So we have to consider this problem. Android groups devices in categories based on their screen density, so we have:

Density	Android name
Small	around 120dpi (**ldpi**)
Normal	around 160dpi (**mdpi**)
High	around 240dpi (**hdpi**)
x-High	around 320dpi (**xhdpi**)

As we did for layouts we can built a directory structure under the directory res that maps these names. This time we don't use layout but instead ***drawable***. So we will have:



This time we see a directory drawable that doesn't have any name specification. It is the default directory that is used when we don't want to map every density.

Create UI Controls

Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, check box, zoom buttons, toggle buttons, and many more.

As explained in previous chapter, a view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is –

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

To create a UI Control/View/Widget you will have to define a view/widget in the layout file and assign it a unique ID as follows –

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

  <TextView android:id="@+id/text_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="I am a TextView" />
</LinearLayout>
```

Then finally create an instance of the Control object and capture it from the layout, use the following –

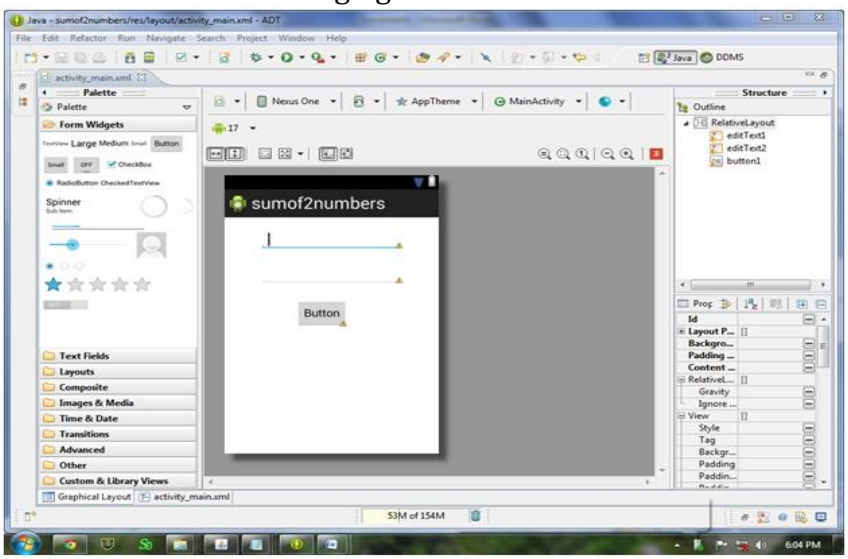
```
TextView myText = (TextView) findViewById(R.id.text_id);
```

Android Button Example

Android Button represents a push-button. The android.widget.Button is subclass of TextView class and CompoundButton is the subclass of Button class.

There are different types of buttons in android such as RadioButton, ToggleButton, CompoundButton etc.

Here, we are going to create two textfields and one button for sum of two numbers. If user clicks button, sum of two input values is displayed on the Toast.
Drag the component or write the code for UI in activity_main.xml
First of all, drag 2 textfields from the Text Fields palette and one button from the Form Widgets palette as shown in the following figure.



The generated code for the ui components will be like this:
File: activity_main.xml

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity" >

  <EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="24dp"
    android:ems="10" />

  <EditText
    android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText1"
    android:layout_below="@+id/editText1"
    android:layout_marginTop="34dp"
    android:ems="10" >
     <requestFocus />
  </EditText>

  <Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/Button" />
    </RelativeLayout>
```

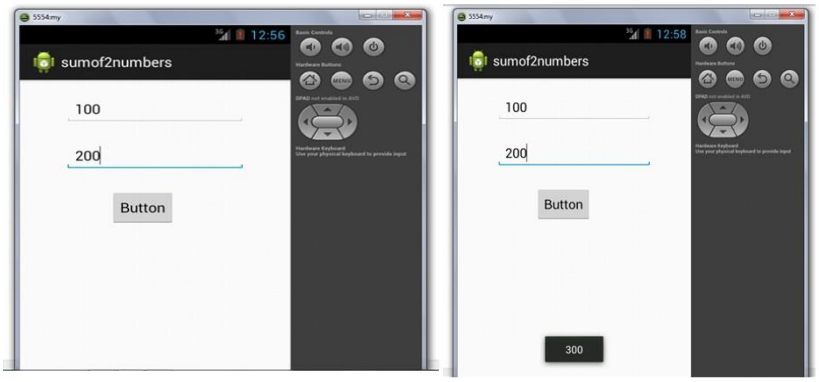
Activity class

Now write the code to display the sum of two numbers.

File: MainActivity.java

```
    package com.example.sumof2numbers;
     import android.os.Bundle;
    import android.widget.Toast;

    public class MainActivity extends Activity {
      private EditText edittext1,edittext2;
      private Button buttonSum;
      @Override
      protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        addListenerOnButton();
         }
      public void addListenerOnButton(){
        edittext1=(EditText)findViewById(R.id.editText1);
        edittext2=(EditText)findViewById(R.id.editText2);
        buttonSum=(Button)findViewById(R.id.button1);

        buttonSum.setOnClickListener(new OnClickListener(){
            @Override
           public void onClick(View view) {
             String value1=edittext1.getText().toString();
             String value2=edittext2.getText().toString();
             int a=Integer.parseInt(value1);
             int b=Integer.parseInt(value2);
             int sum=a+b;
      Toast.makeText(getApplicationContext(),String.valueOf(sum),Toast.LENGTH_LONG).s
    how();
           }
        });
      }
      @Override
      public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
      } }
```

Output:



Android Toast Example

Andorid Toast can be used to display information for the short period of time. A toast contains message to be displayed quickly and disappears after sometime.

The android.widget.Toast class is the subclass of java.lang.Object class.

You can also create custom toast as well for example toast displaying image. You can visit next page to see the code for custom toast.

Toast class

Toast class is used to show notification for a particular interval of time. After sometime it disappears. It doesn't block the user interaction.

Constants of Toast class

There are only 2 constants of Toast class which are given below.

Constant	Description
public static final int LENGTH_LONG	displays view for the long duration of time.
public static final int LENGTH_SHORT	displays view for the short duration of time.

Methods of Toast class

The widely used methods of Toast class are given below.

Method	Description
public static Toast makeText(Context context, CharSequence text, int duration)	makes the toast containing text and duration.
public void show()	displays toast.
public void setMargin (float horizontalMargin, float verticalMargin)	changes the horizontal and vertical margin difference.

Android Toast Example

Toast.makeText(getApplicationContext(),"Hello Javatpoint",Toast.LENGTH_SHORT).show();

Another code:

```
Toast toast=Toast.makeText(getApplicationContext(),"Hello Javatpoint",Toast.LENGTH_SHORT);
toast.setMargin(50,50);
toast.show();
```

Here, getApplicationContext() method returns the instance of Context.

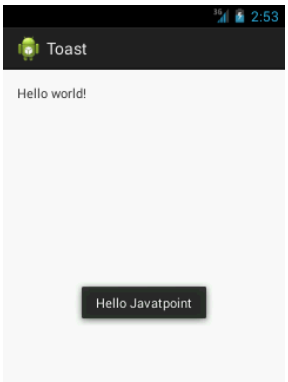
Full code of activity class displaying Toast

Let's see the code to display the toast.

File: MainActivity.java

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

public class MainActivity extends Activity {
    @Override
      public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
              //Displaying Toast with Hello Javatpoint message
        Toast.makeText(getApplicationContext(),"Hello Javatpoint",Toast.LENGTH_SHORT).
show();
      }
       @Override
      public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
      }  }
```

Output:



Android ToggleButton Example

Android Toggle Button can be used to display checked/unchecked (On/Off) state on the button.
It is beneficial if user have to change the setting between two states. It can be used to On/Off Sound, Wifi, Bluetooth etc.
Since Android 4.0, there is another type of toggle button called *switch* that provides slider control.
Android ToggleButton and Switch both are the subclasses of CompoundButton class.
Android ToggleButton class
ToggleButton class provides the facility of creating the toggle button.
XML Attributes of ToggleButton class
The 3 XML attributes of ToggleButton class.

XML Attribute	Description
android:disabledAlpha	The alpha to apply to the indicator when disabled.
android:textOff	The text for the button when it is not checked.
android:textOn	The text for the button when it is checked.

Methods of ToggleButton class
The widely used methods of ToggleButton class are given below.

Method	Description
CharSequence getTextOff()	Returns the text when button is not in the checked state.
CharSequence getTextOn()	Returns the text for when button is in the checked state.
void setChecked(boolean checked)	Changes the checked state of this button.

Android ToggleButton Example

activity_main.xml

Drag two toggle button and one button for the layout. Now the activity_main.xml file will look like this:

File: activity_main.xml

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity" >
   <ToggleButton
    android:id="@+id/toggleButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="60dp"
    android:layout_marginTop="18dp"
    android:text="ToggleButton1"
    android:textOff="Off"
    android:textOn="On" />

  <ToggleButton
    android:id="@+id/toggleButton2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/toggleButton1"
    android:layout_alignBottom="@+id/toggleButton1"
    android:layout_marginLeft="44dp"
    android:layout_toRightOf="@+id/toggleButton1"
    android:text="ToggleButton2"
    android:textOff="Off"
    android:textOn="On" />

  <Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/toggleButton2"
    android:layout_marginTop="82dp"
    android:layout_toRightOf="@+id/toggleButton1"
    android:text="submit" />
```

```
</RelativeLayout>
```

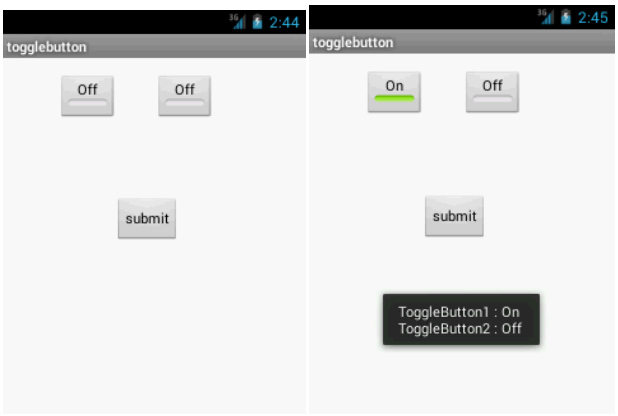
Activity class

Let's write the code to check which toggle button is ON/OFF.

File: MainActivity.java

```
package com.example.togglebutton;
 import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {
   private ToggleButton toggleButton1, toggleButton2;
   private Button buttonSubmit;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      addListenerOnButtonClick();
   }
   public void addListenerOnButtonClick(){
      //Getting the ToggleButton and Button instance from the layout xml file
      toggleButton1=(ToggleButton)findViewById(R.id.toggleButton1);
      toggleButton2=(ToggleButton)findViewById(R.id.toggleButton2);
      buttonSubmit=(Button)findViewById(R.id.button1);

      //Performing action on button click
      buttonSubmit.setOnClickListener(new OnClickListener(){

         @Override
         public void onClick(View view) {
            StringBuilder result = new StringBuilder();
              result.append("ToggleButton1 : ").append(toggleButton1.getText());
              result.append("\nToggleButton2 : ").append(toggleButton2.getText());
            //Displaying the message in toast
            Toast.makeText(getApplicationContext(), result.toString(),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;
   } }
```

Output:



Android Spinner Example

Android Spinner is like the combo box of AWT or Swing. It can be used to display the multiple options to the user in which only one item can be selected by the user.
Android spinner is like the drop down menu with multiple values from which the end user can select only one value.
Android spinner is associated with AdapterView. So you need to use one of the adapter classes with spinner.
Android Spinner class is the subclass of AsbSpinner class.
In this example, we are going to display the country list. You need to use **ArrayAdapter** class to store the country list.
Let's see the simple example of spinner in android.

activity_main.xml

Drag the Spinner from the pallete, now the activity_main.xml file will like this:

File: activity_main.xml

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity" >

  <Spinner
    android:id="@+id/spinner1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="83dp" />
 </RelativeLayout>
```

Activity class

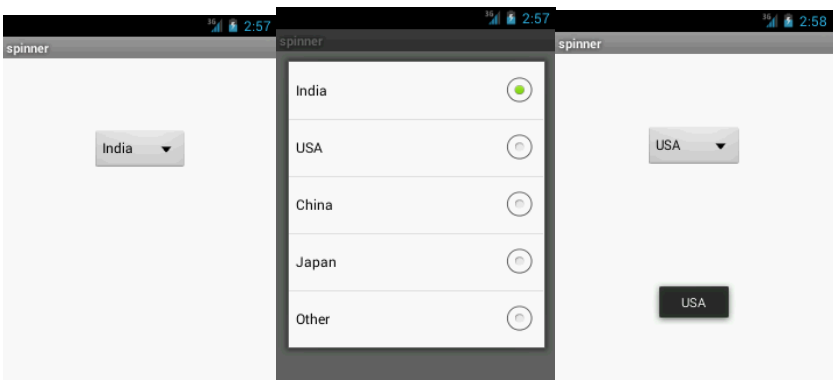
Let's write the code to display item on the spinner and perform event handling.

File: MainActivity.java

```
package com.example.spinner;
import android.widget.Spinner;

public class MainActivity extends Activity implements
AdapterView.OnItemSelectedListener {
```

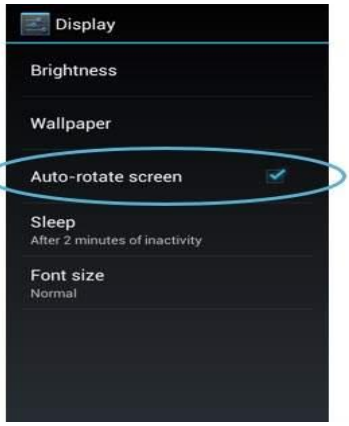
```
        String[] country = { "India", "USA", "China", "Japan", "Other",  };

        @Override
        protected void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           setContentView(R.layout.activity_main);
           //Getting the instance of Spinner and applying OnItemSelectedListener on it
           Spinner spin = (Spinner) findViewById(R.id.spinner1);
           spin.setOnItemSelectedListener(this);

           //Creating the ArrayAdapter instance having the country list
ArrayAdapter aa = new ArrayAdapter(this,android.R.layout.simple_spinner_item,country);
           aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
           //Setting the ArrayAdapter data on the Spinner
           spin.setAdapter(aa);
        }

         //Performing action onItemSelected and onNothing selected
        @Override
        public void onItemSelected(AdapterView<?> arg0, View arg1, int position,long id) {
           Toast.makeText(getApplicationContext(),country[position] ,Toast.LENGTH_LONG).sh
     ow();
        }

        @Override
        public void onNothingSelected(AdapterView<?> arg0) {
           // TODO Auto-generated method stub
         }

        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
           // Inflate the menu; this adds items to the action bar if it is present.
           getMenuInflater().inflate(R.menu.activity_main, menu);
           return true;
        }
     }
```

Output:



Android - CheckBox Control

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

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>
```

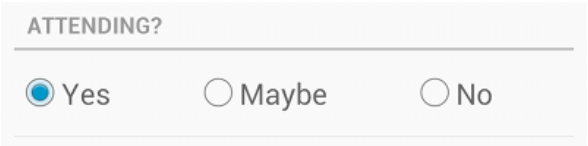
Following will be the content of **res/values/strings.xml** to define these new constants –

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

User needs you check on either do you like android check box or do you like tutorials point check box. and press ok button, if does all process correctly, it gonna be shown toast message as Thanks. Or else do press on cancel button, if user presses cancel button it going to close the application.

Android - RadioButton Control

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



Radio Button

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 tutorialspoint

```
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" />
```

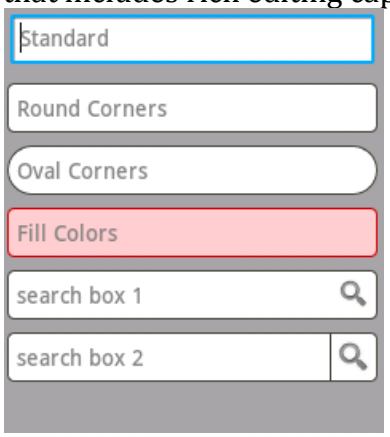
```
<TextView
  android:id="@+id/textView2"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Tutorials point"
  android:textColor="#ff87ff09"
  android:textSize="30dp"
  android:layout_above="@+id/imageButton"
  android:layout_centerHorizontal="true"
  android:layout_marginBottom="40dp"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

<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>
```

If User selected any of a Radio Button, It should give same name on Toast message. for suppose, if User selected JAVA, it gives a toast message as JAVA.

Android - 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.



Styles of edit text

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>
```

Android - Date Picker

Android Date Picker allows you to select the date consisting of day, month and year in your custom user interface. For this functionality android provides DatePicker and DatePickerDialog components.

In this tutorial, we are going to demonstrate the use of Date Picker through DatePickerDialog. DatePickerDialog is a simple dialog containing DatePicker.

In order to show DatePickerDialog , you have to pass the DatePickerDialog id to *showDialog(id_of_dialog)* method. Its syntax is given below −

```
showDialog(999);
```

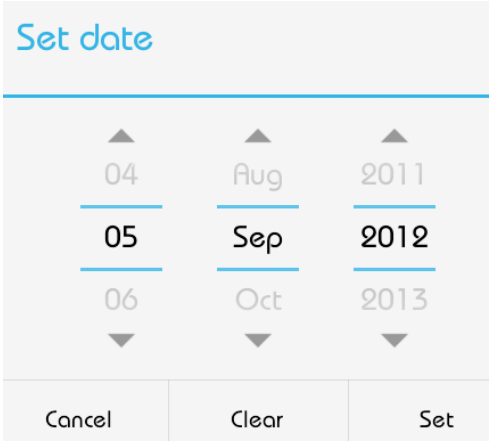
On calling this *showDialog* method, another method called *onCreateDialog* gets automatically called. So we have to override that method too. Its syntax is given below −

```
@Override
protected Dialog onCreateDialog(int id) {
  // TODO Auto-generated method stub
  if (id == 999) {
    return new DatePickerDialog(this, myDateListener, year, month, day);
```

```
  }
  return null;
}
```

In the last step, you have to register the DatePickerDialog listener and override its onDateSet method. This onDateSet method contains the updated day, month and year. Its syntax is given below –

```
private DatePickerDialog.OnDateSetListener myDateListener = new
DatePickerDialog.OnDateSetListener() {
  @Override
  public void onDateSet(DatePicker arg0, int arg1, int arg2, int arg3) {
    // arg1 = year
    // arg2 = month
    // arg3 = day
  }
};
```



Apart form date attributes, DatePicker object is also passed into this function. You can use the following methods of the DatePicker to perform further operation.

Sr.No	Method & description
1	**getDayOfMonth()** This method gets the selected day of month
2	**getMonth()** This method gets the selected month
3	**getYear()** This method gets the selected year
4	**setMaxDate(long maxDate)** This method sets the maximal date supported by this DatePicker in milliseconds since January 1, 1970 00:00:00 in getDefault() time zone
5	**setMinDate(long minDate)** This method sets the minimal date supported by this NumberPicker in milliseconds since January 1, 1970 00:00:00 in getDefault() time zone
6	**setSpinnersShown(boolean shown)** This method sets whether the spinners are shown
7	**updateDate(int year, int month, int dayOfMonth)** This method updates the current date
8	**getCalendarView()** This method returns calendar view
9	**getFirstDayOfWeek()**

This Method returns first day of the week

Example

Here is an example demonstrating the use of DatePickerDialog class. It creates a basic Date Picker application that allows you to set the Date using DatePicker Widget

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 to create an Android application and name it as DatePicker under a package com.example.datepicker.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
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.datepicker/MainActivity.java**.

```
package com.example.datepicker;
import java.util.Calendar;
import android.app.Activity;
import android.app.DatePickerDialog;
import android.app.Dialog;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.DatePicker;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {
  private DatePicker datePicker;
  private Calendar calendar;
  private TextView dateView;
  private int year, month, day;

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

    dateView = (TextView) findViewById(R.id.textView3);
    calendar = Calendar.getInstance();
    year = calendar.get(Calendar.YEAR);

    month = calendar.get(Calendar.MONTH);
    day = calendar.get(Calendar.DAY_OF_MONTH);
    showDate(year, month+1, day);
  }

  @SuppressWarnings("deprecation")
```

```
  public void setDate(View view) {
    showDialog(999);
    Toast.makeText(getApplicationContext(), "ca",
      Toast.LENGTH_SHORT)
    .show();
  }
  @Override
  protected Dialog onCreateDialog(int id) {
    // TODO Auto-generated method stub
    if (id == 999) {
      return new DatePickerDialog(this,
        myDateListener, year, month, day);
    }
    return null;
  }

  private DatePickerDialog.OnDateSetListener myDateListener = new
    DatePickerDialog.OnDateSetListener() {
    @Override
    public void onDateSet(DatePicker arg0,
      int arg1, int arg2, int arg3) {
      // TODO Auto-generated method stub
      // arg1 = year
      // arg2 = month
      // arg3 = day
      showDate(arg1, arg2+1, arg3);
    }
  };

  private void showDate(int year, int month, int day) {
    dateView.setText(new StringBuilder().append(day).append("/")
    .append(month).append("/").append(year));
  }
}
```

Following is the modified content of the xml **res/layout/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:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context=".MainActivity" >

  <Button
    android:id="@+id/button1"
```

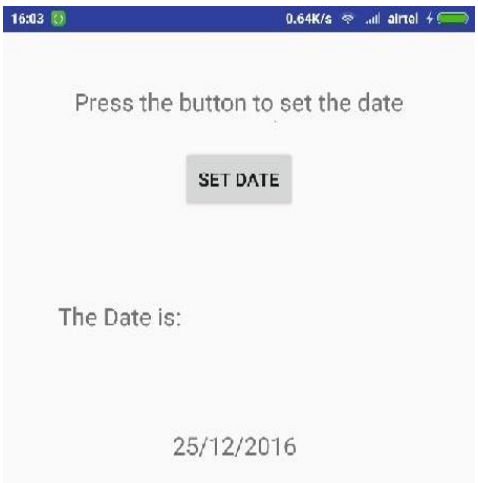
```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="70dp"
    android:onClick="setDate"
    android:text="@string/date_button_set" />
  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="24dp"
    android:text="@string/date_label_set"
    android:textAppearance="?android:attr/textAppearanceMedium" />
  <TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/button1"
    android:layout_marginTop="66dp"
    android:layout_toLeftOf="@+id/button1"
    android:text="@string/date_view_set"
    android:textAppearance="?android:attr/textAppearanceMedium" />
  <TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@+id/button1"
    android:layout_below="@+id/textView2"
    android:layout_marginTop="72dp"
    android:text="@string/date_selected"
    android:textAppearance="?android:attr/textAppearanceMedium" />

</RelativeLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">DatePicker</string>
  <string name="action_settings">Settings</string>
  <string name="hello_world">Hello world!</string>
  <string name="date_label_set">Press the button to set the date</string>
  <string name="date_button_set">Set Date</string>
  <string name="date_view_set">The Date is: </string>
  <string name="date_selected"></string>
</resources>
```

Now you can see that the date has already been set at the bottom label. Now we will change the date through DatePickerDialog by pressing the Set Date button. On pressing the button following screen would appear.

Now set the required date, and after setting the date, press the Done button. This dialog will disappear and your newly setted date will start showing at the screen. This is shown below.

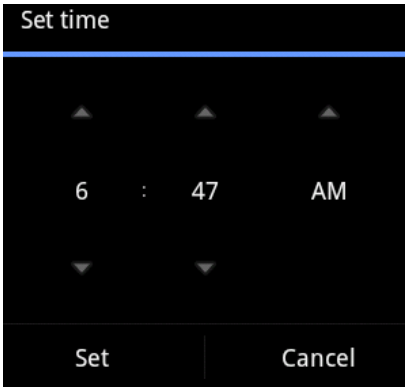


Android - Time Picker

Android Time Picker allows you to select the time of day in either 24 hour or AM/PM mode. The time consists of hours, minutes and clock format. Android provides this functionality through TimePicker class.

In order to use TimePicker class, you have to first define the TimePicker component in your activity.xml. It is define as below –

```
<TimePicker
  android:id="@+id/timePicker1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content" />
```



After that you have to create an object of TimePicker class and get a reference of the above defined xml component. Its syntax is given below.

```
import android.widget.TimePicker;
private TimePicker timePicker1;
timePicker1 = (TimePicker) findViewById(R.id.timePicker1);
```

In order to get the time selected by the user on the screen, you will use getCurrentHour() and getCurrentMinute() method of the TimePicker Class. Their syntax is given below.

```
int hour = timePicker1.getCurrentHour();
int min = timePicker1.getCurrentMinute();
```

Apart form these methods, there are other methods in the API that gives more control over TimePicker Component. They are listed below.

Sr.No	Method & description
1	**is24HourView()** This method returns true if this is in 24 hour view else false
2	**isEnabled()** This method returns the enabled status for this view
3	**setCurrentHour(Integer currentHour)** This method sets the current hour
4	**setCurrentMinute(Integer currentMinute)** This method sets the current minute
5	**setEnabled(boolean enabled)** This method set the enabled state of this view
6	**setIs24HourView(Boolean is24HourView)** This method set whether in 24 hour or AM/PM mode
7	**setOnTimeChangedListener(TimePicker.OnTimeChangedListener onTimeChangedListener)** This method Set the callback that indicates the time has been adjusted by the user

Example

Here is an example demonstrating the use of TimePicker class. It creates a basic Time Picker application that allows you to set the time using TimePicker Widget

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 to create an Android application and name it as TimePicker under a package com.example.timepicker.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components

4	Modify the res/values/string.xml to add necessary string components
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.timepicker/MainActivity.java**.

```
package com.example.timepicker;
import java.util.Calendar;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.TextView;
import android.widget.TimePicker;

public class MainActivity extends Activity {
  private TimePicker timePicker1;
  private TextView time;
  private Calendar calendar;
  private String format = "";

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

    timePicker1 = (TimePicker) findViewById(R.id.timePicker1);
    time = (TextView) findViewById(R.id.textView1);
    calendar = Calendar.getInstance();

    int hour = calendar.get(Calendar.HOUR_OF_DAY);
    int min = calendar.get(Calendar.MINUTE);
    showTime(hour, min);
  }

  public void setTime(View view) {
    int hour = timePicker1.getCurrentHour();
    int min = timePicker1.getCurrentMinute();
    showTime(hour, min);
  }


 public void showTime(int hour, int min) {
    if (hour == 0) {
      hour += 12;
      format = "AM";
    } else if (hour == 12) {
      format = "PM";
    } else if (hour > 12) {
      hour -= 12;
      format = "PM";
    } else {
```

```
      format = "AM";
    }

    time.setText(new StringBuilder().append(hour).append(" : ").append(min)
    .append(" ").append(format));
  }


}
```

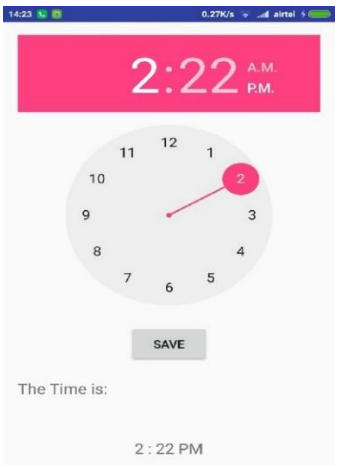
Following is the modified content of the xml **res/layout/activity_main.xml**.

```
<?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: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:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:text="@string/time_pick"
    android:textAppearance="?android:attr/textAppearanceMedium" />

  <Button
    android:id="@+id/set_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="180dp"
    android:onClick="setTime"
    android:text="@string/time_save" />


 <TimePicker
    android:id="@+id/timePicker1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/set_button"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="24dp" />
      <TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/timePicker1"
    android:layout_alignTop="@+id/set_button"
    android:layout_marginTop="67dp"
    android:text="@string/time_current"
    android:textAppearance="?android:attr/textAppearanceMedium" />
  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView3"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="50dp"
    android:text="@string/time_selected"
    android:textAppearance="?android:attr/textAppearanceMedium" />
</RelativeLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">TimePicker</string>
  <string name="action_settings">Settings</string>
  <string name="time_picker_example">Time Picker Example</string>
  <string name="time_pick">Pick the time and press save button</string>
  <string name="time_save">Save</string>
  <string name="time_selected"></string>
  <string name="time_current">The Time is:</string>
</resources>
```



Android Progress Bar using ProgressDialog

Progress bars are used to show progress of a task. For example, when you are uploading or downloading something from the internet, it is better to show the progress of download/upload to the user.

In android there is a class called ProgressDialog that allows you to create progress bar. In order to do this, you need to instantiate an object of this class. Its syntax is.

```
ProgressDialog progress = new ProgressDialog(this);
```

Now you can set some properties of this dialog. Such as, its style, its text etc.

```
progress.setMessage("Downloading Music :) ");
progress.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progress.setIndeterminate(true);
```

Apart from these methods, there are other methods that are provided by the ProgressDialog class

Sr. No	Title & description
1	**getMax()** This method returns the maximum value of the progress.
2	**incrementProgressBy(int diff)** This method increments the progress bar by the difference of value passed as a parameter.
3	**setIndeterminate(boolean indeterminate)** This method sets the progress indicator as determinate or indeterminate.
4	**setMax(int max)** This method sets the maximum value of the progress dialog.
5	**setProgress(int value)** This method is used to update the progress dialog with some specific value.
6	**show(Context context, CharSequence title, CharSequence message)** This is a static method, used to display progress dialog.

Example

This example demonstrates the horizontal use of the progress dialog which is in fact a progress bar. It display a progress bar on pressing the button.

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

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 progress code to display the progress dialog.
3	Modify res/layout/activity_main.xml file to add respective XML code.
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.ProgressDialog;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

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

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    b1 = (Button) findViewById(R.id.button2);
  }

  public void download(View view){
```

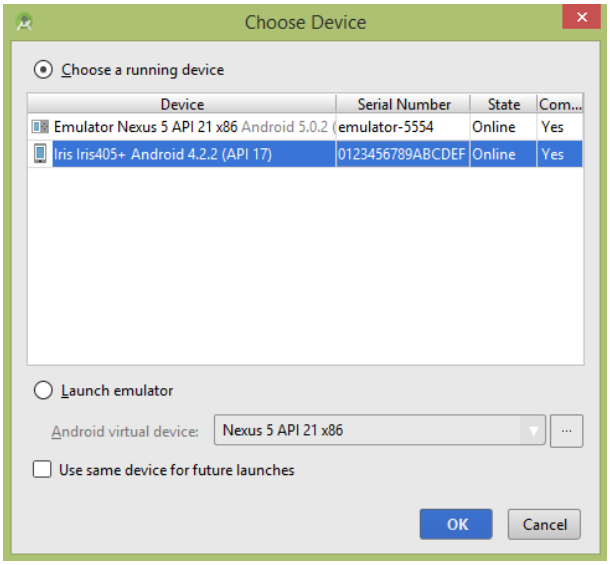
```
    progress=new ProgressDialog(this);
    progress.setMessage("Downloading Music");
    progress.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
    progress.setIndeterminate(true);
    progress.setProgress(0);
    progress.show();

    final int totalProgressTime = 100;
    final Thread t = new Thread() {
      @Override
      public void run() {
        int jumpTime = 0;

        while(jumpTime < totalProgressTime) {
          try {
            sleep(200);
            jumpTime += 5;
            progress.setProgress(jumpTime);
          } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
          }
        }
      }
    };
    t.start();
  }
}
```

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

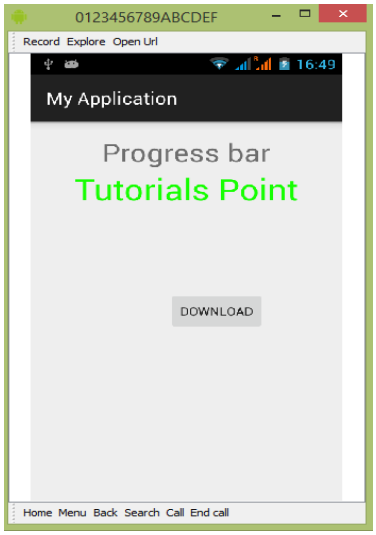
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
  android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  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:id="@+id/textView"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="30dp"
    android:text="Progress bar" />

  <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="35dp"
    android:textColor="#ff16ff01" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Download"
    android:onClick="download"
    android:id="@+id/button2"
    android:layout_marginLeft="125dp"
    android:layout_marginStart="125dp"
    android:layout_centerVertical="true" />

</RelativeLayout>
```

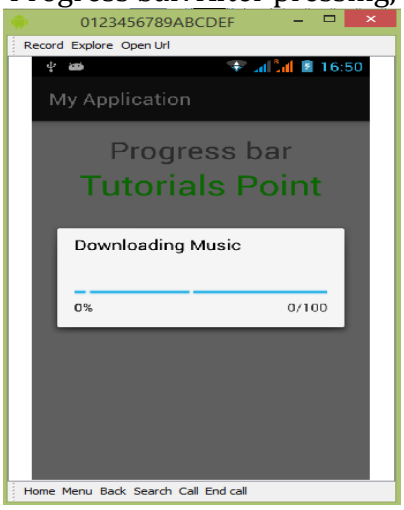
Let's try to run your application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Before starting your application, Android studio will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen –



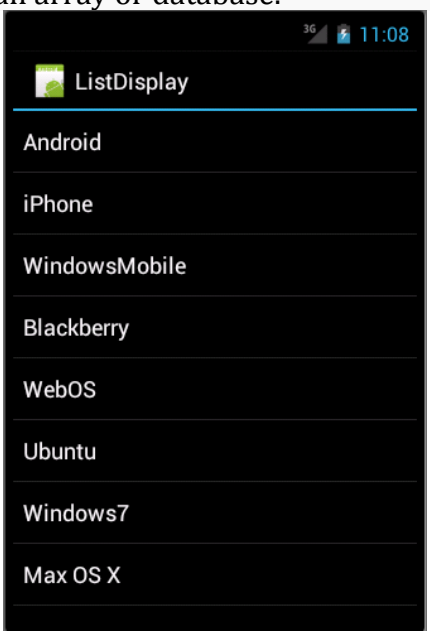
Just press the button to start the Progress bar. After pressing, following screen would appear



It will continuously update itself.

Android List View

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.



LIST VIEW

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 –

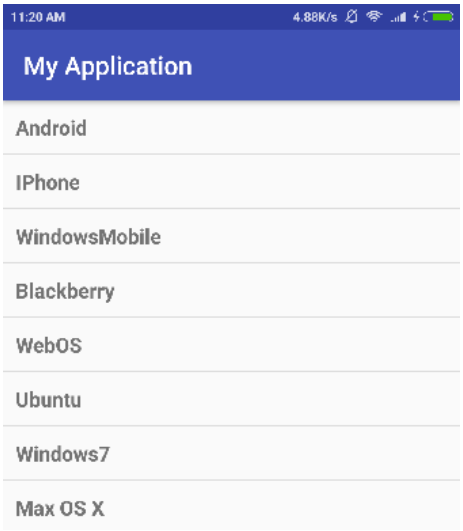
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  tools:context=".ListActivity" >

  <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>
```



Android Analog clock and Digital clock example

The **android.widget.AnalogClock** and **android.widget.DigitalClock** classes provides the functionality to display analog and digital clocks.

Android analog and digital clocks are used to show time in android application.

Android AnalogClock is the subclass of View class.

Android DigitalClock is the subclass of TextView class. Since Android API level 17, it is *deprecated*. You are recommended to use **TextClock** Instead.

The AnalogClock was deprecated in API level 23. This widget is no longer supported. Instead if you want to use AnalogClock in your application you need to hard code. It does not appear in API level 27 to drag from palette.

Note: Analog and Digital clocks cannot be used to change the time of the device. To do so, you need to use DatePicker and TimePicker.

In android, you need to drag analog and digital clocks from the pallet to display analog and digital clocks.

activity_main.xml

Now, drag the analog and digital clocks, now the xml file will look like this.

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.analogdigital.MainActivity">

    <AnalogClock
        android:id="@+id/analogClock1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginLeft="136dp"
        android:layout_marginTop="296dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

```
19.
20.    <DigitalClock
21.        android:id="@+id/digitalClock1"
22.        android:layout_width="wrap_content"
23.        android:layout_height="wrap_content"
24.        android:layout_below="@+id/analogClock1"
25.        android:layout_centerHorizontal="true"
26.        android:layout_marginLeft="176dp"
27.        android:layout_marginTop="84dp"
28.        android:text="DigitalClock"
29.        app:layout_constraintStart_toStartOf="parent"
30.        app:layout_constraintTop_toTopOf="parent" />
31.
32. </android.support.constraint.ConstraintLayout>
```

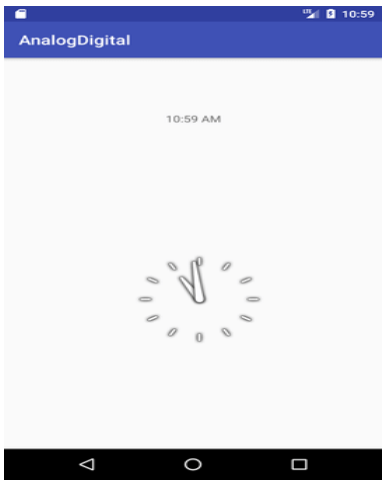
Activity class

We have not write any code here.

File: MainActivity.java

```
1.  package example.javatpoint.com.analogdigital;
2.
3.  import android.support.v7.app.AppCompatActivity;
4.  import android.os.Bundle;
5.
6.  public class MainActivity extends AppCompatActivity {
7.
8.      @Override
9.      protected void onCreate(Bundle savedInstanceState) {
10.         super.onCreate(savedInstanceState);
11.         setContentView(R.layout.activity_main);
12.     }
13. }
```

Output:



Android WebView Example

Android WebView is used to display web page in android. The web page can be loaded from same application or URL. It is used to display online content in android activity.

Android WebView uses webkit engine to display web page. The android.webkit.WebView is the subclass of AbsoluteLayout class. The loadUrl() and loadData() methods of Android WebView class are used to load and display web page.

Android WebView Example

Let's see the simple code to **display javatpoint.com web page** using web view.

```
1. WebView mywebview = (WebView) findViewById(R.id.webView1);
2. mywebview.loadUrl("http://www.javatpoint.com/");
```

Let's see the simple code to **display HTML web page** using web view. In this case, html file must be located inside the asset directory.

```
1. WebView mywebview = (WebView) findViewById(R.id.webView1);
2. mywebview.loadUrl("file:///android_asset/myresource.html");
```

Let's see another code to display **HTML code of a string**.

```
1. String data = "<html><body><h1>Hello, Javatpoint!</h1></body></html>";
2. mywebview.loadData(data, "text/html", "UTF-8");
```

Complete Android WebView Example

Let's see a complete example of Android WebView.

activity_main.xml

File: activity_main.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     xmlns:tools="http://schemas.android.com/tools"
5.     android:layout_width="match_parent"
6.     android:layout_height="match_parent"
7.     tools:context="example.javatpoint.com.webview.MainActivity">
8.
9.     <WebView
10.        android:layout_width="match_parent"
11.        android:layout_height="match_parent"
12.        android:id="@+id/webView"
13.        app:layout_constraintBottom_toBottomOf="parent"
14.        app:layout_constraintLeft_toLeftOf="parent"
15.        app:layout_constraintRight_toRightOf="parent"
```

```
16.        app:layout_constraintTop_toTopOf="parent" />
17.    </android.support.constraint.ConstraintLayout>
```

To add the web page (.html, .jsp) locally in application, they are required to place in the assets folder. An assets folder is created as: right click on app -> New -> Folder -> Assets Folder ->main or simply create an assets directory inside main directory.

Activity class

File: MainActivity.java

```
1.  package example.javatpoint.com.webview;
2.
3.  import android.support.v7.app.AppCompatActivity;
4.  import android.os.Bundle;
5.  import android.webkit.WebView;
6.
7.  public class MainActivity extends AppCompatActivity {
8.
9.      @Override
10.     protected void onCreate(Bundle savedInstanceState) {
11.         super.onCreate(savedInstanceState);
12.         setContentView(R.layout.activity_main);
13.         WebView mywebview = (WebView) findViewById(R.id.webView);
14.         // mywebview.loadUrl("https://www.javatpoint.com/");
15.
16.         /*String data = "<html><body><h1>Hello, Javatpoint!</h1></body></html>";
17.         mywebview.loadData(data, "text/html", "UTF-8"); */
18.
19.         mywebview.loadUrl("file:///android_asset/myresource.html");
20.     }
21. }
```

Output:

Let's see the output if you load the HTML page.

Let's see the output if you load the javatpoint.com web page.

Android - Event Handling

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

There are 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 tutorialspoint 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>
```

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.

Android - list Fragment

Static library support version of the framework's ListFragment. Used to write apps that run on platforms prior to Android 3.0. When running on Android 3.0 or above, this implementation is still used.

The basic implementation of list fragment is for creating list of items in fragments

List in Fragments

Example

This example will explain you how to create your own list fragment based on arrayAdapter. So let's follow the following steps to similar to what we followed while creating Hello World Example –

Step	Description
1	You will use Android Studio to create an Android application and name it as *SimpleListFragment* under a package *com.example.tutorialspoint7.myapplication*, with blank Activity.
2	Modify the string file, which has placed at *res/values/string.xml* to add new string constants
3	Create a layout called *list_fragment.xml* under the directory *res/layout* to define your list fragments. and add fragment tag(<fragment>) to your activity_main.xml
4	Create a myListFragment.java, which is placed at *java/myListFragment.java* and it contained *onCreateView(),onActivityCreated()* and *OnItemClickListener()*
5	Run the application to launch Android emulator and verify the result of the changes done in the application.

Before start coding i will initialize of the string constants inside *string.xml* file under *res/values directory*

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

  <string-array name="Planets">
    <item>Sun</item>
    <item>Mercury</item>
    <item>Venus</item>
    <item>Earth</item>
    <item>Mars</item>
    <item>Jupiter</item>
```

```
    <item>Saturn</item>
    <item>Uranus</item>
    <item>Neptune</item>
  </string-array>

</resources>
```

Following will be the content of **res/layout/activity_main.xml** file. it contained linear layout and fragment tag.

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

  <fragment
    android:id="@+id/fragment1"
    android:name="com.example.tutorialspoint7.myapplication.MyListFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

</LinearLayout>
```

Following will be the content of **res/layout/list_fragment.xml** file. it contained linear layout,list view and text view

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

  <ListView
    android:id="@android:id/list"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
  </ListView>

  <TextView
    android:id="@android:id/empty"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
  </TextView>
</LinearLayout>
```

following will be the content of **src/main/java/myListFragment.java** file.before writing to code, need to follow few steps as shown below

- Create a class MyListFragment and extend it to ListFragment.
- Inside the **onCreateView()** method , inflate the view with above defined list_fragment xml layout.
- Inside the **onActivityCreated()** method , create a arrayadapter from resource ie using String array R.array.planet which you can find inside the string.xml and set this adapter to listview and also set the onItem click Listener.

- Inside the **OnItemClickListener()** method , display a toast message with Item name which is being clicked.

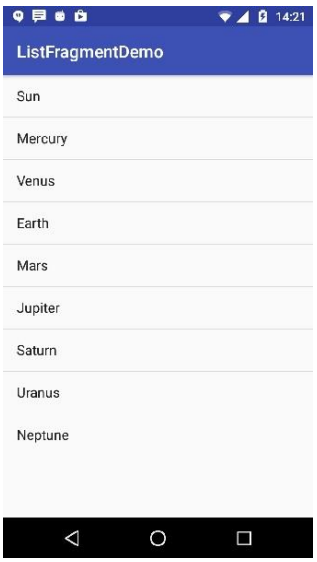
```
package com.example.tutorialspoint7.myapplication;
import android.annotation.SuppressLint;
import android.app.ListFragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Toast;

public class MyListFragment extends ListFragment implements OnItemClickListener {
  @Override
  public View onCreateView(LayoutInflater inflater,
    ViewGroup container, Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.list_fragment, container, false);
    return view;
  }

  @Override
  public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    ArrayAdapter adapter = ArrayAdapter.createFromResource(getActivity(),
      R.array.Planets, android.R.layout.simple_list_item_1);
    setListAdapter(adapter);
    getListView().setOnItemClickListener(this);
  }

  @Override
  public void onItemClick(AdapterView<?> parent, View view, int position,long id) {
    Toast.makeText(getActivity(), "Item: " + position, Toast.LENGTH_SHORT).show();
  }
}
```

Following code will be the content of MainActivity.java

```
package com.example.tutorialspoint7.myapplication;

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);
  }
}
```

following code will be the content of manifest.xml, which has placed at res/AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.tutorialspoint7.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>
```



Android DialogFragment Tutorial

In this tutorial, you will learn how to implement a DialogFragment in your Android application. A DialogFragment is a fragment that displays a dialog window, floating on top of its activity's window. This fragment contains a Dialog object, which it displays as appropriate based on the fragment's state. Alternatively, you can create an entirely custom dialog, such as an AlertDialog, with its own content. We will create buttons that will show a DialogFragment and a custom Alert DialogFragment.

Create a new project in Eclipse **File** > **New** > **Android Application Project.** Fill in the details and name your project **DialogFragmentTutorial.**
Application Name : **DialogFragmentTutorial**
Project Name : **DialogFragmentTutorial**
Package Name : **com.androidbegin.dialogfragmenttutorial**
Open your **MainActivity.java** and paste the following code.
MainActivity.java

```
1
2  package com.androidbegin.dialogfragmenttutorial;
3   import android.os.Bundle;
4  import android.support.v4.app.FragmentActivity;
5  import android.support.v4.app.FragmentManager;
6  import android.view.View;
7  import android.view.View.OnClickListener;
8  import android.widget.Button;
9
10 public class MainActivity extends FragmentActivity {
11
12 Button dfragbutton;
13 Button alertdfragbutton;
14 FragmentManager fm = getSupportFragmentManager();
15
16 @Override
17 public void onCreate(Bundle savedInstanceState) {
18 super.onCreate(savedInstanceState);
19 // Get the view from activity_main.xml
20 setContentView(R.layout.activity_main);
21
22 // Locate the button in activity_main.xml
23 dfragbutton = (Button) findViewById(R.id.dfragbutton);
24 alertdfragbutton = (Button) findViewById(R.id.alertdfragbutton);
25
26 // Capture button clicks
27 dfragbutton.setOnClickListener(new OnClickListener() {
28 public void onClick(View arg0) {
29 DFragment dFragment = new DFragment();
30 // Show DialogFragment
31 dFragment.show(fm, "Dialog Fragment");
32 }
33 });
34
35 // Capture button clicks
36 alertdfragbutton.setOnClickListener(new OnClickListener() {
37 public void onClick(View arg0) {
38 AlertDFragment alertdFragment = new AlertDFragment();
39 // Show Alert DialogFragment
40 alertdFragment.show(fm, "Alert Dialog Fragment");
41 }
42 });
43 }
44 }
```

In this activity, we have created two buttons that will show different dialog fragments. The first button will show a simple DialogFragment and another with a custom Alert DialogFragment.

Next, create an XML graphical layout for the MainActivity. Go to **res** > **layout** > Right Click on **layout** > **New** > **Android XML File**

Name your new XML file **activity_main.xml** and paste the following code.

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" >

   <Button
      android:id="@+id/dfragbutton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerInParent="true"
      android:text="@string/dialogfragment" />

   <Button
      android:id="@+id/alertdfragbutton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_below="@+id/dfragbutton"
      android:layout_centerInParent="true"
      android:text="@string/alertdialogfragment" />

</RelativeLayout>
```

Next, create a fragment. Go to **File** > **New** > **Class** and name it **DFragment.java.** Select your package named **com.androidbegin.dialogfragmenttutorial** and click **Finish**.

Open your **DFragment.java** and paste the following code.

DFragment.java

```
package com.androidbegin.dialogfragmenttutorial;

import android.os.Bundle;
import android.support.v4.app.DialogFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class DFragment extends DialogFragment {
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
View rootView = inflater.inflate(R.layout.dialogfragment, container,
false);
getDialog().setTitle("DialogFragment Tutorial");
// Do something else
return rootView;
}
}
```

Next, create an XML graphical layout for the DialogFragment. Go to **res** > **layout** > Right Click on **layout** > **New** > **Android XML File**

Name your new XML file **dialogfragment.xml** and paste the following code.

dialogfragment.xml

```
1   <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent" >
5
6      <TextView
7         android:layout_width="wrap_content"
8         android:layout_height="wrap_content"
9         android:layout_centerInParent="true"
10        android:padding="10dp"
11        android:text="@string/welcome" />
12
13 </RelativeLayout>
```

Next, create another fragment. Go to **File** > **New** > **Class** and name it **AlertDFragment.java.** Select your package named **com.androidbegin.dialogfragmenttutorial** and click **Finish**.
Open your **AlertDFragment.java** and paste the following code.

AlertDFragment.java

```
1   package com.androidbegin.dialogfragmenttutorial;
2
3   import android.app.AlertDialog;
4   import android.app.Dialog;
5   import android.content.DialogInterface;
6   import android.os.Bundle;
7   import android.support.v4.app.DialogFragment;
8
9   public class AlertDFragment extends DialogFragment {
10 @Override
11 public Dialog onCreateDialog(Bundle savedInstanceState) {
12 return new AlertDialog.Builder(getActivity())
13 // Set Dialog Icon
14 .setIcon(R.drawable.androidhappy)
15 // Set Dialog Title
16 .setTitle("Alert DialogFragment")
17 // Set Dialog Message
18 .setMessage("Alert DialogFragment Tutorial")
19
20 // Positive button
21 .setPositiveButton("OK", new DialogInterface.OnClickListener() {
22 public void onClick(DialogInterface dialog, int which) {
23 // Do something else
24 }
25 })
26
27 // Negative Button
28 .setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
29 public void onClick(DialogInterface dialog, int which) {
30 // Do something else
31 }
32 }).create();
33 }
```

```
34 }
```

In this fragment, we have created custom alert DialogFragment that consist of two buttons, a positive and negative. We have prepared a sample alert dialog icon for this tutorial. Insert your downloaded sample alert dialog icon into your **res** > **drawable-hdpi.**

Sample Alert Dialog Icon

Androidhappy (1.2 KiB, 1861 downloads)

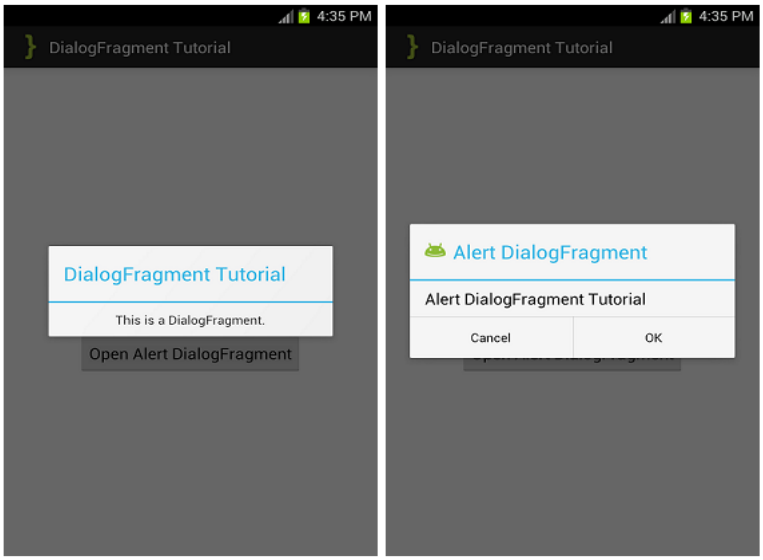
Next, change the application name and texts. Open your **strings.xml** in your **res > values** folder and paste the following code.

strings.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4      <string name="app_name">DialogFragment Tutorial</string>
5      <string name="action_settings">Settings</string>
6      <string name="hello_world">Hello world!</string>
7      <string name="welcome">This is a DialogFragment.</string>
8      <string name="dialogfragment">Open DialogFragment</string>
9      <string name="alertdialogfragment">Open Alert DialogFragment</string>
10 </resources>
```

No changes needed for the AndroidManifest.xml.

Output:



Source Code

Adroid ButtonClick / Android AlertDialog / Android Development / Android DialogFragment / Android Tutorial

Android Option Menu Example

Android Option Menus are the primary menus of android. They can be used for settings, search, delete item etc.

Here, we are going to see two examples of option menus. First, the simple option menus and second, options menus with images.

Here, we are inflating the menu by calling the **inflate()** method of **MenuInflater** class. To perform event handling on menu items, you need to override **onOptionsItemSelected()** method of Activity class.

Android Option Menu Example

Let's see how to create menu in android. Let's see the simple option menu example that contains three menu items.

activity_main.xml

We have only one textview in this file.

File: activity_main.xml

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

menu_main.xml

It contains three items as show below. It is created automatically inside the res/menu directory.

File: menu_main.xml

```
<menu xmlns:androclass="http://schemas.android.com/apk/res/android" >
  <item  android:id="@+id/item1"
    android:title="Item 1"/>
  <item  android:id="@+id/item2"
    android:title="Item 2"/>
  <item  android:id="@+id/item3"
    android:title="Item 3"/>
</menu>
```

Activity class

This class displays the content of menu.xml file and performs event handling on clicking the menu items.

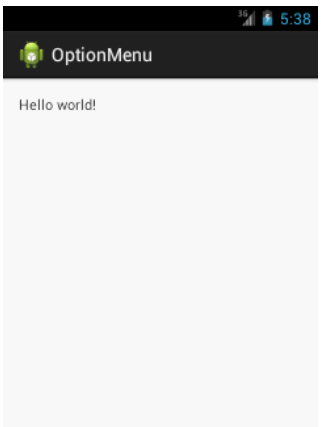
File: MainActivity.java

```
package com.javatpoint.optionmenu;
import android.os.Bundle;
import android.view.MenuItem;
import android.widget.Toast;
public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
  @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);//Menu Resource, Menu
    return true;
  }
  @Override
```

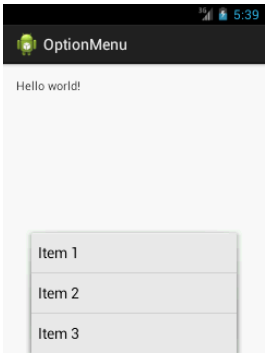
```
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.item1:
             Toast.makeText(getApplicationContext(),"Item 1 Selected",Toast.LENGTH_LONG).s
how();
             return true;
            case R.id.item2:
              Toast.makeText(getApplicationContext(),"Item 2 Selected",Toast.LENGTH_LONG).
show();
             return true;
            case R.id.item3:
              Toast.makeText(getApplicationContext(),"Item 3 Selected",Toast.LENGTH_LONG).
show();
             return true;
             default:
              return super.onOptionsItemSelected(item);
        }
    }
}
```

Output:

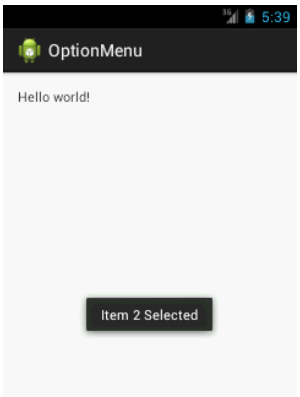
Output without clicking on the menu button.



Output after clicking on the menu button.



Output after clicking on the second menu item .



Option Menu with Icon

You need to have icon images inside the res/drawable directory. The android:icon element is used to display the icon on the option menu. You can write the string information in the strings.xml file. But we have written it inside the menu_main.xml file.

File: menu_main.xml

```
<menu xmlns:androclass="http://schemas.android.com/apk/res/android" >
  <item  android:id="@+id/item1"
    android:icon="@drawable/add"
    android:title="Item 1"/>
  <item  android:id="@+id/item2"
    android:icon="@drawable/minus"
    android:title="Item 2"/>
  <item  android:id="@+id/item3"
    android:icon="@drawable/delete"
    android:title="Item 3"/>
</menu>
```

Android Context Menu Example

Android context menu appears when user press long click on the element. It is also known as floating menu.

It doesn't support item shortcuts and icons.

Android Context Menu Example

Let's see the simple example of context menu in android.

activity_main.xml

Drag one listview from the pallete, now the xml file will look 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"
  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" >

  <ListView
    android:id="@+id/listView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="66dp"
    android:layout_marginTop="53dp" >
  </ListView>
</RelativeLayout>
```

Activity class

Let's write the code to display the context menu on press of the listview.

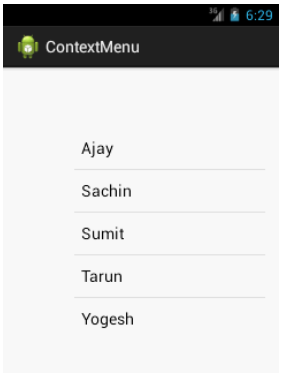
File: MainActivity.java

```
package com.javatpoint.contextmenu;
```

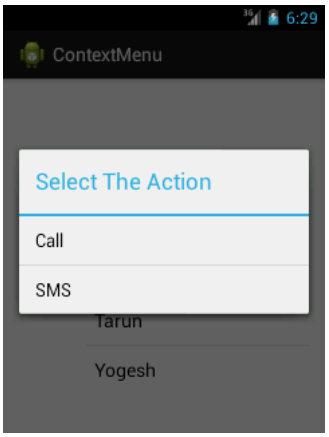
```
import android.os.Bundle;
import android.widget.Toast;
public class MainActivity extends Activity {
   ListView listView1;
   String contacts[]={"Ajay","Sachin","Sumit","Tarun","Yogesh"};
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      listView1=(ListView)findViewById(R.id.listView1);
      ArrayAdapter<String> adapter=new ArrayAdapter<String>(this,android.R.layout.sim
ple_list_item_1,contacts);
      listView1.setAdapter(adapter);
      // Register the ListView  for Context menu
      registerForContextMenu(listView1);
   }
   @Override
   public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menu
Info)
   {
         super.onCreateContextMenu(menu, v, menuInfo);
         menu.setHeaderTitle("Select The Action");
         menu.add(0, v.getId(), 0, "Call");//groupId, itemId, order, title
         menu.add(0, v.getId(), 0, "SMS");
   }
   @Override
   public boolean onContextItemSelected(MenuItem item){
         if(item.getTitle()=="Call"){
     Toast.makeText(getApplicationContext(),"calling code",Toast.LENGTH_LONG).show();
          }
         else if(item.getTitle()=="SMS"){
            Toast.makeText(getApplicationContext(),"sending sms code",Toast.LENGTH_LON
G).show();
         }else{
```

```
            return false;
          }
        return true;
      }}
```

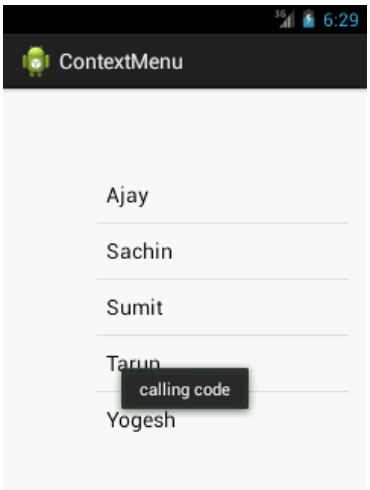
Output:



Output after long press on the listview.



Output after clicking on the context menu.



Android Popup Menu Example

Android Popup Menu displays the menu below the anchor text if space is available otherwise above the anchor text. It disappears if you click outside the popup menu.

The android.widget.PopupMenu is the direct subclass of java.lang.Object class.

Android Popup Menu Example

Let's see how to create popup menu in android.

activity_main.xml

It contains only one button.

File: activity_main.xml

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

  <Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="62dp"
    android:layout_marginTop="50dp"
    android:text="Show Popup" />
</RelativeLayout>
```

popup_menu.xml

It contains three items as show below. It is created inside the res/menu directory.

File: poupup_menu.xml

```
<menu xmlns:androclass="http://schemas.android.com/apk/res/android" >
  <item
```

```
        android:id="@+id/one"
        android:title="One"/>
    <item
        android:id="@+id/two"
        android:title="Two"/>
    <item
        android:id="@+id/three"
        android:title="Three"/>
</menu>
```

Activity class

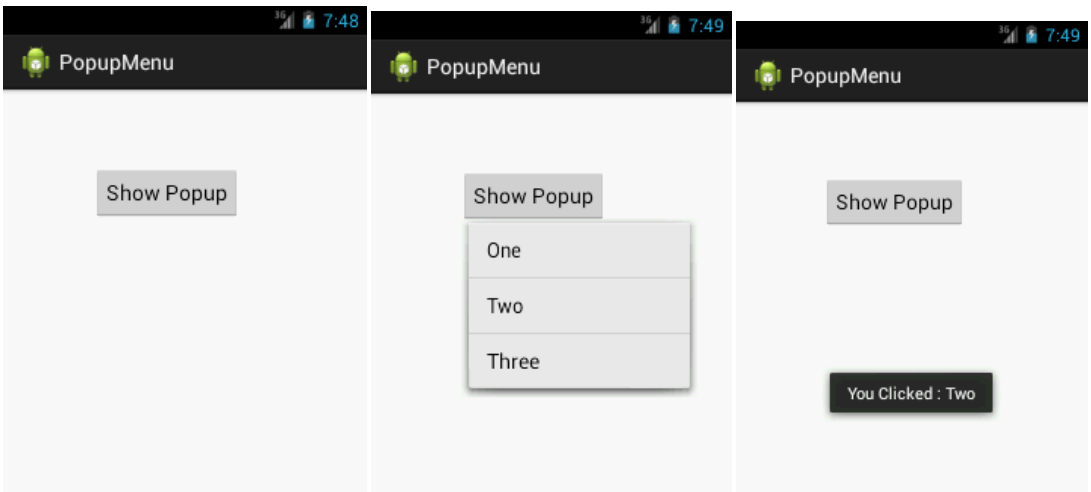
It displays the popup menu on button click.

File: MainActivity.java

```
package com.javatpoint.popupmenu;
import android.widget.Toast;
public class MainActivity extends Activity {
Button button1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);
     button1 = (Button) findViewById(R.id.button1);
     button1.setOnClickListener(new OnClickListener() {

      @Override
      public void onClick(View v) {
       //Creating the instance of PopupMenu
       PopupMenu popup = new PopupMenu(MainActivity.this, button1);
       //Inflating the Popup using xml file
       popup.getMenuInflater().inflate(R.menu.popup_menu, popup.getMenu());

       //registering popup with OnMenuItemClickListener
```

```
        popup.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener()
    {
            public boolean onMenuItemClick(MenuItem item) {
             Toast.makeText(MainActivity.this,"You Clicked : " + item.getTitle(),Toast.LENGTH_
    SHORT).show();
             return true;
            }
           });
            popup.show();//showing popup menu
          }
         });//closing the setOnClickListener method
        }
      }
```

Output:



IMAGES

Android provides many views which we can use to define a user interface for our apps. Amongst these it provides a large number to display information and take input from the user, these include text and image views.

Android provides views which can be used to display images from various sources and provide transitions between them. Some of these views are the ImageView and the ImageSwitcher. These views provide a high level of functionality to display images in a user interface so that we can concentrate on the images we want to display rather than taking care of rendering.

Using the Image View to Display Images.

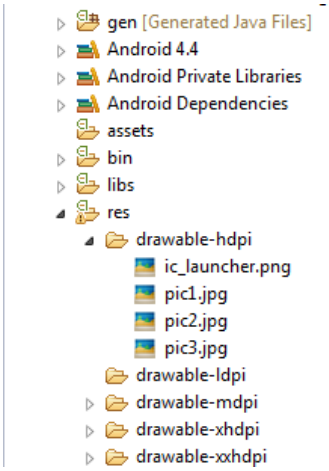
To render images Android provides us with theImageView class. Let's start by creating a program that will use an ImageView to display some images and a button which when clicked will change the image in the ImageView. You can find the code for this section on GitHub.

Create a basic Android project with an Activity that sets the main view from a layout file and does nothing else. Then open the layout file and add the ImageView and a button as shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="16dp"
  android:paddingRight="16dp"
  android:orientation="vertical">

  <ImageView
    android:id="@+id/imageDisplay"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

  <Button
    android:id="@+id/btnRotateImage"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Next Image" />
</LinearLayout>
```

In the code above we created a LinearLayout and added an ImageView to display an image and a button which will rotate the images in the imageView.

Add some images to the resource folder depending on the screen sizes you are planning to support as shown below:



Now update your Activity code as follows, using appropriate names for your project:

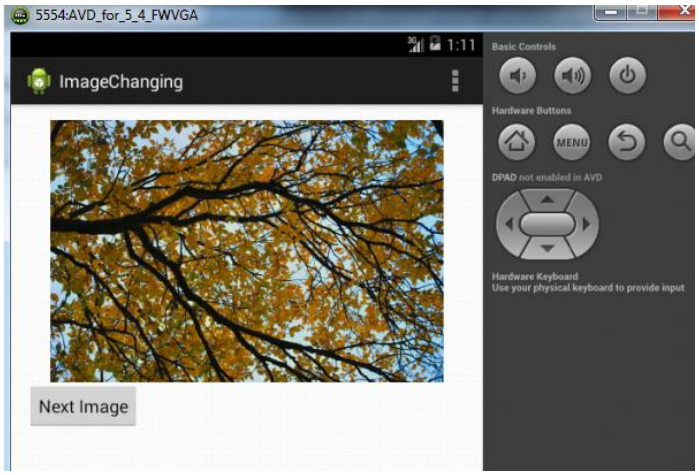
```
public class ImageChangingActivity extends Activity {
   private Integer images[] = {R.drawable.pic1, R.drawable.pic2, R.drawable.pic3};
   private int currImage = 0;

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

      setInitialImage();
      setImageRotateListener();
   }
   private void setImageRotateListener() {
      final Button rotatebutton = (Button) findViewById(R.id.btnRotateImage);
      rotatebutton.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View arg0) {
            currImage++;
            if (currImage == 3) {
               currImage = 0;
            }
            setCurrentImage();
         }
      });
   }
```

```
  private void setInitialImage() {
    setCurrentImage();
  }
  private void setCurrentImage() {
    final ImageView imageView = (ImageView) findViewById(R.id.imageDisplay);
    imageView.setImageResource(images[currImage]);

  }
}
```

Above we created an Array of the resource IDs for the images stored in our resources folder. In the `OnCreate` method we set the content view to the layout created. In the `setImageRotateListener` function we set up a listener to the `onClick` event of the button which changes the `currentImage` counter and sets the new image in the `ImageView`.

The `setCurrentImage` function gets the `ImageView` object using the `findViewById` function, then sets the resource id of the current image using the `setImageResource` function on the `ImageView` which will display the image in the image view.

If you run the program you should see the image in the image view and clicking it should change the image to the next one:



Using the Image Switcher View in Android.

In the above example we switched the image in the image view. This switching of images does not happen in a very smooth way and you might want to use a transition when the image changes. For this we use the `ImageSwitcher` View.

Find the code for this example on GitHub.

First add an image switcher view to the layout as follows:

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

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical">
    <ImageSwitcher
        android:id="@+id/imageSwitcher"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/btnRotateImage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Next Image" />
</LinearLayout>
```

Then add the following code to our Activity which will initialise the ImageSwitcher and setup the button to change the image with a transition.

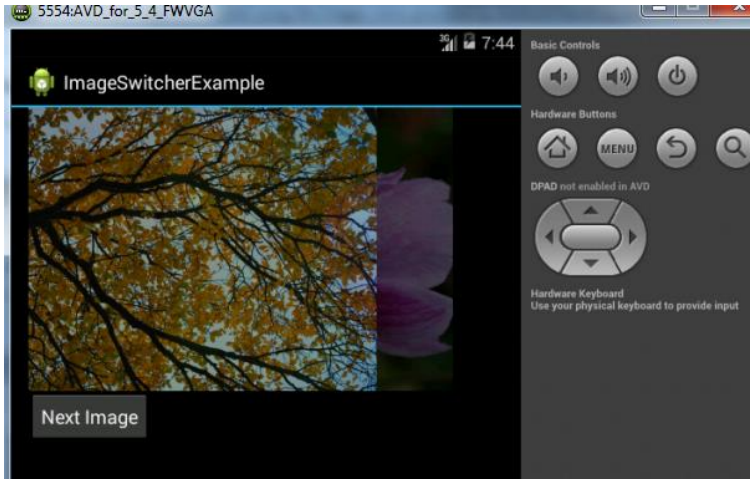
```
public class ImageSwitcherExampleActivity extends Activity {

    private Integer images[] = {R.drawable.pic1, R.drawable.pic2, R.drawable.pic3};
    private int currImage = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_image_switcher_example);
        initializeImageSwitcher();
        setInitialImage();
        setImageRotateListener();
    }

    private void initializeImageSwitcher() {
```

```
    final ImageSwitcher imageSwitcher = (ImageSwitcher)
findViewById(R.id.imageSwitcher);
    imageSwitcher.setFactory(new ViewFactory() {
      @Override
      public View makeView() {
        ImageView imageView = new ImageView(ImageSwitcherExampleActivity.this);
        return imageView;
      }
    });

    imageSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
android.R.anim.slide_in_left));
    imageSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
android.R.anim.slide_out_right));
  }
  private void setImageRotateListener() {
    final Button rotatebutton = (Button) findViewById(R.id.btnRotateImage);
    rotatebutton.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View arg0) {
        currImage++;
        if (currImage == 3) {
          currImage = 0;
        }
        setCurrentImage();
      }
    });
  }
  private void setInitialImage() {
    setCurrentImage();
  }
  private void setCurrentImage() {
    final ImageSwitcher imageSwitcher = (ImageSwitcher)
findViewById(R.id.imageSwitcher);
```

```
    imageSwitcher.setImageResource(images[currImage]);
  }
}
```

In the code above we get the ImageSwitcher object and then set the ViewFactory which creates a plainImageView. Then we set the animation to fade in and fade out. We update the setCurrentImagefunction to set the images in the ImageSwitcher.

Now the image will change with an animation.



Downloading and Setting Images on ImageView from Internet.

The code for this section can be found on GitHub.

You might not always have images available to you locally and may instead want to display them from the internet.

You should not undertake any network operations in the UI thread, instead the download should happen in a different background thread. We will do this in an Async Task. First set the permission to use the Internet in our AndroidManifest.xml as follows:

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

The layout of your project should have one ImageView and a Button as shown:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="16dp"
  android:paddingRight="16dp"
  android:orientation="vertical">

  <ImageView
    android:id="@+id/imageDisplay"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
  <Button
    android:id="@+id/btnRotateImage"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Next Image" />
</LinearLayout>
```

Then we create an AsyncTask which takes the ImageView and URL to download, downloads the image and sets the image in the ImageView.

The ImageDownloader async task downloads the image data and sets the ImageView. The complete activity code should now be as follows:

```
public class ImagedownloaderActivity extends Activity {

  private String imageUrls[] = {"http://www.abc.com/image1.jpg",
      "http://www.abc.com/image2.jpg",
      "http://www.abc.com/image3.jpg"
  };

  private class ImageDownloader extends AsyncTask<String, Void, Bitmap> {
    ImageView bmImage;
    public ImageDownloader(ImageView bmImage) {
      this.bmImage = bmImage;
    }
    protected Bitmap doInBackground(String... urls) {
      String url = urls[0];
      Bitmap bitmap = null;
      try {
        InputStream in = new java.net.URL(url).openStream();
        bitmap = BitmapFactory.decodeStream(in);
      } catch (Exception e) {
        Log.e("MyApp", e.getMessage());
      }
      return bitmap;
```

```
    }
    protected void onPostExecute(Bitmap result) {
        bmImage.setImageBitmap(result);
    }  }

  private int currImage = 0;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_imagedownloader);
    setInitialImage();
    setImageRotateListener();
  }
  private void setImageRotateListener() {
    final Button rotatebutton = (Button) findViewById(R.id.btnRotateImage);
    rotatebutton.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View arg0) {
        currImage++;
        if (currImage == 3) {
          currImage = 0;
        }
        setCurrentImage();
      }    });   }
  private void setInitialImage() {
    setCurrentImage();
  }
  private void setCurrentImage() {
    final ImageView imageView = (ImageView) findViewById(R.id.imageDisplay);
    ImageDownloader imageDownLoader = new ImageDownloader(imageView);
    imageDownLoader.execute(imageUrls[currImage]);
  }}
```

In the above code we stored the URLs of the images in the `imageUrls` array. In the `setCurrentImage`function we pass the `ImageView` to the `ImageDownloader` Async task and

pass the URL of the image to download and set in the ImageView. The ImageDownloader Async task will download the image and set it in the ImageView.

Implementing a Gallery Using the Horizontal Scroll View.

The code for this example can be found on GitHub.

In the above examples we saw how to display one image at a time using an ImageView. Sometimes we might want to display a variable number of images and let the user scroll through them. This we can achieve by putting a LinearLayout inside a horizontal scrollView and then dynamically addImageViews to that linear layout. For this we create a new activity called ImageGalleryActivityand update the layout file accordingly:

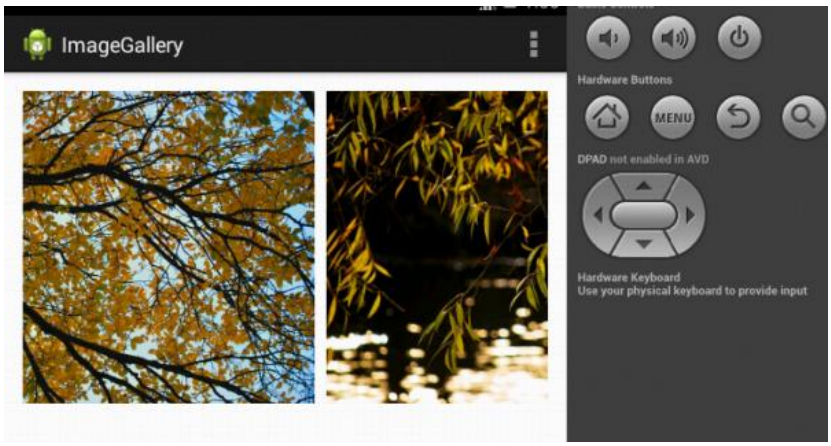
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="com.example.imagegallery.ImageGalleryActivity">
  <HorizontalScrollView
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <LinearLayout
      android:id="@+id/imageGallery"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:orientation="horizontal" />
  </HorizontalScrollView>
</RelativeLayout>
```

And code of the Activity:

```
public class ImageGalleryActivity extends Activity {
  private Integer images[] = {R.drawable.pic1, R.drawable.pic2, R.drawable.pic3};
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.activity_image_gallery);
    addImagesToThegallery();
  }
  private void addImagesToThegallery() {
    LinearLayout imageGallery = (LinearLayout) findViewById(R.id.imageGallery);
    for (Integer image : images) {
      imageGallery.addView(getImageView(image));
    } }
  private View getImageView(Integer image) {
    ImageView imageView = new ImageView(getApplicationContext());
    LinearLayout.LayoutParams lp = new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.WRAP_CONTENT,
LinearLayout.LayoutParams.WRAP_CONTENT);
    lp.setMargins(0, 0, 10, 0);
    imageView.setLayoutParams(lp);
    imageView.setImageResource(image);
    return imageView;
  }}
```

In the code above we dynamically create the ImageViews and add margins to them. The LinerLayouthas the orientation set to horizontal. Now if we run the program, we will be able to see the images in a Horizontally scrollable gallery as seen below.



Android AlertDialog Example

Android AlertDialog can be used to display the dialog message with OK and Cancel buttons. It can be used to interrupt and ask the user about his/her choice to continue or discontinue.

Android AlertDialog is composed of three regions: title, content area and action buttons.

Android AlertDialog is the subclass of Dialog class.

Android AlertDialog Example

Let's see a simple example of android alert dialog.

activity_main.xml

You can have multiple components, here we are having only a textview.

File: activity_main.xml

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity" >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:text="@string/hello_world" />
</RelativeLayout>
```

---

strings.xml

Optionally, you can store the dialog message and title in the strings.xml file.

File: strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">alertdialog</string>
  <string name="hello_world">Hello world!</string>
  <string name="menu_settings">Settings</string>
  <string name="dialog_message">Welcome to Alert Dialog</string>
    <string name="dialog_title">Javatpoint Alert Dialog</string>
</resources>
```

Activity class

Let's write the code to create and show the AlertDialog.

File: MainActivity.java

```
package com.example.alertdialog;
 import android.os.Bundle;
import android.app.Activity;
import android.app.AlertDialog;

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

    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    //Uncomment the below code to Set the message and title from the strings.xml file
    //builder.setMessage(R.string.dialog_message) .setTitle(R.string.dialog_title);
          //Setting message manually and performing action on button click
    builder.setMessage("Do you want to close this application ?")
      .setCancelable(false)
      .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
        finish();
        }
      })
      .setNegativeButton("No", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
        //  Action for 'NO' Button
        dialog.cancel();
       }
      });
     //Creating dialog box
    AlertDialog alert = builder.create();
    //Setting the title manually
    alert.setTitle("AlertDialogExample");
    alert.show();
    setContentView(R.layout.activity_main);
  }
```

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
}
```

Output:

Android AlarmManager

Android AlarmManager allows you to access system alarm.

By the help of **Android AlarmManager** in android, you can *schedule your application to run at a specific time* in the future. It works whether your phone is running or not.

The Android AlarmManager holds a CPU wake lock that *provides guarantee not to sleep the phone* until broadcast is handled.

Android AlarmManager Example

Let's see a simple AlarmManager example that runs after a specific time provided by user.

activity_main.xml

You need to drag only a edittext and a button as given below.

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/time"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="28dp"
        android:ems="10"
        android:hint="Number of seconds"
        android:inputType="numberDecimal" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignRight="@+id/time"
        android:layout_below="@+id/time"
        android:layout_marginRight="60dp"
        android:layout_marginTop="120dp"
        android:text="Start" />
</RelativeLayout>
```

Activity class

The activity class starts the alarm service when user clicks on the button.

File: MainActivity.java

```
package com.example.alarmexample;

import android.app.Activity;
import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Intent;
```

```
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

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.button1);

    b1.setOnClickListener(new OnClickListener() {

      @Override
      public void onClick(View v) {
        // TODO Auto-generated method stub
        startAlert();
      }
    });

}  public void startAlert() {
    EditText text = (EditText) findViewById(R.id.time);
    int i = Integer.parseInt(text.getText().toString());
    Intent intent = new Intent(this, MyBroadcastReceiver.class);
    PendingIntent pendingIntent = PendingIntent.getBroadcast(
                this.getApplicationContext(), 234324243, intent, 0);
    AlarmManager alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE
);
    alarmManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis()
```

```
                + (i * 1000), pendingIntent);
        Toast.makeText(this, "Alarm set in " + i + " seconds",Toast.LENGTH_LONG).show();
      }
    }
```

Let's create BroadcastReceiver class that starts alarm.

File: MyBroadcastReceiver.java

```
package com.example.alarmexample;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.media.MediaPlayer;
import android.widget.Toast;

public class MyBroadcastReceiver extends BroadcastReceiver {
    MediaPlayer mp;
    @Override
    public void onReceive(Context context, Intent intent) {
        mp=MediaPlayer.create(context, R.raw.alrm   );
        mp.start();
        Toast.makeText(context, "Alarm....", Toast.LENGTH_LONG).show();
    }
}
```

File: AndroidManifest.xml

You need to provide a receiver entry in AndroidManifest.xml file.

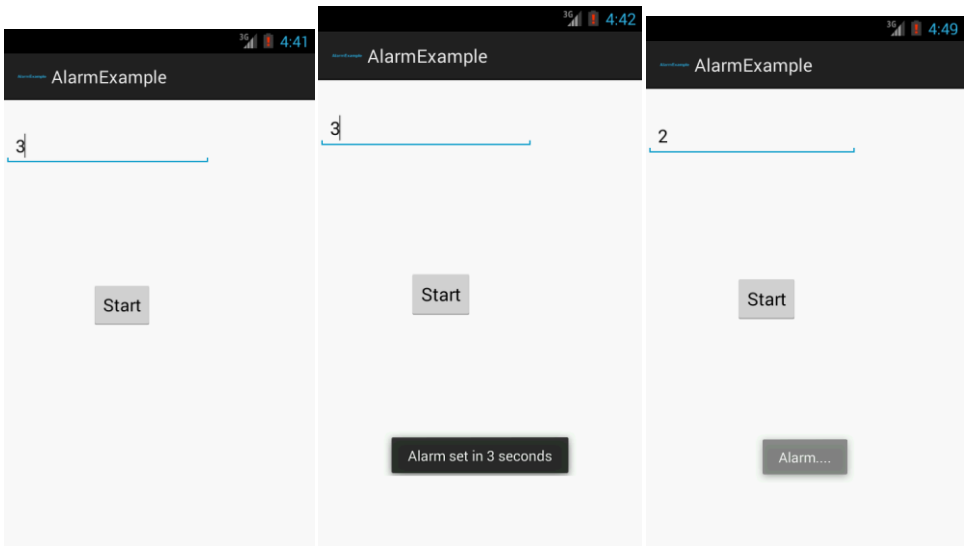
```
<receiver android:name="MyBroadcastReceiver" >
</receiver>
```

Let's see the full code of AndroidManifest.xml file.

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

```
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="16" />

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


    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.alarmexample.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>

        <receiver android:name="MyBroadcastReceiver" >
        </receiver>
    </application>
</manifest>
```

Output:



SMS Messaging

We can send sms in android via intent. You need to write only 4 lines of code the send sms in android.

```
//Getting intent and PendingIntent instance

Intent intent=new Intent(getApplicationContext(),MainActivity.class);

PendingIntent pi=PendingIntent.getActivity(getApplicationContext(), 0, intent,0);

    //Get the SmsManager instance and call the sendTextMessage method to send message

SmsManager sms=SmsManager.getDefault();

sms.sendTextMessage("8802177690", null, "hello javatpoint", pi,null);
```

Example of sending sms in android

activity_main.xml

Drag the 2 edittexts, 2 textviews and 1 button from the pallete, now the activity_main.xml file will like this:

File: activity_main.xml

```
1.  <RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
2.      xmlns:tools="http://schemas.android.com/tools"
3.      android:layout_width="match_parent"
4.      android:layout_height="match_parent"
5.      tools:context=".MainActivity" >
6.       <EditText
7.         android:id="@+id/editText1"
```

```
8.      android:layout_width="wrap_content"
9.      android:layout_height="wrap_content"
10.     android:layout_alignParentRight="true"
11.     android:layout_alignParentTop="true"
12.     android:layout_marginRight="20dp"
13.     android:ems="10" />
14.
15.  <EditText
16.     android:id="@+id/editText2"
17.     android:layout_width="wrap_content"
18.     android:layout_height="wrap_content"
19.     android:layout_alignLeft="@+id/editText1"
20.     android:layout_below="@+id/editText1"
21.     android:layout_marginTop="26dp"
22.     android:ems="10"
23.     android:inputType="textMultiLine" />
24.
25.  <TextView
26.     android:id="@+id/textView1"
27.     android:layout_width="wrap_content"
28.     android:layout_height="wrap_content"
29.     android:layout_alignBaseline="@+id/editText1"
30.     android:layout_alignBottom="@+id/editText1"
31.     android:layout_toLeftOf="@+id/editText1"
32.     android:text="Mobile No:" />
33.
34.  <TextView
35.     android:id="@+id/textView2"
36.     android:layout_width="wrap_content"
37.     android:layout_height="wrap_content"
38.     android:layout_alignBaseline="@+id/editText2"
39.     android:layout_alignBottom="@+id/editText2"
40.     android:layout_alignLeft="@+id/textView1"
41.     android:text="Message:" />
42.
43.  <Button
44.     android:id="@+id/button1"
45.     android:layout_width="wrap_content"
46.     android:layout_height="wrap_content"
47.     android:layout_alignLeft="@+id/editText2"
48.     android:layout_below="@+id/editText2"
49.     android:layout_marginLeft="34dp"
50.     android:layout_marginTop="48dp"
51.     android:text="Send SMS" />
52.
53. </RelativeLayout>
```

Write the permission code in Android-Manifest.xml file

You need to write SEND_SMS permission as given below:

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

File: Android-Manifest.xml

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

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="16" />
     <uses-permission android:name="android.permission.SEND_SMS"/>
      <uses-permission android:name="android.permission.RECEIVE_SMS"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.sendsms.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

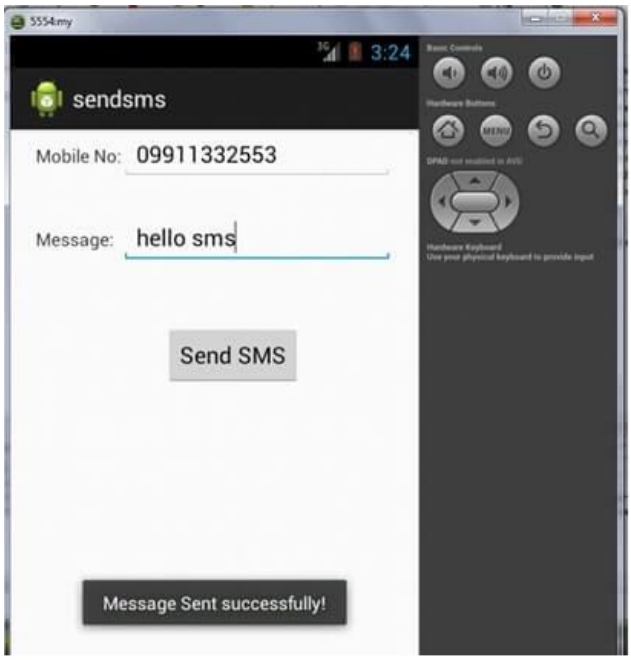
Let's write the code to make the phone call via intent.

File: MainActivity.java

```
package com.example.sendsms;

import android.os.Bundle;
import android.app.Activity;
import android.app.PendingIntent;
import android.content.Intent;
import android.telephony.SmsManager;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
```

```
13. import android.widget.Toast;
14.
15. public class MainActivity extends Activity {
16.      EditText mobileno,message;
17.    Button sendsms;
18.    @Override
19.    protected void onCreate(Bundle savedInstanceState) {
20.       super.onCreate(savedInstanceState);
21.       setContentView(R.layout.activity_main);
22.
23.       mobileno=(EditText)findViewById(R.id.editText1);
24.       message=(EditText)findViewById(R.id.editText2);
25.       sendsms=(Button)findViewById(R.id.button1);
26.
27.    //Performing action on button click
28.       sendsms.setOnClickListener(new OnClickListener() {
29.
30.          @Override
31.          public void onClick(View arg0) {
32.             String no=mobileno.getText().toString();
33.             String msg=message.getText().toString();
34.
35.             //Getting intent and PendingIntent instance
36.             Intent intent=new Intent(getApplicationContext(),MainActivity.class);
37.             PendingIntent pi=PendingIntent.getActivity(getApplicationContext(), 0, intent,0);
38.
39.             //Get the SmsManager instance and call the sendTextMessage method to send message
40.             SmsManager sms=SmsManager.getDefault();
41.             sms.sendTextMessage(no, null, msg, pi,null);
42.
43.             Toast.makeText(getApplicationContext(), "Message Sent successfully!",
44.                Toast.LENGTH_LONG).show();
45.          }
46.       });
47.    }
48.
49.    @Override
50.    public boolean onCreateOptionsMenu(Menu menu) {
51.       // Inflate the menu; this adds items to the action bar if it is present.
52.       getMenuInflater().inflate(R.menu.activity_main, menu);
53.       return true;
54.    }
55.
56. }
```

Install and Run the apk file on the Real Device (e.g. Mobile) to send the sms.



SENDING MAIL

How to send email in android using intent

We can easily send email in android via intent. You need to write few lines of code only as given below

1. Intent email = new Intent(Intent.ACTION_SEND);
2. email.putExtra(Intent.EXTRA_EMAIL, new String[]{ to});

```
3. email.putExtra(Intent.EXTRA_SUBJECT, subject);
4. email.putExtra(Intent.EXTRA_TEXT, message);
5. //need this to prompts email client only
6. email.setType("message/rfc822");
7.  startActivity(Intent.createChooser(email, "Choose an Email client :"));
```

activity_main.xml

Drag the 2 EditTexts, 1 MultiLine EditText, 3 TextViews and 1 Button from the pallete, now the activity_main.xml file will like this:

File: activity_main.xml

```
1. <RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
2.    xmlns:tools="http://schemas.android.com/tools"
3.    android:layout_width="match_parent"
4.    android:layout_height="match_parent"
5.    tools:context=".MainActivity" >
6.
7.    <EditText
8.       android:id="@+id/editText1"
9.       android:layout_width="wrap_content"
10.      android:layout_height="wrap_content"
11.      android:layout_alignParentRight="true"
12.      android:layout_alignParentTop="true"
13.      android:layout_marginRight="22dp"
14.      android:layout_marginTop="16dp"
15.      android:ems="10" />
16.
17.   <EditText
18.      android:id="@+id/editText2"
19.      android:layout_width="wrap_content"
20.      android:layout_height="wrap_content"
21.      android:layout_alignLeft="@+id/editText1"
22.      android:layout_below="@+id/editText1"
23.      android:layout_marginTop="18dp"
24.      android:ems="10" >
25.
26.      <requestFocus />
27.   </EditText>
28.
29.   <EditText
30.      android:id="@+id/editText3"
31.      android:layout_width="wrap_content"
32.      android:layout_height="wrap_content"
33.      android:layout_alignLeft="@+id/editText2"
34.      android:layout_below="@+id/editText2"
35.      android:layout_marginTop="28dp"
36.      android:ems="10"
37.      android:inputType="textMultiLine" />
38.
39.   <TextView
40.      android:id="@+id/textView1"
```

```
41.       android:layout_width="wrap_content"
42.       android:layout_height="wrap_content"
43.       android:layout_alignBaseline="@+id/editText1"
44.       android:layout_alignBottom="@+id/editText1"
45.       android:layout_alignParentLeft="true"
46.       android:text="To:" />
47.
48.    <TextView
49.       android:id="@+id/textView2"
50.       android:layout_width="wrap_content"
51.       android:layout_height="wrap_content"
52.       android:layout_alignBaseline="@+id/editText2"
53.       android:layout_alignBottom="@+id/editText2"
54.       android:layout_alignParentLeft="true"
55.       android:text="Subject:" />
56.
57.    <TextView
58.       android:id="@+id/textView3"
59.       android:layout_width="wrap_content"
60.       android:layout_height="wrap_content"
61.       android:layout_alignBaseline="@+id/editText3"
62.       android:layout_alignBottom="@+id/editText3"
63.       android:layout_alignParentLeft="true"
64.       android:text="Message:" />
65.
66.    <Button
67.       android:id="@+id/button1"
68.       android:layout_width="wrap_content"
69.       android:layout_height="wrap_content"
70.       android:layout_alignLeft="@+id/editText3"
71.       android:layout_below="@+id/editText3"
72.       android:layout_marginLeft="76dp"
73.       android:layout_marginTop="20dp"
74.       android:text="Send" />
75.
76. </RelativeLayout>
```

Activity class

Let's write the code to send email via intent.

File: MainActivity.java

```
1.  package com.example.sendemail;
2.    import android.os.Bundle;
3.  import android.app.Activity;
4.  import android.content.Intent;
5.  import android.view.Menu;
6.  import android.view.View;
7.  import android.view.View.OnClickListener;
8.  import android.widget.Button;
```

```
9.  import android.widget.EditText;
10.
11. public class MainActivity extends Activity {
12.     EditText editTextTo,editTextSubject,editTextMessage;
13.     Button send;
14.     @Override
15.     protected void onCreate(Bundle savedInstanceState) {
16.         super.onCreate(savedInstanceState);
17.         setContentView(R.layout.activity_main);
18.         editTextTo=(EditText)findViewById(R.id.editText1);
19.         editTextSubject=(EditText)findViewById(R.id.editText2);
20.         editTextMessage=(EditText)findViewById(R.id.editText3);
21.         send=(Button)findViewById(R.id.button1);
22.               send.setOnClickListener(new OnClickListener(){
23.              @Override
24.             public void onClick(View arg0) {
25.                 String to=editTextTo.getText().toString();
26.                 String subject=editTextSubject.getText().toString();
27.                 String message=editTextMessage.getText().toString();
28.
29.                  Intent email = new Intent(Intent.ACTION_SEND);
30.                  email.putExtra(Intent.EXTRA_EMAIL, new String[]{ to});
31.                  email.putExtra(Intent.EXTRA_SUBJECT, subject);
32.                  email.putExtra(Intent.EXTRA_TEXT, message);
33.
34.                  //need this to prompts email client only
35.                  email.setType("message/rfc822");
36.                       startActivity(Intent.createChooser(email, "Choose an Email client :"));
37.                 }
38.               });
39.     }
40.
41.     @Override
42.     public boolean onCreateOptionsMenu(Menu menu) {
43.         // Inflate the menu; this adds items to the action bar if it is present.
44.         getMenuInflater().inflate(R.menu.activity_main, menu);
45.         return true;
46.     }
47.
48. }
```

Android - 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
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;. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
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.app.Activity;
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);
    }
  };

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
  }

  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
```

```
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement

    if (id == R.id.action_settings) {
      return true;
    }
    return super.onOptionsItemSelected(item);
  }
}
```

Following is the modified content of the xml **res/layout/activity_main.xml**.

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

  <TextView android: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=">>"
    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="||"
    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="<"
    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="<<"
    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="hello_world">Hello world!</string>
  <string name="action_settings">Settings</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.mediaplayer"
  android:versionCode="1"
  android:versionName="1.0" >
```

```
    <uses-sdk
        android:minSdkVersion="13"
        android:targetSdkVersion="22" />

    <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>
```

Before starting your application, Android studio will display following screens

By default you would see the pause button disabled. Now press play button and it would become disable and pause button become enable. It is shown in the picture below –

Up till now, the music has been playing. Now press the pause button and see the pause notification. This is shown below –

Now when you press the play button again, the song will not play from the beginning but from where it was paused. Now press the fast forward or backward button to jump the song forward or backward 5 seconds. A time came when the song cannot be jump forward. At this point , the notification would appear which would be something like this –

Your music would remain playing in the background while you are doing other tasks in your mobile. In order to stop it , you have to exit this application from background activities.

Above image shows when you pick rewind button.

Android - 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. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to add intent code to launch the activity and result method to recieve the output.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required. Here we add only imageView and a textView.
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.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;


import android.content.ClipData;
```

```
import android.content.ClipboardManager;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;

import android.graphics.Bitmap;
import android.os.BatteryManager;
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.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.Set;

public class MainActivity extends ActionBarActivity {
  Button b1,b2;
  ImageView iv;

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

```
    b1=(Button)findViewById(R.id.button);
    iv=(ImageView)findViewById(R.id.imageView);

    b1.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Intent intent = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
        startActivityForResult(intent, 0);
      }
    });
  }

  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // TODO Auto-generated method stub
    super.onActivityResult(requestCode, resultCode, data);

    Bitmap bp = (Bitmap) data.getExtras().get("data");
    iv.setImageBitmap(bp);
  }

  @Override
  protected void onDestroy() {
    super.onDestroy();
  }
    @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
  }
    @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
```

```
    // as you specify a parent activity in AndroidManifest.xml.

    int id = item.getItemId();
        //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
      return true;
    }
    return super.onOptionsItemSelected(item);
  }
}
```

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: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="Camera 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:layout_below="@+id/textView"
    android:layout_centerHorizontal="true" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="camera"
    android:id="@+id/button"
    android:layout_below="@+id/imageView"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="86dp" />
        </RelativeLayout>
```

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

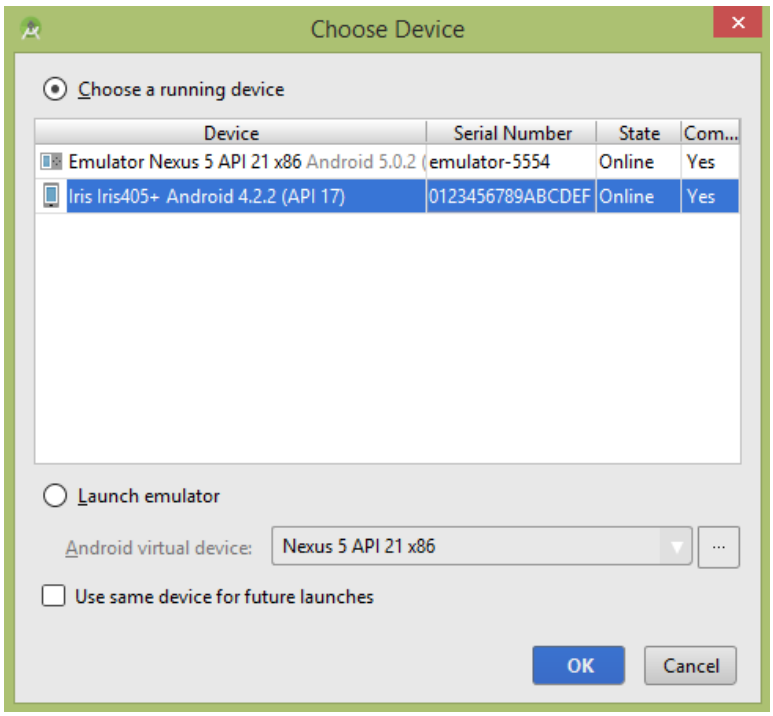
```
<resources>
  <string name="app_name">My Application</string>
  <string name="hello_world">Hello world!</string>
  <string name="action_settings">Settings</string>
</resources>
```

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

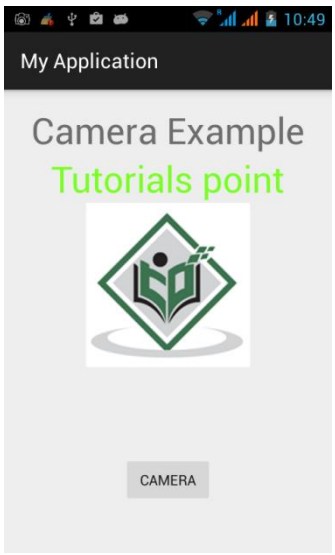
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.camera"
  android:versionCode="1"
  android:versionName="1.0" >

  <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. Before starting your application, Android studio will display following window to select an option where you want to run your Android application.

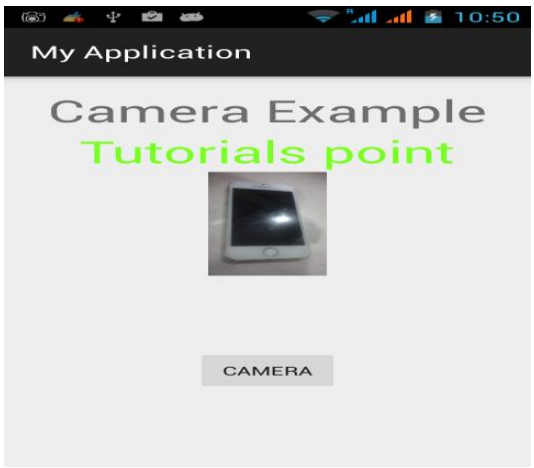


Select your mobile device as an option and then check your mobile device which will display following screen –



Now just tap on the button on and the camera will be opened. Just capture a picture. After capturing it , two buttons will appear asking you to discard it or keep it

Just press the tic button and you will be brought back to your application with the captured image in place of android icon



Directly using Camera API provided by android in our application

We will be using the camera API to integrate the camera in our application

First you will need to initialize the camera object using the static method provide by the api called *Camera.open*. Its syntax is

```
Camera object = null;
object = Camera.open();
```

Apart from the above function , there are other functions provided by the Camera class that which are listed below

Sr.No	Method & Description
1	**getCameraInfo(int cameraId, Camera.CameraInfo cameraInfo)** It returns the information about a particular camera
2	**getNumberOfCameras()** It returns an integer number defining of cameras available on device
3	**lock()** It is used to lock the camera , so no other application can access it
4	**release()** It is used to release the lock on camera , so other applications can access it
5	**open(int cameraId)** It is used to open particular camera when multiple cameras are supported
6	**enableShutterSound(boolean enabled)** It is used to enable/disable default shutter sound of image capture

Now you need make an separate class and extend it with SurfaceView and implements SurfaceHolder interface.

The two classes that have been used have the following purpose

Class	Description
Camera	It is used to control the camera and take images or capture video from the camera
SurfaceView	This class is used to present a live camera preview to the user.

You have to call the preview method of the camera class to start the preview of the camera to the user

```
public class ShowCamera extends SurfaceView implements SurfaceHolder.Callback {
  private Camera theCamera;

  public void surfaceCreated(SurfaceHolder holder) {
    theCamera.setPreviewDisplay(holder);
```

```
    theCamera.startPreview();
  }

  public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3){
  }
    public void surfaceDestroyed(SurfaceHolder arg0) {
  }
}
```

Apart from the preview there are other options of the camera that can be set using the other functions provided by the Camera API

Sr.No	Method & Description
1	**startFaceDetection()** This function starts the face detection in the camera
2	**stopFaceDetection()** It is used to stop the face detection which is enabled by the above function
3	**startSmoothZoom(int value)** It takes an integer value and zoom the camera very smoothly to that value
4	**stopSmoothZoom()** It is used to stop the zoom of the camera
5	**stopPreview()** It is used to stop the preview of the camera to the user
6	**takePicture(Camera.ShutterCallback shutter, Camera.PictureCallback raw, Camera.PictureCallback jpeg)** It is used to enable/disable default shutter sound of image capture

Example

Following example demonstrates the usage of the camera API in the application

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, because camera is not supported by the emulator

Steps	Description
1	You will use Android studio IDE to create an Android application and name it as Camera under a package com.example.sairamkrishna.myapplication;. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.

2	Modify src/MainActivity.java file to add the respective code of camera.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required. Here we add only FrameView and a button and a ImageView.
4	Modify AndroidManifest.xml as shown below to add the necessary permissions for camera
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.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;

import android.content.ClipData;
import android.content.ClipboardManager;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;

import android.graphics.Bitmap;
import android.hardware.Camera;
import android.hardware.Camera.PictureCallback;
import android.hardware.Camera.ShutterCallback;

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

import android.view.Menu;
import android.view.MenuItem;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
```

```
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.TextView;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import java.util.ArrayList;
import java.util.Set;

public class MainActivity extends Activity implements SurfaceHolder.Callback {
  Camera camera;
  SurfaceView surfaceView;
  SurfaceHolder surfaceHolder;

  Camera.PictureCallback rawCallback;
  Camera.ShutterCallback shutterCallback;
  Camera.PictureCallback jpegCallback;

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

    surfaceView = (SurfaceView) findViewById(R.id.surfaceView);
    surfaceHolder = surfaceView.getHolder();

    surfaceHolder.addCallback(this);
    surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
```

```
    jpegCallback = new PictureCallback() {

      @Override
      public void onPictureTaken(byte[] data, Camera camera) {
        FileOutputStream outStream = null;
          try {
            outStream = new FileOutputStream(String.format("/sdcard/%d.jpg",
System.currentTimeMillis()));

            outStream.write(data);
            outStream.close();
          }

          catch (FileNotFoundException e) {
            e.printStackTrace();
          }

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

          finally {
          }

          Toast.makeText(getApplicationContext(), "Picture Saved",
Toast.LENGTH_LONG).show();
          refreshCamera();
        }
      };
    }

    public void captureImage(View v) throws IOException {
      camera.takePicture(null, null, jpegCallback);
```

```
  }

  public void refreshCamera() {
    if (surfaceHolder.getSurface() == null) {
      return;
    }

    try {
      camera.stopPreview();
    }

    catch (Exception e) {
    }
          try {
      camera.setPreviewDisplay(surfaceHolder);
      camera.startPreview();
    }
    catch (Exception e) {
  }
}
  @Override
protected void onDestroy() {
  super.onDestroy();
}
  @Override
public boolean onCreateOptionsMenu(Menu menu) {
  // Inflate the menu; this adds items to the action bar if it is present.
  getMenuInflater().inflate(R.menu.menu_main, menu);
  return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
  // Handle action bar item clicks here. The action bar
```

```
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.

    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
      return true;
    }
    return super.onOptionsItemSelected(item);
  }

  @Override
  public void surfaceCreated(SurfaceHolder holder) {
    try {
      camera = Camera.open();
    }
     catch (RuntimeException e) {
      System.err.println(e);
      return;
    }

    Camera.Parameters param;
    param = camera.getParameters();
    param.setPreviewSize(352, 288);
    camera.setParameters(param);

    try {
      camera.setPreviewDisplay(surfaceHolder);
      camera.startPreview();
    }
    catch (Exception e) {
      System.err.println(e);
      return;
```

```
    }
  }

  @Override
  public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
    refreshCamera();
  }

  @Override
  public void surfaceDestroyed(SurfaceHolder holder) {
    camera.stopPreview();
    camera.release();
    camera = null;
  }
}
```

Modify the content of the **res/layout/activity_main.xml**

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

  <TextView android:text="Camera 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" />

  <SurfaceView
    android:id="@+id/surfaceView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"/>

</RelativeLayout>
```

Modify the content of the **res/values/string.xml**

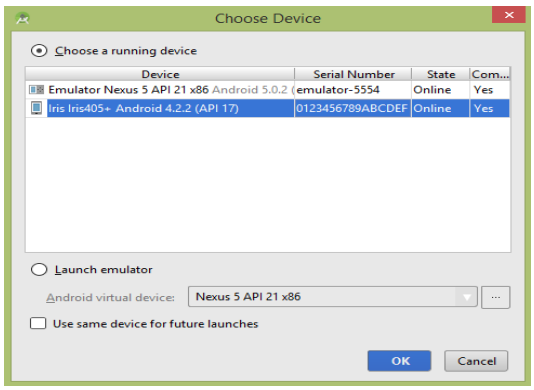
```
<resources>
  <string name="app_name">My Application</string>
  <string name="hello_world">Hello world!</string>
  <string name="action_settings">Settings</string>
</resources>
```

Modify the content of the **AndroidManifest.xml** and add the necessary permissions as shown below.

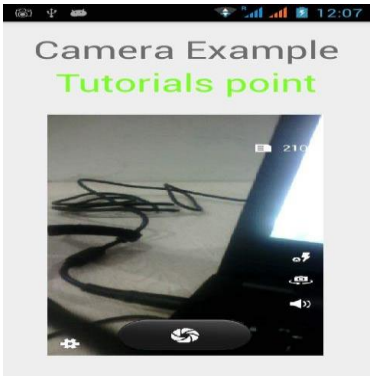
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.camera1"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-permission android:name="android.permission.CAMERA" />
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  <uses-feature android:name="android.hardware.camera" />
  <uses-feature android:name="android.hardware.camera.autofocus" />

  <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 window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen:



Above image shows the camera surface view on layout.

Understanding basic classes of Camera Intent and API

There are mainly four classes that we are going to discuss.

Intent

By the help of 2 constants of **MediaStore** class, we can capture picture and video without using the instance of Camera class.

1. ACTION_IMAGE_CAPTURE
2. ACTION_VIDEO_CAPTURE

Camera

It is main class of camera api, that can be used to take picture and video.

SurfaceView

It represents a surface view ore preview of live camera.

MediaRecorder

It is used to record video using camera. It can also be used to record audio files as we have seen in the previous example of media framework.

Android camera app example by camera intent

In this example, we are writing the simple code to capture image using camera and displaying the image using imageview.

activity_main.xml

Drag one imageview and one button from the pallete, now the xml file will look like this:

File: activity_main.xml

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

   <Button
      android:id="@+id/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentBottom="true"
      android:layout_centerHorizontal="true"
      android:text="Take a Photo" >
   </Button>

   <ImageView
      android:id="@+id/imageView1"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:layout_above="@+id/button1"
      android:layout_alignParentTop="true"
      android:src="@drawable/ic_launcher" >
   </ImageView>
</RelativeLayout>
```

Activity class

Let's write the code to capture image using camera and displaying it on the image view.

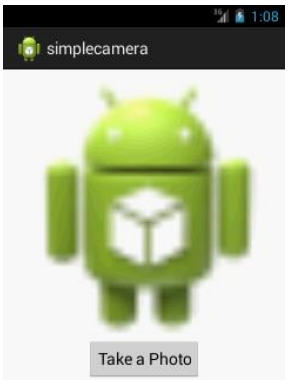
File: MainActivity.java

```
package com.example.simplecamera;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
```

```
import android.widget.Button;
import android.widget.ImageView;

public class MainActivity extends Activity {
    private static final int CAMERA_REQUEST = 1888;
    ImageView imageView;
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        imageView = (ImageView) this.findViewById(R.id.imageView1);
        Button photoButton = (Button) this.findViewById(R.id.button1);

        photoButton.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            Intent cameraIntent = new Intent(android.provider.MediaStore.ACTION_IMAGE_C
APTURE);
            startActivityForResult(cameraIntent, CAMERA_REQUEST);
        }
       });
      }

    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
     if (requestCode == CAMERA_REQUEST) {
      Bitmap photo = (Bitmap) data.getExtras().get("data");
      imageView.setImageBitmap(photo);
     }
   }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
```

```
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
}
```

Output:



Android TelephonyManager

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

Android TelephonyManager Example

Let's see the simple example of TelephonyManager that prints information of the telephony services.

activity_main.xml

Drag one textview from the pallete, now the xml file will look like this.

File: activity_main.xml

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

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

Activity class

Now, write the code to display the information about the telephony services.

File: MainActivity.java

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

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

      textView1=(TextView)findViewById(R.id.textView1);

      //Get the instance of TelephonyManager
      TelephonyManager  tm=(TelephonyManager)getSystemService(Context.TELEPHONY_
SERVICE);
```

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

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

int phoneType=tm.getPhoneType();

switch (phoneType)
{
      case (TelephonyManager.PHONE_TYPE_CDMA):
            strphoneType="CDMA";
              break;
      case (TelephonyManager.PHONE_TYPE_GSM):
            strphoneType="GSM";
              break;
      case (TelephonyManager.PHONE_TYPE_NONE):
             strphoneType="NONE";
               break;
 }
//getting information if phone is in roaming
boolean isRoaming=tm.isNetworkRoaming();

String info="Phone Details:\n";
info+="\n IMEI Number:"+IMEINumber;
info+="\n SubscriberID:"+subscriberID;
info+="\n Sim Serial Number:"+SIMSerialNumber;
```

```
        info+="\n Network Country ISO:"+networkCountryISO;
        info+="\n SIM Country ISO:"+SIMCountryISO;
        info+="\n Software Version:"+softwareVersion;
        info+="\n Voice Mail Number:"+voiceMailNumber;
        info+="\n Phone Network Type:"+strphoneType;
        info+="\n In Roaming? :"+isRoaming;
        textView1.setText(info);//displaying the information in the textView
    }
}
```

AndroidManifest.xml

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

File: AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:androclass="http://schemas.android.com/apk/res/android"
    package="com.javatpoint.telephonymanager"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

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

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.javatpoint.telephonymanager.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>
```

Output:

Android Call State Example

We can also get the information of call state using the **TelephonyManager** class. For this purpose, we need to call the listen method of TelephonyManager class by passing the PhonStateListener instance.

The **PhoneStateListener** interface must be implemented to get the call state. It provides one method onCallStateChanged().

Android Call State Example

Let's see the example, where we are determining whether phone is ringing or phone is in a call or phone is neither ringing nor in a call.

activity_main.xml

In this example, we don't have any component in this file..

Activity class

Let's write the code to know the call state.

File: MainActivity.java

```
package com.javatpoint.callstates;
```

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

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

    TelephonyManager telephonyManager =
            (TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);

    PhoneStateListener callStateListener = new PhoneStateListener() {
    public void onCallStateChanged(int state, String incomingNumber)
    {
        if(state==TelephonyManager.CALL_STATE_RINGING){
             Toast.makeText(getApplicationContext(),"Phone Is Riging",
                                        Toast.LENGTH_LONG).show();
        }
         if(state==TelephonyManager.CALL_STATE_OFFHOOK){
            Toast.makeText(getApplicationContext(),"Phone is Currently in A call",
                                            Toast.LENGTH_LONG).show();
         }

         if(state==TelephonyManager.CALL_STATE_IDLE){
            Toast.makeText(getApplicationContext(),"phone is neither ringing nor in a call"
, Toast.LENGTH_LONG).show();
         }
    }
```

```
        };
        telephonyManager.listen(callStateListener,PhoneStateListener.LISTEN_CALL_STATE);
    }

    @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;
    }
}
```

AndroidManifest.xml

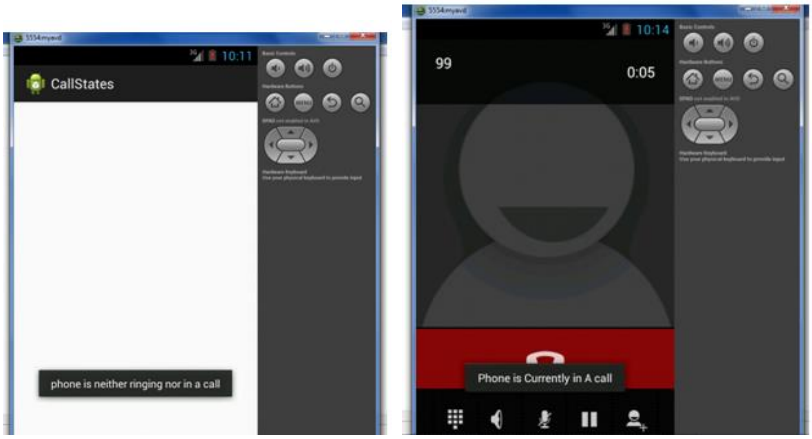
You need to provide **READ_PHONE_STATE** permission in the AndroidManifest.xml file.

File: AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:androclass="http://schemas.android.com/apk/res/android"
    package="com.javatpoint.callstates"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />


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

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

```
        android:name="com.javatpoint.callstates.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>
```

Output:



How to make a phone call in android

We are able to make a phone call in android via intent. You need to write only three lines of code to make a phone call.

```
Intent callIntent = new Intent(Intent.ACTION_CALL);
callIntent.setData(Uri.parse("tel:"+8802177690));//change the number
startActivity(callIntent);
```

Example of phone call in android

activity_main.xml

Drag the EditText and Button from the pallete, now the activity_main.xml file will like this:

File: activity_main.xml

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    tools:context=".MainActivity" >
     <Button
      android:id="@+id/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="118dp"
      android:text="Call" />

    <EditText
      android:id="@+id/editText1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="25dp"
      android:ems="10" />
</RelativeLayout>
```

Write the permission code in Android-Manifest.xml file

You need to write CALL_PHONE permission as given below:

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

File: Android-Manifest.xml

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

<manifest xmlns:androclass="http://schemas.android.com/apk/res/android"
  package="com.example.phonecall"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="8"
```

```
    android:targetSdkVersion="16" />

    <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.phonecall.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 make the phone call via intent.

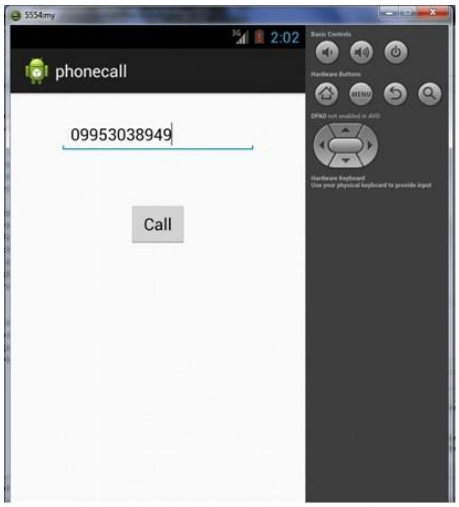
File: MainActivity.java

```
package com.example.phonecall;
 import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
```

```
public class MainActivity extends Activity {
    EditText edittext1;
    Button button1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Getting the edittext and button instance
        edittext1=(EditText)findViewById(R.id.editText1);
        button1=(Button)findViewById(R.id.button1);

        //Performing action on button click
        button1.setOnClickListener(new OnClickListener(){

            @Override
            public void onClick(View arg0) {
                String number=edittext1.getText().toString();
                Intent callIntent = new Intent(Intent.ACTION_CALL);
                callIntent.setData(Uri.parse("tel:"+number));
                startActivity(callIntent);
            }

        });
    }

    @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;
    }
}
```

Install and Run the apk file on the Real Device (e.g. Mobile) to make the phone call.

Output:



UNIT - V **(12 Lectures)**

Storing the data persistently-Introducing the Data Storage Options: The preferences, The Internal Storage, The External Storage, The Content Provider , The SQLite database, Connecting with the SQLite database and operations-Insert, Delete, Update, Fetch, Publishing android applications-preparing for publishing, Deploying APK files

Android Preferences Example

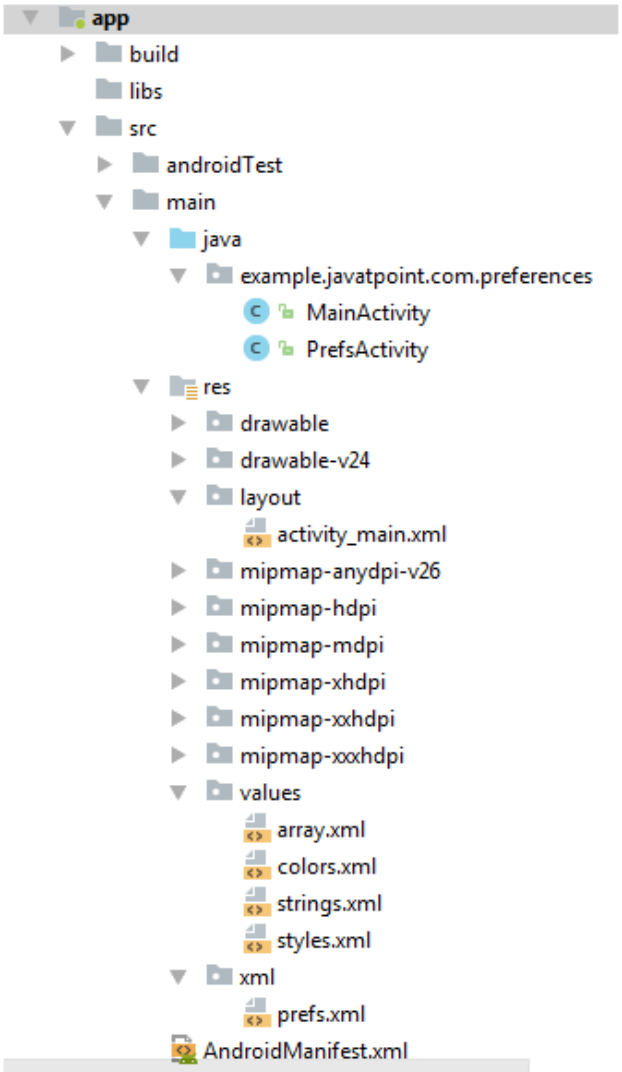
Android shared preference is used to store and retrieve primitive information. In android, string, integer, long, number etc. are considered as primitive data type.

Android Shared preferences are used to store data in key and value pair so that we can retrieve the value on the basis of key.

It is widely used to get information from user such as in settings.

Android Preferences Example

Let's see a simple example of android shared preference.



activity_main.xml

Drag one textview and two buttons from the pallete.

File: activity_main.xml

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

    <TextView
        android:id="@+id/txtPrefs"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:text="Data:" />

    <Button
        android:id="@+id/storeinformation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/showinformation"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="18dp"
        android:text="Store Information" />

    <Button
        android:id="@+id/showinformation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="17dp"
```

```
32.       android:text="Show Information" />
33.
34. </RelativeLayout>
```

array.xml

It is created inside res/values directory.

File: array.xml

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.     <string-array name="listOptions">
4.        <item>English</item>
5.        <item>Hindi</item>
6.        <item>Other</item>
7.     </string-array>
8.
9.     <string-array name="listValues">
10.       <item>English Language</item>
11.       <item>Hindi Language</item>
12.       <item>Other Language</item>
13.    </string-array>
14. </resources>
```

prefs.xml

It is created inside res/xml directory.

File: prefs.xml

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
3.     <PreferenceCategory
4.        android:summary="Username and password information"
5.        android:title="Login information" >
6.        <EditTextPreference
7.           android:key="username"
8.           android:summary="Please enter your login username"
9.           android:title="Username" />
10.       <EditTextPreference
11.          android:key="password"
```

```
12.         android:summary="Enter your password"
13.         android:title="Password" />
14.    </PreferenceCategory>
15.
16.    <PreferenceCategory
17.      android:summary="Username and password information"
18.      android:title="Settings" >
19.      <CheckBoxPreference
20.         android:key="checkBox"
21.         android:summary="On/Off"
22.         android:title="Keep me logged in" />
23.
24.      <ListPreference
25.         android:entries="@array/listOptions"
26.         android:entryValues="@array/listValues"
27.         android:key="listpref"
28.         android:summary="List preference example"
29.         android:title="List preference" />
30.    </PreferenceCategory>
31. </PreferenceScreen>
```

Main Activity Class

File: MainActivity.java

```
1.  package example.javatpoint.com.preferences;
2.
3.  import android.content.Intent;
4.  import android.content.SharedPreferences;
5.  import android.preference.PreferenceManager;
6.  import android.support.v7.app.AppCompatActivity;
7.  import android.os.Bundle;
8.  import android.view.View;
9.  import android.widget.Button;
10. import android.widget.TextView;
11.
```

```
12. public class MainActivity extends AppCompatActivity {
13.     TextView textView;
14.     @Override
15.     protected void onCreate(Bundle savedInstanceState) {
16.         super.onCreate(savedInstanceState);
17.         setContentView(R.layout.activity_main);
18.
19.         Button storeinformation = (Button) findViewById(R.id.storeinformation);
20.         Button showinformation = (Button) findViewById(R.id.showinformation);
21.         textView = (TextView) findViewById(R.id.txtPrefs);
22.
23.         View.OnClickListener listener = new View.OnClickListener() {
24.             @Override
25.             public void onClick(View v) {
26.                 switch (v.getId()) {
27.                     case R.id.storeinformation:
28.                         Intent intent = new Intent(MainActivity.this,PrefsActivity.class);
29.                         startActivity(intent);
30.                         break;
31.                     case R.id.showinformation:
32.                         displaySharedPreferences();
33.                         break;
34.                     default:
35.                         break;
36.                 }
37.             }
38.         };
39.         storeinformation.setOnClickListener(listener);
40.         showinformation.setOnClickListener(listener);
41.     }
42.
43.
44.     private void displaySharedPreferences() {
45.         SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(MainActi
```

```
    vity.this);
46.       String username = prefs.getString("username", "Default NickName");
47.       String passw = prefs.getString("password", "Default Password");
48.       boolean checkBox = prefs.getBoolean("checkBox", false);
49.       String listPrefs = prefs.getString("listpref", "Default list prefs");
50.
51.
52.       StringBuilder builder = new StringBuilder();
53.       builder.append("Username: " + username + "\n");
54.       builder.append("Password: " + passw + "\n");
55.       builder.append("Keep me logged in: " + String.valueOf(checkBox) + "\n");
56.       builder.append("List preference: " + listPrefs);
57.       textView.setText(builder.toString());
58.
59.    }
60.
61. }
```

PrefsActivity class

File: PrefsActivity.java

```
1.  package example.javatpoint.com.preferences;
2.
3.  import android.os.Bundle;
4.  import android.preference.PreferenceActivity;
5.
6.  public class PrefsActivity extends PreferenceActivity {
7.     @Override
8.     protected void onCreate(Bundle savedInstanceState) {
9.        super.onCreate(savedInstanceState);
10.       addPreferencesFromResource(R.xml.prefs);
11.    }
12. }
```

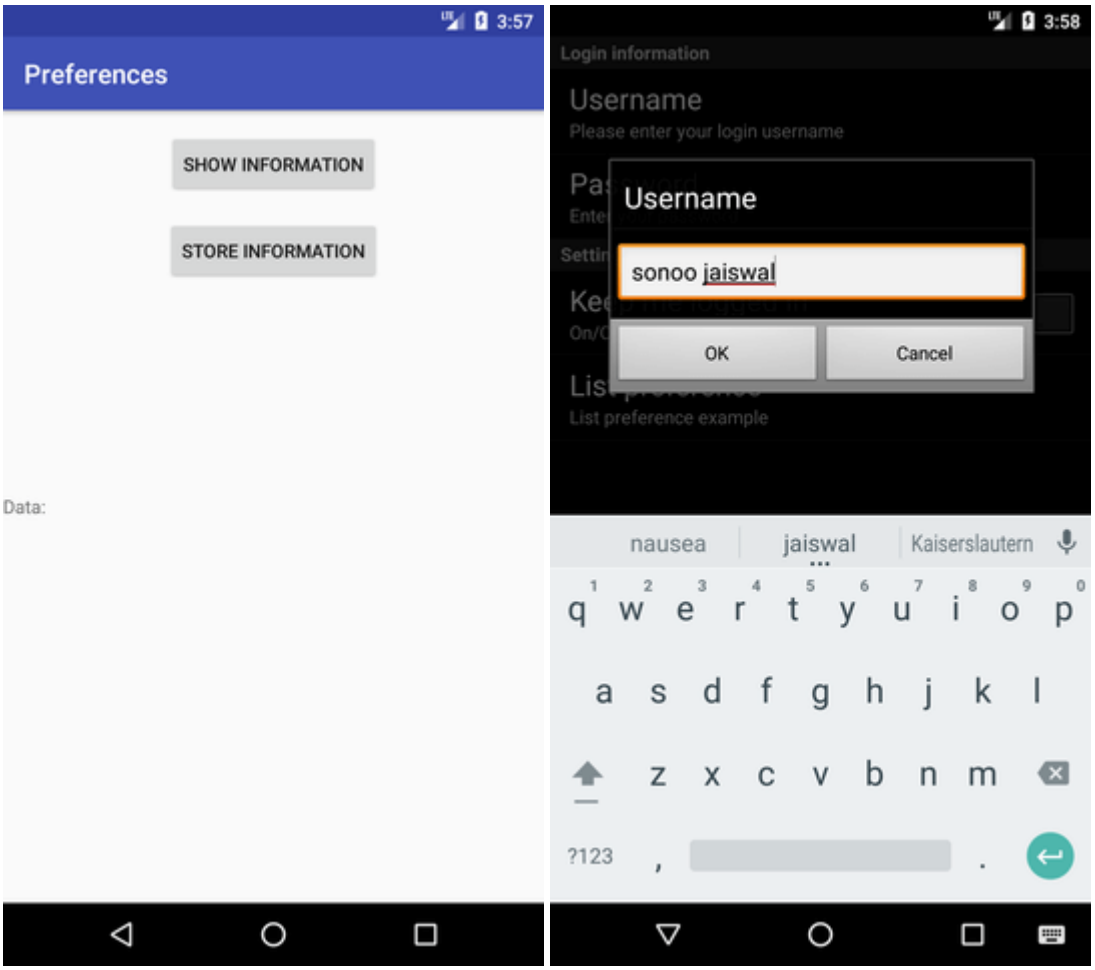
AndroidManifest.xml

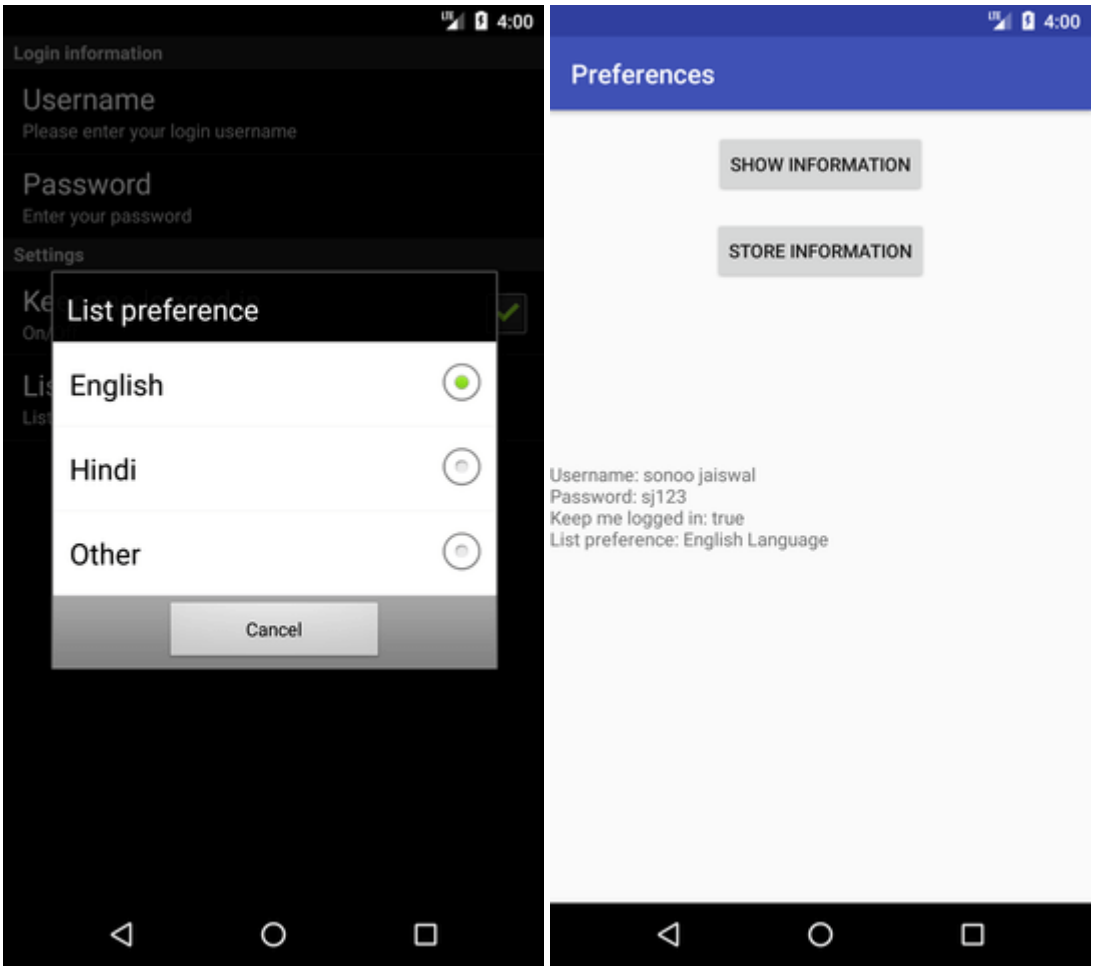
File: AndroidManifest.xml

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

```
2.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.    package="example.javatpoint.com.preferences">
4.
5.    <application
6.      android:allowBackup="true"
7.      android:icon="@mipmap/ic_launcher"
8.      android:label="@string/app_name"
9.      android:roundIcon="@mipmap/ic_launcher_round"
10.     android:supportsRtl="true"
11.     android:theme="@style/AppTheme">
12.     <activity android:name=".MainActivity">
13.       <intent-filter>
14.         <action android:name="android.intent.action.MAIN" />
15.
16.         <category android:name="android.intent.category.LAUNCHER" />
17.       </intent-filter>
18.     </activity>
19.     <activity  android:name=".PrefsActivity"
20.       android:theme="@android:style/Theme.Black.NoTitleBar" >
21.     </activity>
22.   </application>
23.
24. </manifest>
```

Output:


3:57
Preferences
SHOW INFORMATION
STORE INFORMATION
Data:
3:58
Login information
Username
Please enter your login username
Username
sonoo jaiswal
OK
Cancel
List preference example
nausea
jaiswal
Kaiserslautern



Android Internal Storage Example

We are able to save or read data from the device internal memory. FileInputStream and FileOutputStream classes are used to read and write data into the file.

Here, we are going to read and write data to the internal storage of the device.

Example of reading and writing data to the android internal 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>
```

Activity class

Let's write the code to write and read data from the internal storage.

File: MainActivity.java

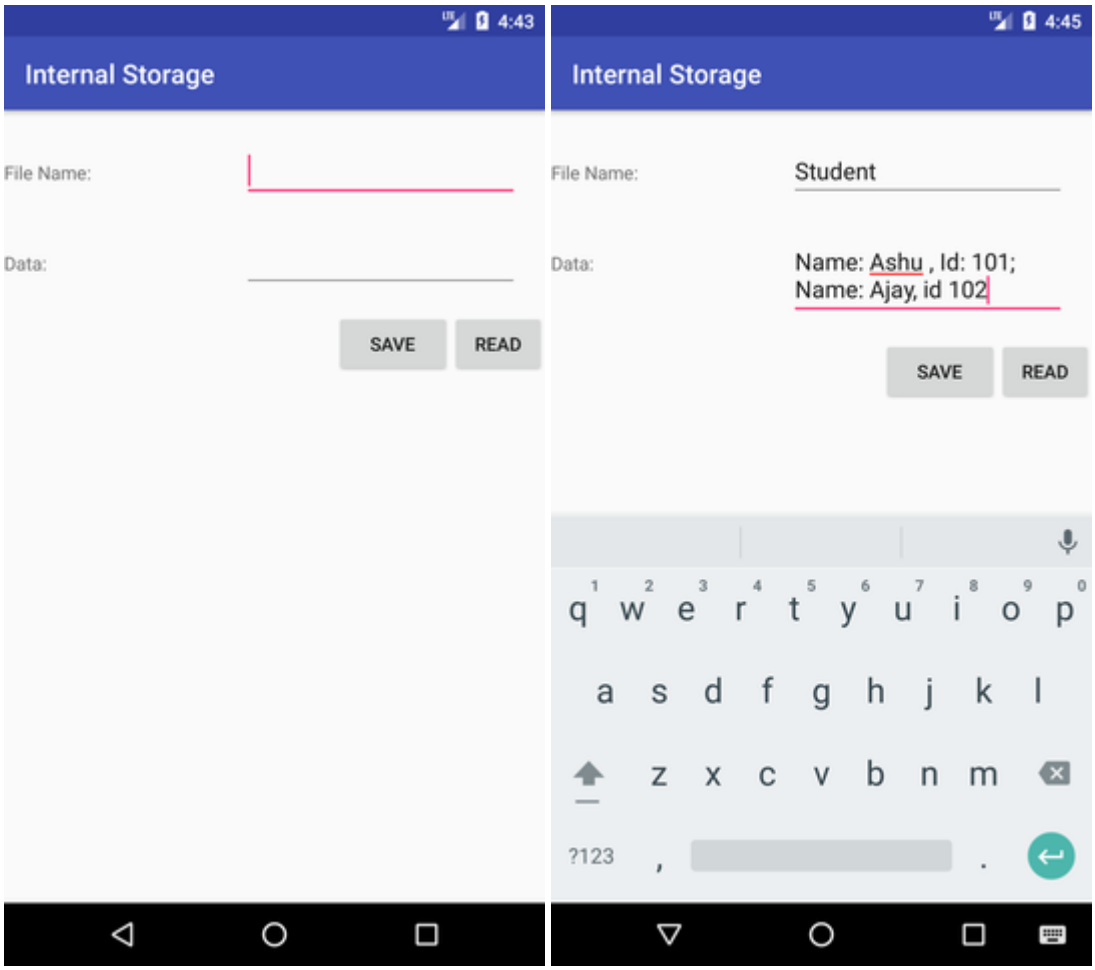
```
package example.javatpoint.com.internalstorage;

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

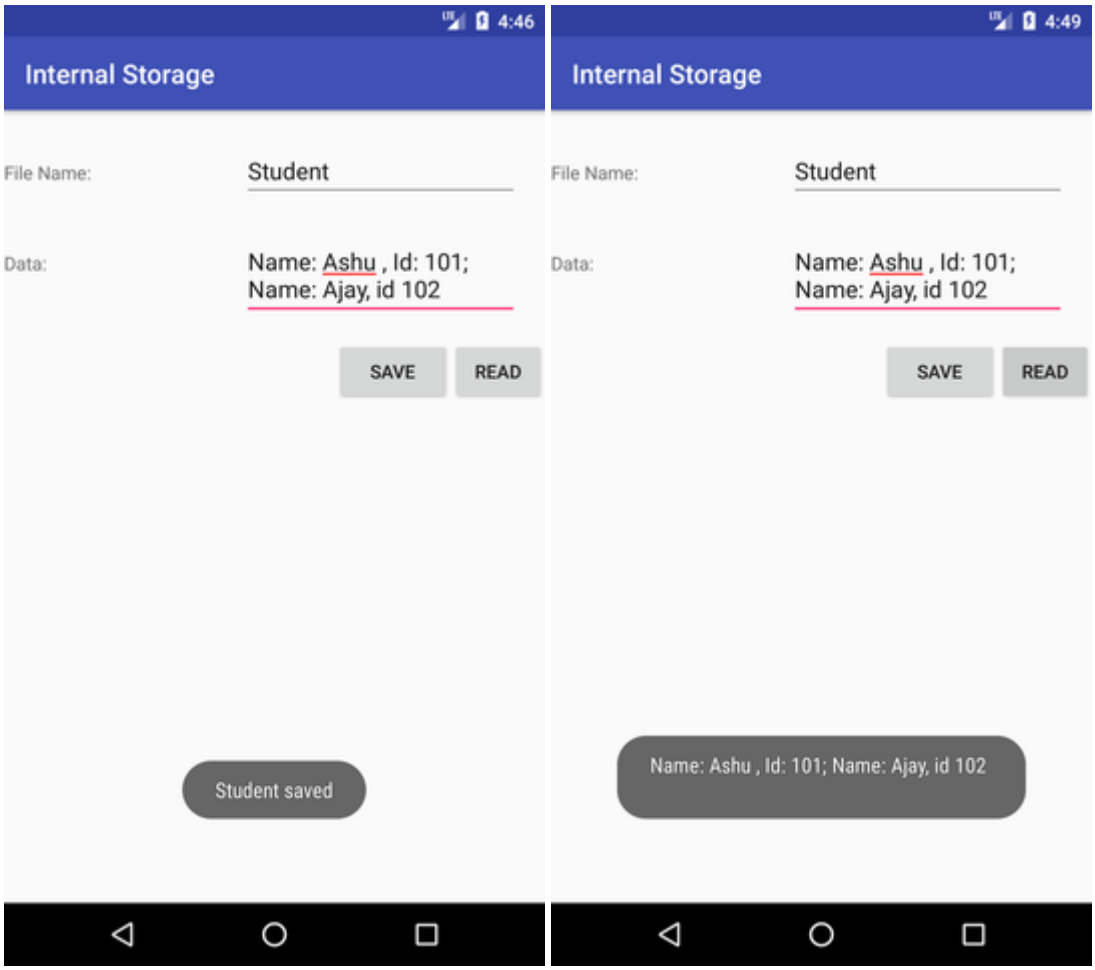
```
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;

public class MainActivity extends AppCompatActivity {
   EditText editTextFileName,editTextData;
   Button saveButton,readButton;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      editTextFileName=findViewById(R.id.editText1);
      editTextData=findViewById(R.id.editText2);
      saveButton=findViewById(R.id.button1);
      readButton=findViewById(R.id.button2);

      //Performing Action on Read Button
      saveButton.setOnClickListener(new View.OnClickListener(){

         @Override
         public void onClick(View arg0) {
            String filename=editTextFileName.getText().toString();
            String data=editTextData.getText().toString();

            FileOutputStream fos;
            try {
```

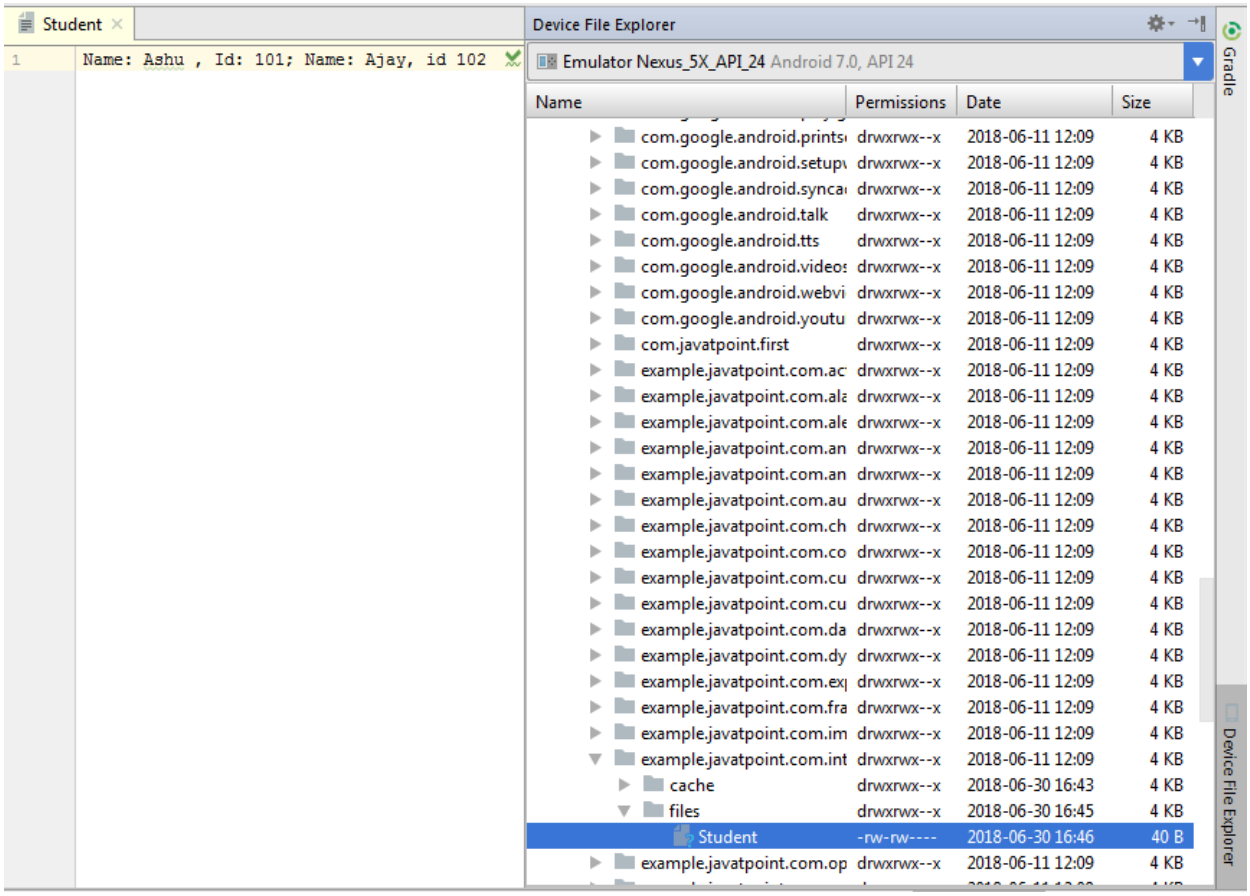
```
            fos = openFileOutput(filename, Context.MODE_PRIVATE);
            //default mode is PRIVATE, can be APPEND etc.
            fos.write(data.getBytes());
            fos.close();

            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 View.OnClickListener(){

        @Override
        public void onClick(View arg0) {
          String filename=editTextFileName.getText().toString();
          StringBuffer stringBuffer = new StringBuffer();
          try {
            //Attaching BufferedReader to the FileInputStream by the help of InputStreamRea
der
            BufferedReader inputReader = new BufferedReader(new InputStreamReader(
                  openFileInput(filename)));
            String inputString;
            //Reading data line by line and storing it into the stringbuffer
            while ((inputString = inputReader.readLine()) != null) {
              stringBuffer.append(inputString + "\n");
            }
```

```
            } catch (IOException e) {
                e.printStackTrace();
            }
            //Displaying data on the toast
        Toast.makeText(getApplicationContext(),stringBuffer.toString(),Toast.LENGTH_LONG).
show();

        }

    });
  }
}
```

Output:




4:46
Internal Storage
File Name:
Student
Data:
Name: Ashu , Id: 101;
Name: Ajay, id 102
SAVE
READ
Student saved
4:49
Internal Storage
File Name:
Student
Data:
Name: Ashu , Id: 101;
Name: Ajay, id 102
SAVE
READ
Name: Ashu , Id: 101; Name: Ajay, id 102


Student
1
Name: Ashu , Id: 101; Name: Ajay, id 102
Device File Explorer
Emulator Nexus_5X_API_24 Android 7.0, API 24
Name
Permissions
Date
Size
com.google.android.printsı drwxrwx--x 2018-06-11 12:09 4 KB
com.google.android.setupv drwxrwx--x 2018-06-11 12:09 4 KB
com.google.android.syncad drwxrwx--x 2018-06-11 12:09 4 KB
com.google.android.talk drwxrwx--x 2018-06-11 12:09 4 KB
com.google.android.tts drwxrwx--x 2018-06-11 12:09 4 KB
com.google.android.videos drwxrwx--x 2018-06-11 12:09 4 KB
com.google.android.webvi drwxrwx--x 2018-06-11 12:09 4 KB
com.google.android.youtu drwxrwx--x 2018-06-11 12:09 4 KB
com.javatpoint.first drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.ac drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.ala drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.ale drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.an drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.an drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.au drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.ch drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.co drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.cu drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.cu drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.da drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.dy drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.ex drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.fra drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.im drwxrwx--x 2018-06-11 12:09 4 KB
example.javatpoint.com.int drwxrwx--x 2018-06-11 12:09 4 KB
cache drwxrwx--x 2018-06-30 16:43 4 KB
files drwxrwx--x 2018-06-30 16:45 4 KB
Student -rw-rw---- 2018-06-30 16:46 40 B
example.javatpoint.com.op drwxrwx--x 2018-06-11 12:09 4 KB
Gradle
Device File Explorer

Android External Storage Example

Like internal storage, we are able to save or read data from the device external memory such as sdcard. The FileInputStream and FileOutputStream classes are used to read and write data into the file.

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

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context="example.javatpoint.com.externalstorage.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="example.javatpoint.com.externalstorage">
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

```
</manifest>
```

Activity class

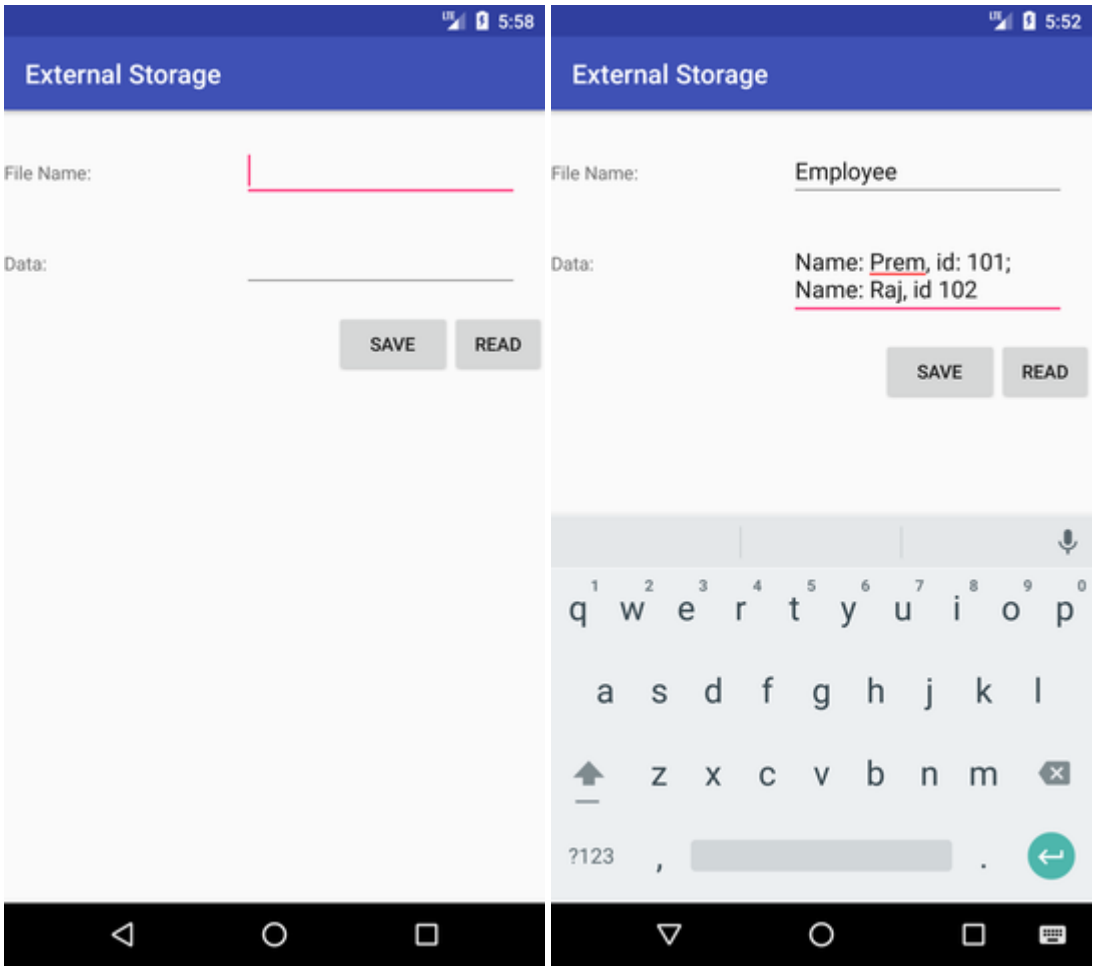
Let's write the code to write and read data from the android external storage.

File: MainActivity.java

```
package example.javatpoint.com.externalstorage;

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

import 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;

public class MainActivity extends AppCompatActivity {
   EditText editTextFileName,editTextData;
   Button saveButton,readButton;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      editTextFileName=findViewById(R.id.editText1);
      editTextData=findViewById(R.id.editText2);
      saveButton=findViewById(R.id.button1);
```

```
    readButton=findViewById(R.id.button2);

    //Performing action on save button
    saveButton.setOnClickListener(new View.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 View.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();
         }
      });
   }
}
```



Android - SQLite Database

SQLite is a opensource SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation.

SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC,ODBC e.t.c

What is SQLite?

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is the one database, which is zero-configured, that means like other database you do not need to configure it in your system.

SQLite engine is not a standalone process like other databases, you can link it statically or dynamically as per your requirement with your application. The SQLite accesses its storage files directly.

Why SQLite?

- SQLite does not require a separate server process or system to operate.(serverless).
- SQLite comes with zero-configuration, which means no setup or administration needed.
- A complete SQLite database is stored in a single cross-platform disk file.
- SQLite is very small and light weight, less than 400KiB fully configured or less than 250KiB with optional features omitted.
- SQLite is self-contained, which means no external dependencies.
- SQLite transactions are fully ACID-compliant, allowing safe access from multiple processes or threads.
- SQLite supports most of the query language features found in the SQL92 (SQL2) standard.
- SQLite is written in ANSI-C and provides simple and easy-to-use API.
- SQLite is available on UNIX (Linux, Mac OS-X, Android, iOS) and Windows (Win32, WinCE, WinRT).

History:

- 2000 -- D. Richard Hipp had designed SQLite for the purpose of no administration required for operating a program.
- 2000 -- In August SQLite 1.0 released with GNU Database Manager.
- 2011 -- Hipp announced to add UNQl interface to SQLite DB and to develop UNQLite (Document oriented database).

SQLite Limitations:

There are few unsupported features of SQL92 in SQLite which are shown below:

Feature	Description
RIGHT OUTER JOIN	Only LEFT OUTER JOIN is implemented.
FULL OUTER JOIN	Only LEFT OUTER JOIN is implemented.
ALTER TABLE	The RENAME TABLE and ADD COLUMN variants of the ALTER TABLE command are supported. The DROP COLUMN, ALTER COLUMN, ADD CONSTRAINT not

	supported.
Trigger support	FOR EACH ROW triggers are supported but not FOR EACH STATEMENT triggers.
VIEWs	VIEWs in SQLite are read-only. You may not execute a DELETE, INSERT, or UPDATE statement on a view.
GRANT and REVOKE	The only access permissions that can be applied are the normal file access permissions of the underlying operating system.

SQLite Commands:

The standard SQLite commands to interact with relational databases are similar as SQL. They are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their operational nature:

DDL - Data Definition Language:

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

DML - Data Manipulation Language:

Command	Description
INSERT	Creates a record
UPDATE	Modifies records
DELETE	Deletes records

DQL - Data Query Language:

Command	Description
SELECT	Retrieves certain records from one or more tables

Database - Package

The main package is android.database.sqlite that contains the classes to manage your own databases

Database - Creation

In order to create a database you just need to call this method openOrCreateDatabase with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own object.Its syntax is given below

```
SQLiteDatabase mydatabase = openOrCreateDatabase("your database name",MODE_PRIVATE,null);
```

Apart from this , there are other functions available in the database package , that does this job. They are listed below

Sr.No	Method & Description
1	**openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)** This method only opens the existing database with the appropriate flag mode. The common flags mode could be OPEN_READWRITE OPEN_READONLY
2	**openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)** It is similar to the above method as it also opens the existing database but it does not define any handler to handle the errors of databases
3	**openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)** It not only opens but create the database if it not exists. This method is equivalent to openDatabase method
4	**openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)** This method is similar to above method but it takes the File object as a path rather then a string. It is equivalent to file.getPath()

Database - Insertion

we can create table or insert data into table using execSQL method defined in SQLiteDatabase class. Its syntax is given below

```
mydatabase.execSQL("CREATE TABLE IF NOT EXISTS TutorialsPoint(Username VARCHAR,Password VARCHAR);");
mydatabase.execSQL("INSERT INTO TutorialsPoint VALUES('admin','admin');");
```

This will insert some values into our table in our database. Another method that also does the same job but take some additional parameter is given below

Sr.No	Method & Description
1	**execSQL(String sql, Object[] bindArgs)** This method not only insert data , but also used to update or modify already existing data in database using bind arguments

Database - Fetching

We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called rawQuery and it will return a resultset with the cursor pointing to the table. We can move the cursor forward and retrieve the data.

```
Cursor resultSet = mydatbase.rawQuery("Select * from TutorialsPoint",null);
resultSet.moveToFirst();
String username = resultSet.getString(1);
String password = resultSet.getString(2);
```

There are other functions available in the Cursor class that allows us to effectively retrieve the data. That includes

Sr.No	Method & Description
1	**getColumnCount()**This method return the total number of columns of the table.
2	**getColumnIndex(String columnName)** This method returns the index number of a column by specifying the name of the column
3	**getColumnName(int columnIndex)** This method returns the name of the column by specifying the index of the

	column
4	**getColumnNames()** This method returns the array of all the column names of the table.
5	**getCount()** This method returns the total number of rows in the cursor
6	**getPosition()** This method returns the current position of the cursor in the table
7	**isClosed()** This method returns true if the cursor is closed and return false otherwise

Database - Helper class

For managing all the operations related to the database , an helper class has been given and is called SQLiteOpenHelper. It automatically manages the creation and update of the database. Its syntax is given below

```
public class DBHelper extends SQLiteOpenHelper {
  public DBHelper(){
    super(context,DATABASE_NAME,null,1);
  }
  public void onCreate(SQLiteDatabase db) {}
  public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}
}
```

Example

Here is an example demonstrating the use of SQLite Database. It creates a basic contacts applications that allows insertion , deletion and modification of contacts.

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 to create an Android application under a package

	com.example.sairamkrishna.myapplication. While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.
2	Modify src/MainActivity.java file to get references of all the XML components and populate the contacts on listView.
3	Create new src/DBHelper.java that will manage the database work
4	Create a new Activity as DisplayContact.java that will display the contact on the screen
5	Modify the res/layout/activity_main to add respective XML components
6	Modify the res/layout/activity_display_contact.xml to add respective XML components
7	Modify the res/values/string.xml to add necessary string components
8	Modify the res/menu/display_contact.xml to add necessary menu components
9	Create a new menu as res/menu/mainmenu.xml to add the insert contact option
10	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 **MainActivity.java**.

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

import android.content.Context;
import android.content.Intent;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;

import android.view.KeyEvent;
import android.view.Menu;
```

```
import android.view.MenuItem;
import android.view.View;

import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends ActionBarActivity {
  public final static String EXTRA_MESSAGE = "MESSAGE";
  private ListView obj;
  DBHelper mydb;

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

    mydb = new DBHelper(this);
    ArrayList array_list = mydb.getAllCotacts();
    ArrayAdapter  arrayAdapter=new  ArrayAdapter(this,android.R.layout.simple_list_item_1,
array_list);

    obj = (ListView)findViewById(R.id.listView1);
    obj.setAdapter(arrayAdapter);
    obj.setOnItemClickListener(new OnItemClickListener(){
      @Override
      public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,long arg3) {
        // TODO Auto-generated method stub
        int id_To_Search = arg2 + 1;
```

```
        Bundle dataBundle = new Bundle();
        dataBundle.putInt("id", id_To_Search);

        Intent intent = new Intent(getApplicationContext(),DisplayContact.class);

        intent.putExtras(dataBundle);
        startActivity(intent);
      }
    });
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
  // Inflate the menu; this adds items to the action bar if it is present.
  getMenuInflater().inflate(R.menu.menu_main, menu);
  return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item){
  super.onOptionsItemSelected(item);

  switch(item.getItemId())
  {
    case R.id.item1:Bundle dataBundle = new Bundle();
    dataBundle.putInt("id", 0);

    Intent intent = new Intent(getApplicationContext(),DisplayContact.class);
    intent.putExtras(dataBundle);

    startActivity(intent);
    return true;
    default:
    return super.onOptionsItemSelected(item);
```

```
    }
  }

  public boolean onKeyDown(int keycode, KeyEvent event) {
    if (keycode == KeyEvent.KEYCODE_BACK) {
      moveTaskToBack(true);
    }
    return super.onKeyDown(keycode, event);
  }
}
```

Following is the modified content of display contact activity**DisplayContact.java**

```
package com.example.addressbook;

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

import android.content.DialogInterface;
import android.content.Intent;
import android.database.Cursor;

import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class DisplayContact extends Activity {
  int from_Where_I_Am_Coming = 0;
  private DBHelper mydb ;

  TextView name ;
```

```
  TextView phone;
  TextView email;
  TextView street;
  TextView place;
  int id_To_Update = 0;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_display_contact);
    name = (TextView) findViewById(R.id.editTextName);
    phone = (TextView) findViewById(R.id.editTextPhone);
    email = (TextView) findViewById(R.id.editTextStreet);
    street = (TextView) findViewById(R.id.editTextEmail);
    place = (TextView) findViewById(R.id.editTextCity);

    mydb = new DBHelper(this);

    Bundle extras = getIntent().getExtras();
    if(extras !=null)
    {
      int Value = extras.getInt("id");

      if(Value>0){
        //means this is the view part not the add contact part.
        Cursor rs = mydb.getData(Value);
        id_To_Update = Value;
        rs.moveToFirst();

        String nam = rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_NAME));
        String phon =
rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_PHONE));
        String emai = rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_EMAIL));
```

```
        String stree =
rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_STREET));
        String plac = rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_CITY));

        if (!rs.isClosed())
        {
          rs.close();
        }
        Button b = (Button)findViewById(R.id.button1);
        b.setVisibility(View.INVISIBLE);

        name.setText((CharSequence)nam);
        name.setFocusable(false);
        name.setClickable(false);

        phone.setText((CharSequence)phon);
        phone.setFocusable(false);
        phone.setClickable(false);

        email.setText((CharSequence)emai);
        email.setFocusable(false);
        email.setClickable(false);

        street.setText((CharSequence)stree);
        street.setFocusable(false);
        street.setClickable(false);

        place.setText((CharSequence)plac);
        place.setFocusable(false);
        place.setClickable(false);
      }
    }
  }
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  // Inflate the menu; this adds items to the action bar if it is present.
  Bundle extras = getIntent().getExtras();

  if(extras !=null)
  {
    int Value = extras.getInt("id");
    if(Value>0){
      getMenuInflater().inflate(R.menu.display_contact, menu);
    }

    else{
      getMenuInflater().inflate(R.menu.main, menu);
    }
  }
  return true;
}

public boolean onOptionsItemSelected(MenuItem item)
{
  super.onOptionsItemSelected(item);
  switch(item.getItemId())
{
  case R.id.Edit_Contact:
  Button b = (Button)findViewById(R.id.button1);
  b.setVisibility(View.VISIBLE);
  name.setEnabled(true);
  name.setFocusableInTouchMode(true);
  name.setClickable(true);

  phone.setEnabled(true);
  phone.setFocusableInTouchMode(true);
  phone.setClickable(true);
```

```
    email.setEnabled(true);
    email.setFocusableInTouchMode(true);
    email.setClickable(true);

    street.setEnabled(true);
    street.setFocusableInTouchMode(true);
    street.setClickable(true);

    place.setEnabled(true);
    place.setFocusableInTouchMode(true);
    place.setClickable(true);

    return true;
    case R.id.Delete_Contact:

    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setMessage(R.string.deleteContact)
    .setPositiveButton(R.string.yes, new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int id) {
        mydb.deleteContact(id_To_Update);
        Toast.makeText(getApplicationContext(), "Deleted Successfully",
Toast.LENGTH_SHORT).show();
        Intent intent = new Intent(getApplicationContext(),MainActivity.class);
        startActivity(intent);
      }
    })
    .setNegativeButton(R.string.no, new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int id) {
        // User cancelled the dialog
      }
    });
    AlertDialog d = builder.create();
    d.setTitle("Are you sure");
```

```
    d.show();

    return true;
    default:
    return super.onOptionsItemSelected(item);

    }
  }

  public void run(View view)
  {
    Bundle extras = getIntent().getExtras();
    if(extras !=null)
    {
      int Value = extras.getInt("id");
      if(Value>0){
        if(mydb.updateContact(id_To_Update,name.getText().toString(),
phone.getText().toString(),          email.getText().toString(),          street.getText().toString(),
place.getText().toString())){
          Toast.makeText(getApplicationContext(), "Updated", Toast.LENGTH_SHORT).show();
          Intent intent = new Intent(getApplicationContext(),MainActivity.class);
          startActivity(intent);
        }
        else{
          Toast.makeText(getApplicationContext(), "not Updated",
Toast.LENGTH_SHORT).show();
        }
      }
      else{
        if(mydb.insertContact(name.getText().toString(), phone.getText().toString(),
email.getText().toString(), street.getText().toString(), place.getText().toString())){
          Toast.makeText(getApplicationContext(), "done", Toast.LENGTH_SHORT).show();
        }
```

```
        else{
          Toast.makeText(getApplicationContext(), "not done", Toast.LENGTH_SHORT).show();
        }
        Intent intent = new Intent(getApplicationContext(),MainActivity.class);
        startActivity(intent);
      }
    }
  }
}
```

Following is the content of Database class **DBHelper.java**

```
package com.example.addressbook;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Hashtable;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.DatabaseUtils;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase;

public class DBHelper extends SQLiteOpenHelper {

  public static final String DATABASE_NAME = "MyDBName.db";
  public static final String CONTACTS_TABLE_NAME = "contacts";
  public static final String CONTACTS_COLUMN_ID = "id";
  public static final String CONTACTS_COLUMN_NAME = "name";
  public static final String CONTACTS_COLUMN_EMAIL = "email";
  public static final String CONTACTS_COLUMN_STREET = "street";
  public static final String CONTACTS_COLUMN_CITY = "place";
  public static final String CONTACTS_COLUMN_PHONE = "phone";
  private HashMap hp;
```

```
  public DBHelper(Context context)
  {
    super(context, DATABASE_NAME , null, 1);
  }

  @Override
  public void onCreate(SQLiteDatabase db) {
    // TODO Auto-generated method stub
    db.execSQL(
    "create table contacts " +
    "(id integer primary key, name text,phone text,email text, street text,place text)"
    );
  }

  @Override
  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // TODO Auto-generated method stub
    db.execSQL("DROP TABLE IF EXISTS contacts");
    onCreate(db);
  }

  public boolean insertContact  (String name, String phone, String email, String street,String
place)
  {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put("name", name);
    contentValues.put("phone", phone);
    contentValues.put("email", email);
    contentValues.put("street", street);
    contentValues.put("place", place);
    db.insert("contacts", null, contentValues);
    return true;
  }
```

```
  public Cursor getData(int id){
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor res =  db.rawQuery( "select * from contacts where id="+id+"", null );
    return res;
  }

  public int numberOfRows(){
    SQLiteDatabase db = this.getReadableDatabase();
    int numRows = (int) DatabaseUtils.queryNumEntries(db, CONTACTS_TABLE_NAME);
    return numRows;
  }

  public boolean updateContact (Integer id, String name, String phone, String email, String
street,String place)
  {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put("name", name);
    contentValues.put("phone", phone);
    contentValues.put("email", email);
    contentValues.put("street", street);
    contentValues.put("place", place);
    db.update("contacts", contentValues, "id = ? ", new String[] { Integer.toString(id) } );
    return true;
  }

  public Integer deleteContact (Integer id)
  {
    SQLiteDatabase db = this.getWritableDatabase();
    return db.delete("contacts",
    "id = ? ",
    new String[] { Integer.toString(id) });
  }
```

```
  public ArrayList<String> getAllCotacts()
  {
    ArrayList<String> array_list = new ArrayList<String>();
        //hp = new HashMap();
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor res =  db.rawQuery( "select * from contacts", null );
    res.moveToFirst();

    while(res.isAfterLast() == false){
      array_list.add(res.getString(res.getColumnIndex(CONTACTS_COLUMN_NAME)));
      res.moveToNext();
    }
  return array_list;
  }
}
```

Following is the content of the **res/layout/activity_main.xml**

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

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/textView"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:textSize="30dp"
      android:text="Data Base" />
```

```
<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="35dp"
   android:textColor="#ff16ff01" />

<ImageView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:id="@+id/imageView"
   android:layout_below="@+id/textView2"
   android:layout_centerHorizontal="true"
   android:src="@drawable/logo"/>

<ScrollView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:id="@+id/scrollView"
   android:layout_below="@+id/imageView"
   android:layout_alignParentLeft="true"
   android:layout_alignParentStart="true"
   android:layout_alignParentBottom="true"
   android:layout_alignParentRight="true"
   android:layout_alignParentEnd="true">

<ListView
   android:id="@+id/listView1"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
```

```
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true" >
  </ListView>
  </ScrollView>

</RelativeLayout>;
```

Following is the content of the **res/layout/activity_display_contact.xml**

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/scrollView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    tools:context=".DisplayContact" >

<RelativeLayout
  android:layout_width="match_parent"
  android:layout_height="370dp"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin">

  <EditText
    android:id="@+id/editTextName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_marginTop="5dp"
    android:layout_marginLeft="82dp"
    android:ems="10"
    android:inputType="text" >
  </EditText>

  <EditText
```

```
    android:id="@+id/editTextEmail"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editTextStreet"
    android:layout_below="@+id/editTextStreet"
    android:layout_marginTop="22dp"
    android:ems="10"
    android:inputType="textEmailAddress" />

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/editTextName"
    android:layout_alignParentLeft="true"
    android:text="@string/name"
    android:textAppearance="?android:attr/textAppearanceMedium" />

  <Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editTextCity"
    android:layout_alignParentBottom="true"
    android:layout_marginBottom="28dp"
    android:onClick="run"
    android:text="@string/save" />

  <TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/editTextEmail"
    android:layout_alignLeft="@+id/textView1"
```

```
    android:text="@string/email"
    android:textAppearance="?android:attr/textAppearanceMedium" />

  <TextView
    android:id="@+id/textView5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/editTextPhone"
    android:layout_alignLeft="@+id/textView1"
    android:text="@string/phone"
    android:textAppearance="?android:attr/textAppearanceMedium" />

  <TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/editTextEmail"
    android:layout_alignLeft="@+id/textView5"
    android:text="@string/street"
    android:textAppearance="?android:attr/textAppearanceMedium" />

  <EditText
    android:id="@+id/editTextCity"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@+id/editTextName"
    android:layout_below="@+id/editTextEmail"
    android:layout_marginTop="30dp"
    android:ems="10"
    android:inputType="text" />

  <TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/editTextCity"
    android:layout_alignBottom="@+id/editTextCity"
    android:layout_alignParentLeft="true"
    android:layout_toLeftOf="@+id/editTextEmail"
    android:text="@string/country"
    android:textAppearance="?android:attr/textAppearanceMedium" />

  <EditText
    android:id="@+id/editTextStreet"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editTextName"
    android:layout_below="@+id/editTextPhone"
    android:ems="10"
    android:inputType="text" >

    <requestFocus />
  </EditText>

  <EditText
    android:id="@+id/editTextPhone"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editTextStreet"
    android:layout_below="@+id/editTextName"
    android:ems="10"
    android:inputType="phone|text" />

</RelativeLayout>
</ScrollView>
```

Following is the content of the **res/value/string.xml**

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

```
    <string name="app_name">Address Book</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="Add_New">Add New</string>
    <string name="edit">Edit Contact</string>
    <string name="delete">Delete Contact</string>
    <string name="title_activity_display_contact">DisplayContact</string>
    <string name="name">Name</string>
    <string name="phone">Phone</string>
    <string name="email">Email</string>
    <string name="street">Street</string>
    <string name="country">City/State/Zip</string>
    <string name="save">Save Contact</string>
    <string name="deleteContact">Are you sure, you want to delete it.</string>
    <string name="yes">Yes</string>
    <string name="no">No</string>
</resources>
```

Following is the content of the **res/menu/main_menu.xml**

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

    <item android:id="@+id/item1"
        android:icon="@drawable/add"
        android:title="@string/Add_New" >
    </item>

</menu>
```

Following is the content of the **res/menu/display_contact.xml**

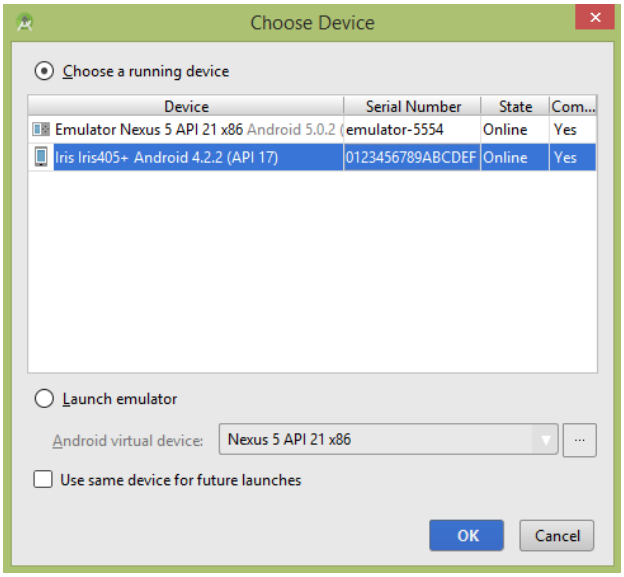
```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/Edit_Contact"
        android:orderInCategory="100"
        android:title="@string/edit"/>
```

```
  <item
    android:id="@+id/Delete_Contact"
    android:orderInCategory="100"
    android:title="@string/delete"/>

</menu>
```

This is the defualt **AndroidManifest.xml** of this project

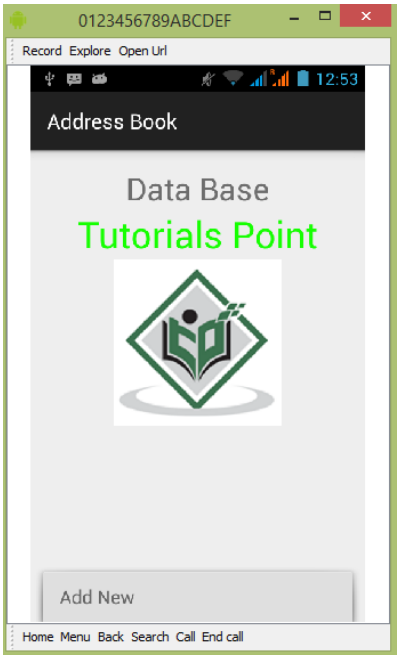
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.sairamkrishna.myapplication" >

  <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>

    <activity android:name=".DisplayContact"/>

  </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.



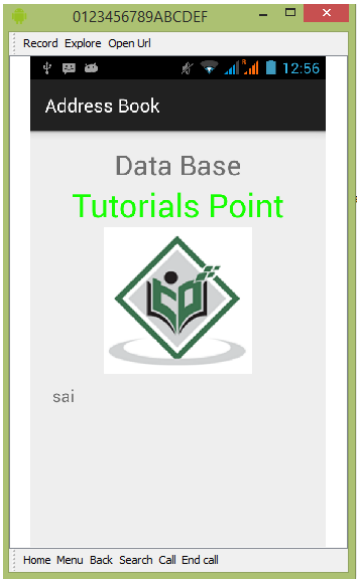
Select your mobile device as an option and then check your mobile device which will display following screen –

Now open your optional menu, it will show as below image: **Optional menu appears different places on different versions**



Click on the add button of the menu screen to add a new contact. It will display the following screen –

It will display the following fields. Please enter the required information and click on save contact. It will bring you back to main screen.

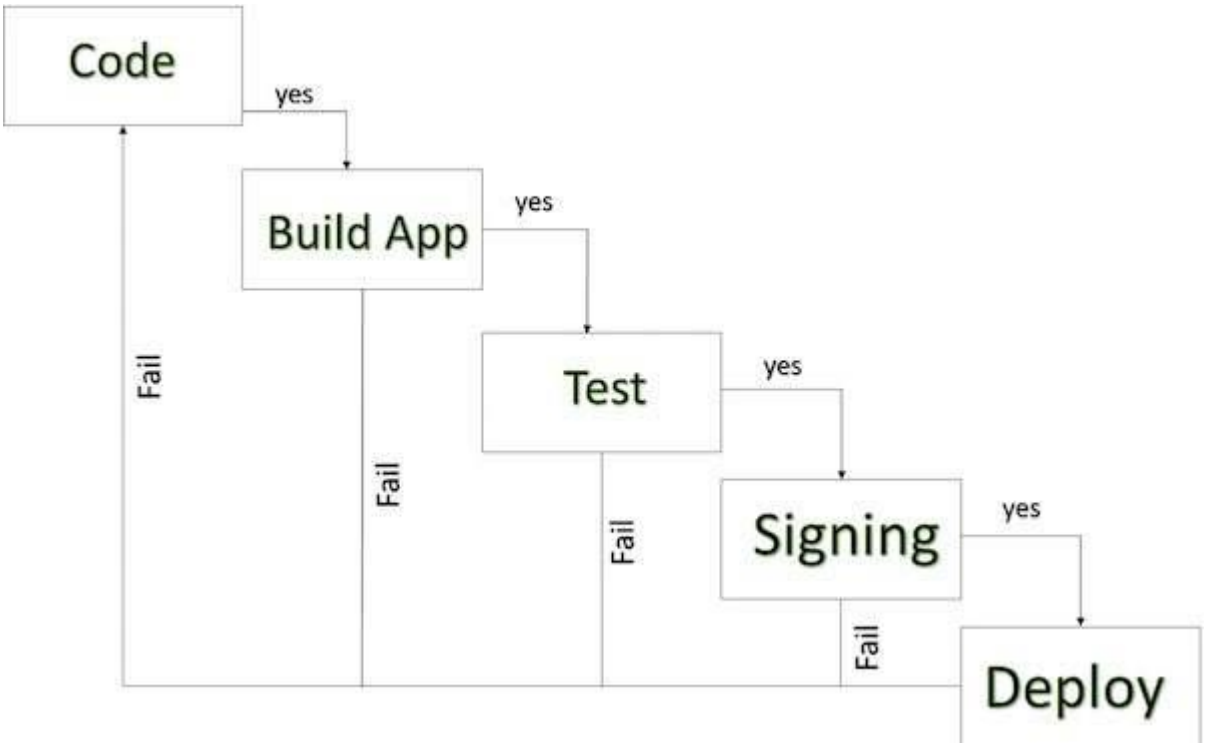


Now our contact john has been added. Tap on this to edit it or delete it.It will bring you to the following screen. Now select menu from your mobile. And there will be two options there.

Now our contact sai has been added.In order to see that where is your database is created. Open your android studio, connect your mobile. Go **tools/android/android device monitor**. Now browse the file explorer tab. Now browse this folder **/data/data/<your.package.name>/databases<database-name>.**

Publishing Android Application

Android application publishing is a process that makes your Android applications available to users. Infect, publishing is the last phase of the Android application development process.



Android development life cycle

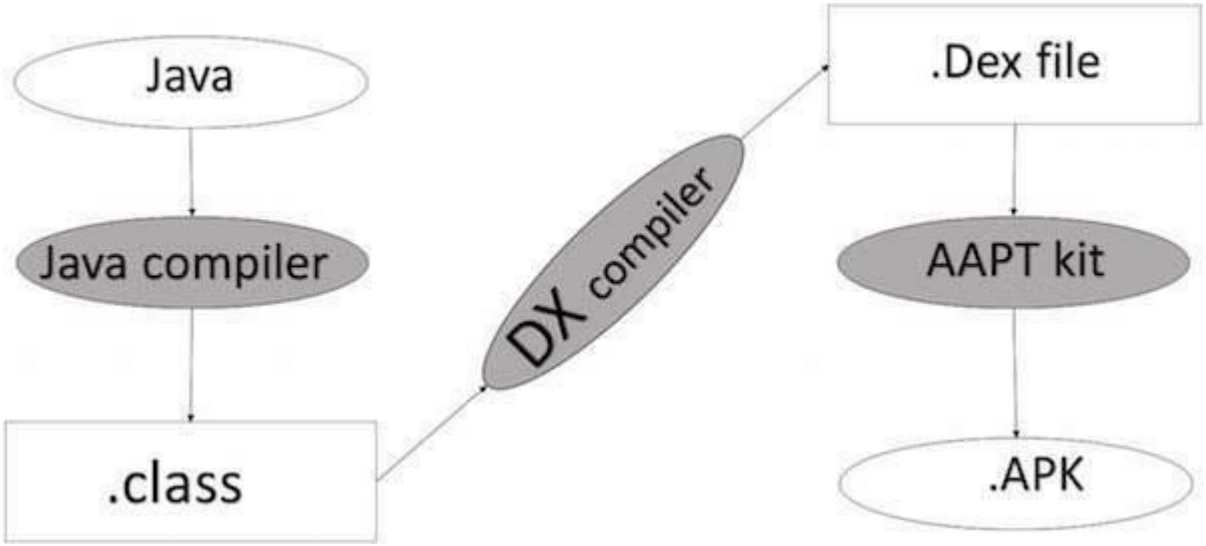
Once you developed and fully tested your Android Application, you can start selling or distributing free using Google Play (A famous Android marketplace). You can also release your applications by sending them directly to users or by letting users download them from your own website.

You can check a detailed publishing process at Android official website, but this tutorial will take you through simple steps to launch your application on Google Play. Here is a simplified check list which will help you in launching your Android application –

Step	Activity
1	**Regression Testing** Before you publish your application, you need to make sure that its meeting the basic quality expectations for all Android apps, on all of the devices that you are targeting. So perform all the required testing on different devices including phone and tablets.
2	**Application Rating** When you will publish your application at Google Play, you will have to specify a content rating for your app, which informs Google Play users of its maturity level. Currently available ratings are (a) Everyone (b) Low maturity (c) Medium maturity (d) High maturity.
3	**Targeted Regions** Google Play lets you control what countries and territories where your application will be sold. Accordingly you must take care of setting up time zone, localization or any other specific requirement as per the targeted region.
4	**Application Size** Currently, the maximum size for an APK published on Google Play is 50 MB. If your app exceeds that size, or if you want to offer a secondary download, you can use APK Expansion Files, which Google Play will host for free on its server infrastructure and automatically handle the download to devices.
5	**SDK and Screen Compatibility** It is important to make sure that your app is designed to run properly on the Android platform versions and device screen sizes that you want to target.
6	**Application Pricing** Deciding whether you app will be free or paid is important because, on Google Play, free app's must remain free. If you want to sell your

	application then you will have to specify its price in different currencies.
7	**Promotional Content** It is a good marketing practice to supply a variety of high-quality graphic assets to showcase your app or brand. After you publish, these appear on your product details page, in store listings and search results, and elsewhere.
8	**Build and Upload release-ready APK** The release-ready APK is what you you will upload to the Developer Console and distribute to users. You can check complete detail on how to create a release-ready version of your app: Preparing for Release.
9	**Finalize Application Detail** Google Play gives you a variety of ways to promote your app and engage with users on your product details page, from colourful graphics, screen shots, and videos to localized descriptions, release details, and links to your other apps. So you can decorate your application page and provide as much as clear crisp detail you can provide.

Export Android Application Process



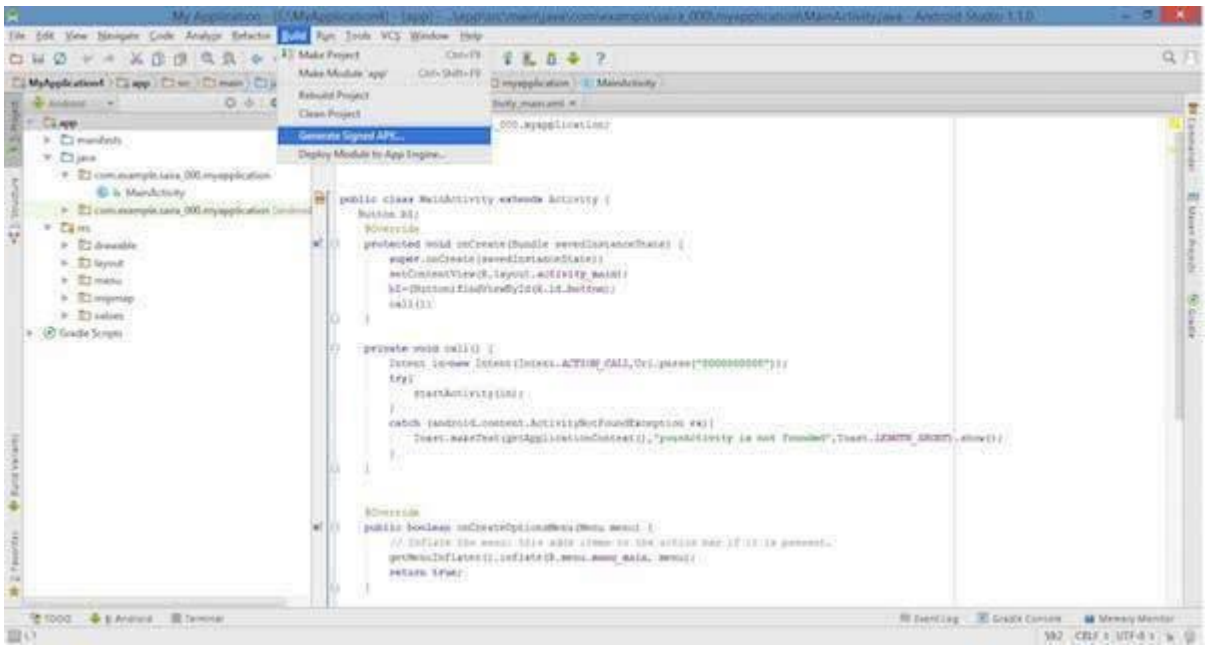
Apk development process

Before exporting the apps, you must some of tools

- **Dx tools**(Dalvik executable tools): It going to convert **.class file** to **.dex file**. it has useful for memory optimization and reduce the boot-up speed time
- **AAPT**(Android assistance packaging tool):it has useful to convert **.Dex file** to**.Apk**
- **APK**(Android packaging kit): The final stage of deployment process is called as .apk.

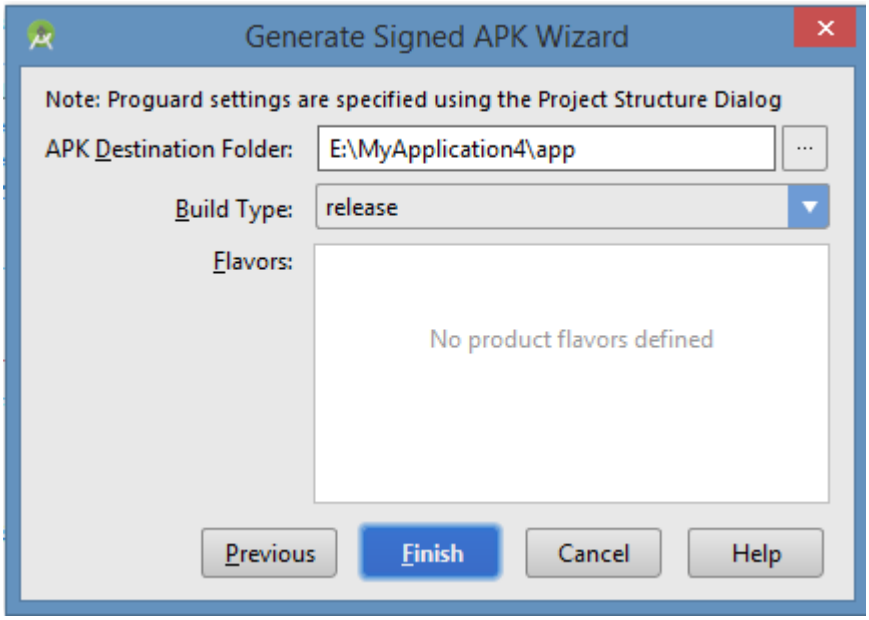
You will need to export your application as an APK (Android Package) file before you upload it Google Play marketplace.

To export an application, just open that application project in Android studio and select **Build → Generate Signed APK** from your Android studio and follow the simple steps to export your application –

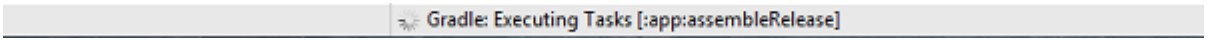


Next select, **Generate Signed APK** option as shown in the above screen shot and then click it so that you get following screen where you will choose **Create new keystore** to store your application.

Enter your key store path,key store password,key alias and key password to protect your application and click on **Next** button once again. It will display following screen to let you create an application –



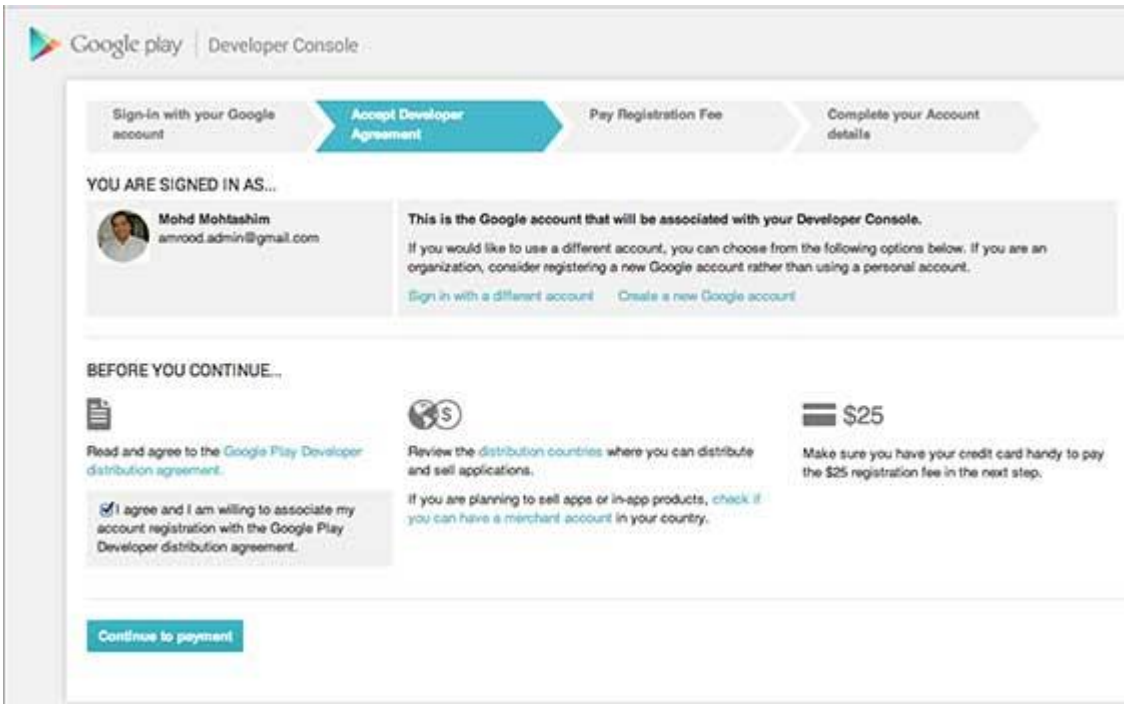
Once you filled up all the information,like app destination,build type and flavours click **finish** button While creating an application it will show as below



Finally, it will generate your Android Application as APK formate File which will be uploaded at Google Play marketplace.

Google Play Registration

The most important step is to register with Google Play using Google Play Marketplace. You can use your existing google ID if you have any otherwise you can create a new Google ID and then register with the marketplace. You will have following screen to accept terms and condition.



You can use **Continue to payment** button to proceed to make a payment of $25 as a registration fee and finally to complete your account detail.

Once you are a registered user at Google Play, you can upload **release-ready APK** for your application and finally you will complete application detail using application detail page as mentioned in step 9 of the above mentioned checklist.

Signing Your App Manually

You do not need Android Studio to sign your app. You can sign your app from the command line using standard tools from the Android SDK and the JDK. To sign an app in release mode from the command line –

- Generate a private key using keytool

```
$ keytool -genkey -v -keystore my-release-key.keystore
-alias alias_name -keyalg RSA -keysize 2048 -validity 10000
```

- Compile your app in release mode to obtain an unsigned APK
- Sign your app with your private key using jarsigner

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1
-keystore my-release-key.keystore my_application.apk alias_name
```

- Verify that your APK is signed. For example –

```
$ jarsigner -verify -verbose -certs my_application.apk
```

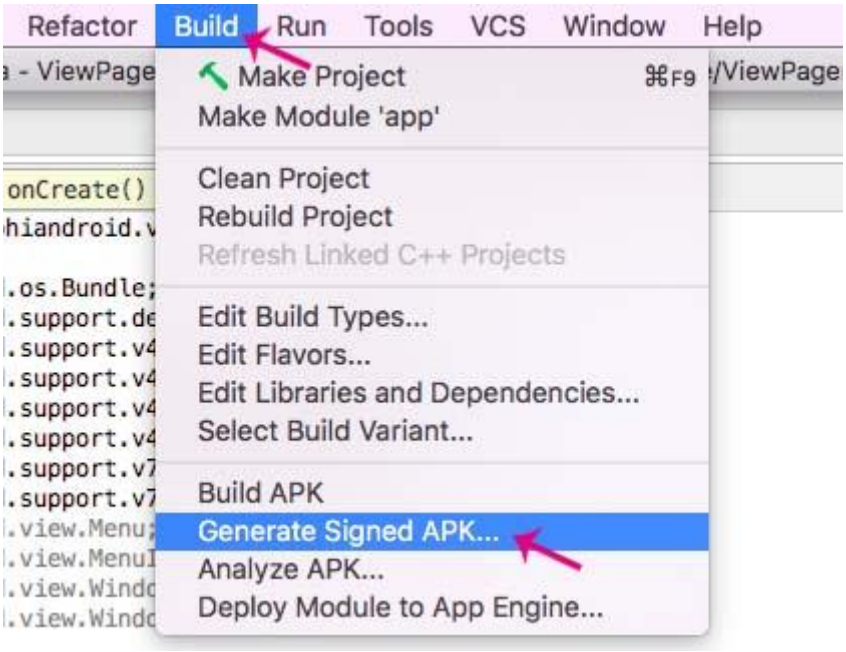
- Align the final APK package using zipalign.

```
$ zipalign -v 4 your_project_name-unaligned.apk your_project_name.apk
```

How To Generate Signed Apk For Publishing New App In Android Studio:

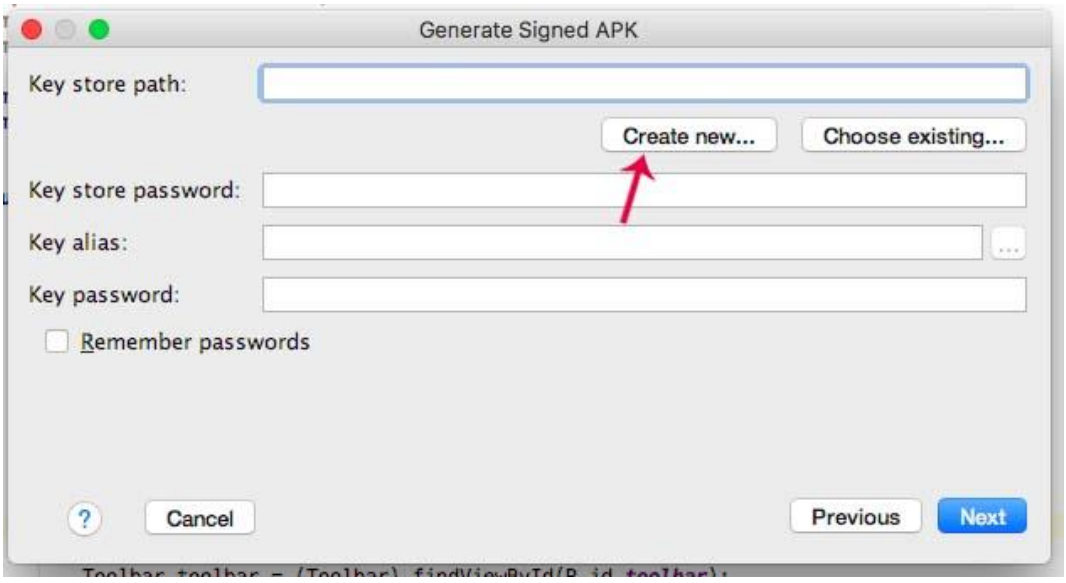
Follow the below steps:

Step 1: Go to build and click on Generate Signed Apk...

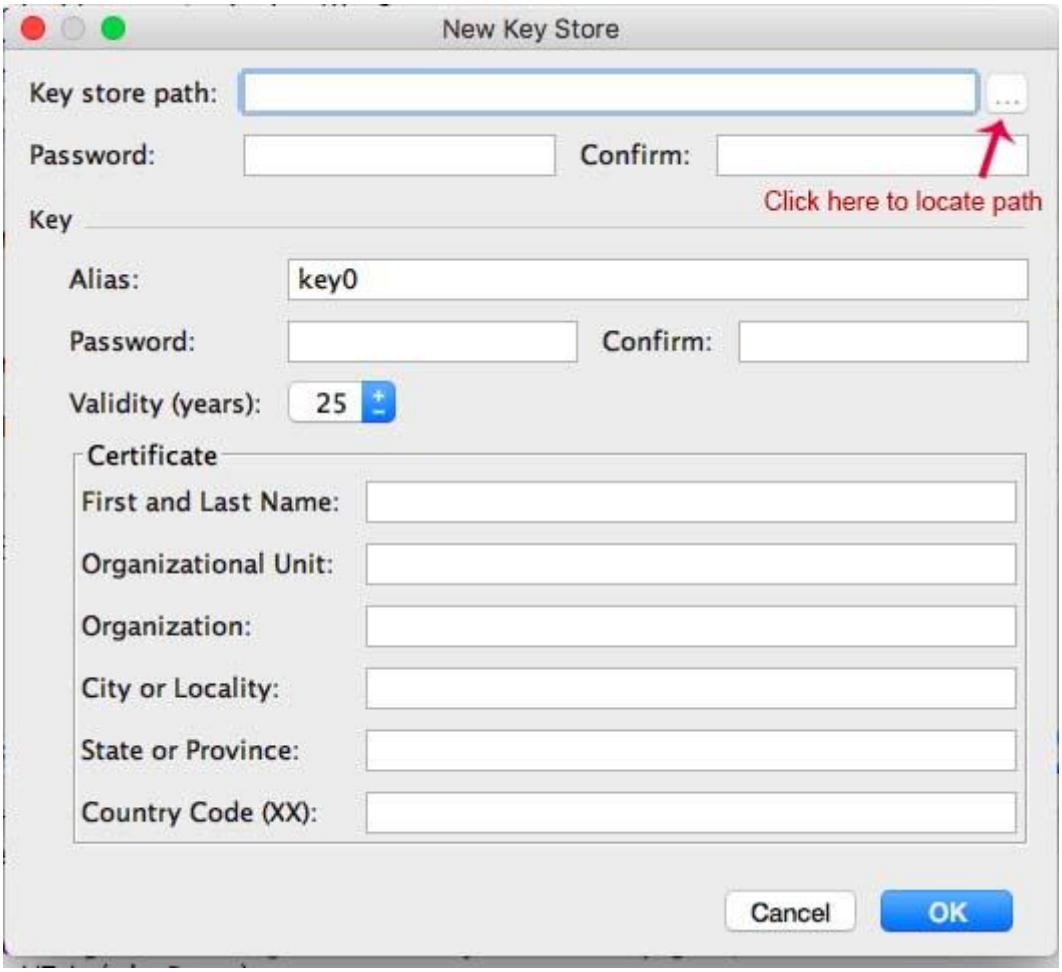


Step 2: Android Studio will now open a dialog box. Click on Next.

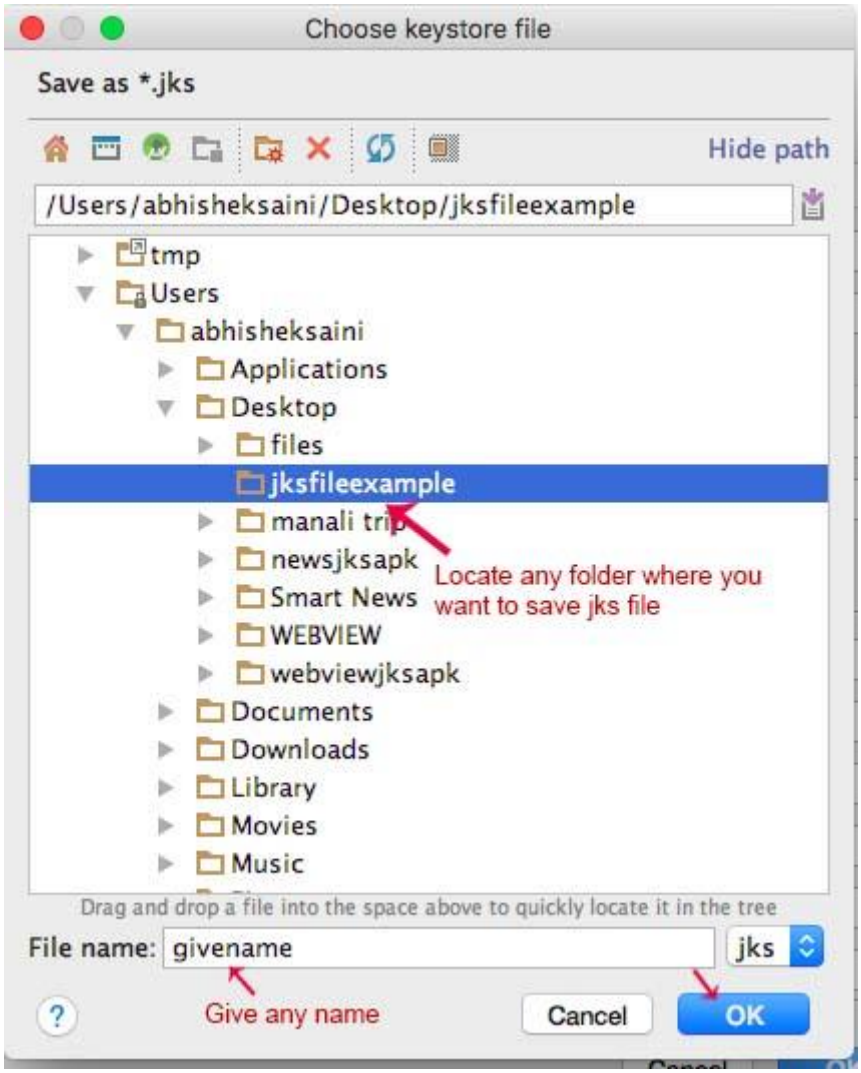
Step 3: Now you will need to create Keystore path. Click on Create new...



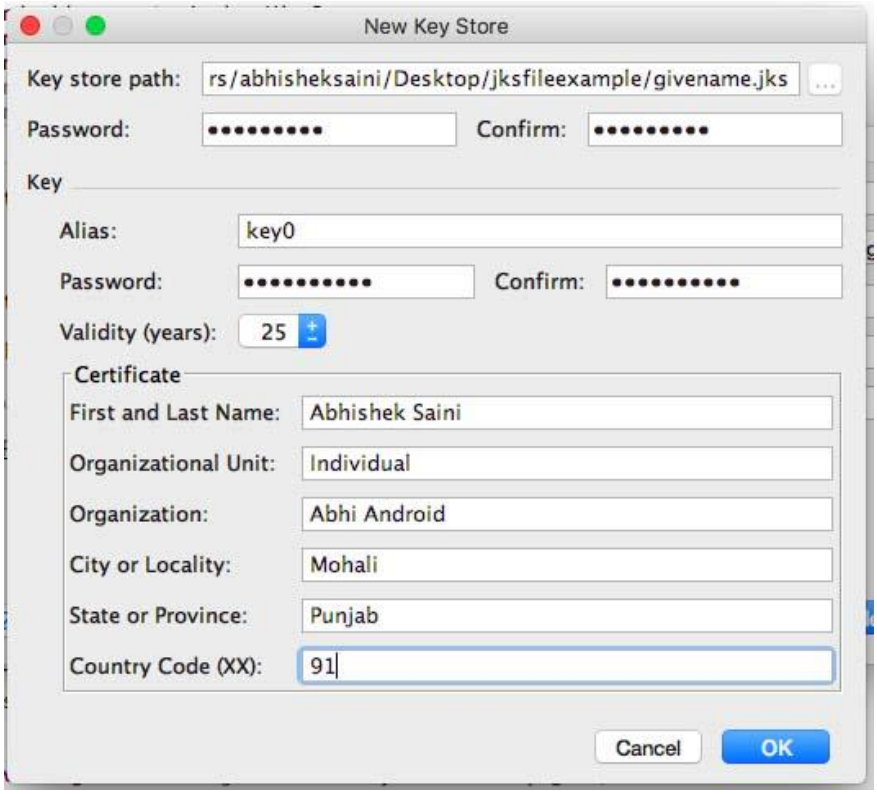
Step 4: Now locate key store path in your system where you want to save jks file of your project. Click on … to locate the path.



Step 5: After locating the path from your system. Give any name to the jks file that will be created and click ok.



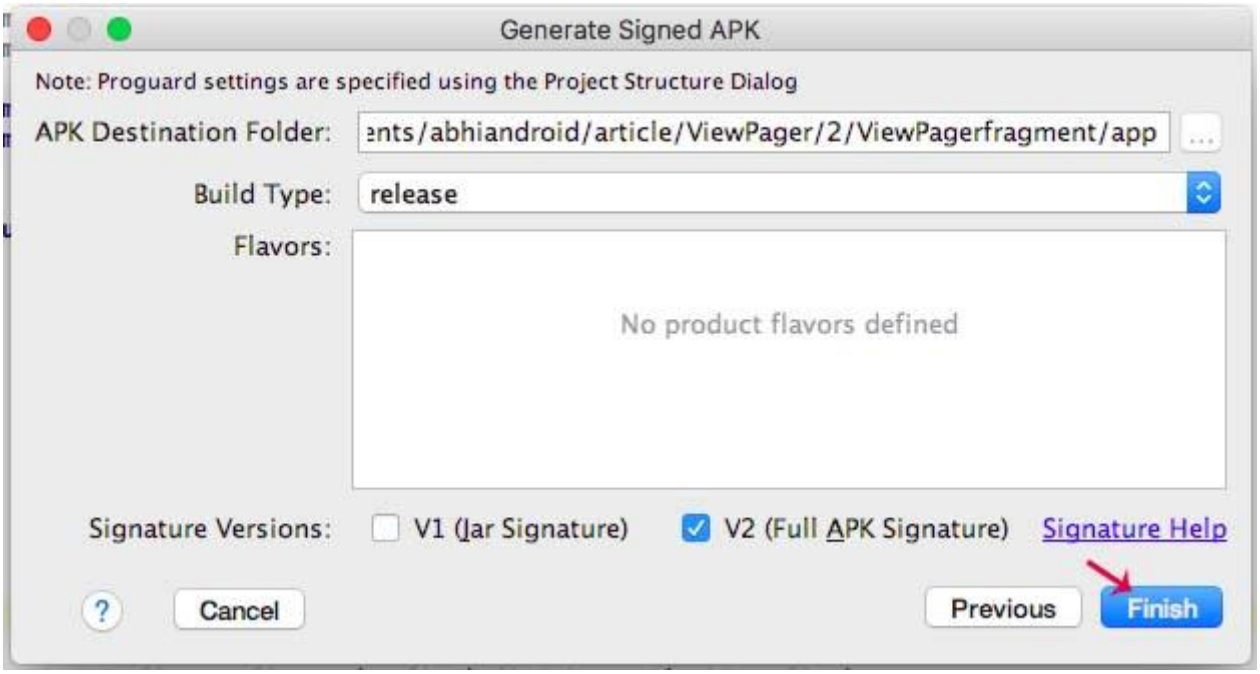
Step 6: Fill the other details and click ok. For example you can refer to the image below:



Step 7: Click next.

Step 8: Now edit the destination folder of signed apk file, choose build type and select signature versions. Finally click Finish:

Important Note: Regarding signature versions, you can consider V2 because it offers faster app install times and more protection against unauthorized alterations to APK files. If you face any problem then you can consider the traditional V1 signing scheme, which uses JAR signing.



Step 9: Now you can use this signed apk for publishing app on Playstore via your developer console.

Important Note 1: After you generate signed apk, it is very important to keep jks file safe and secure. If you lost it then you won't be able to send future updates to your App.

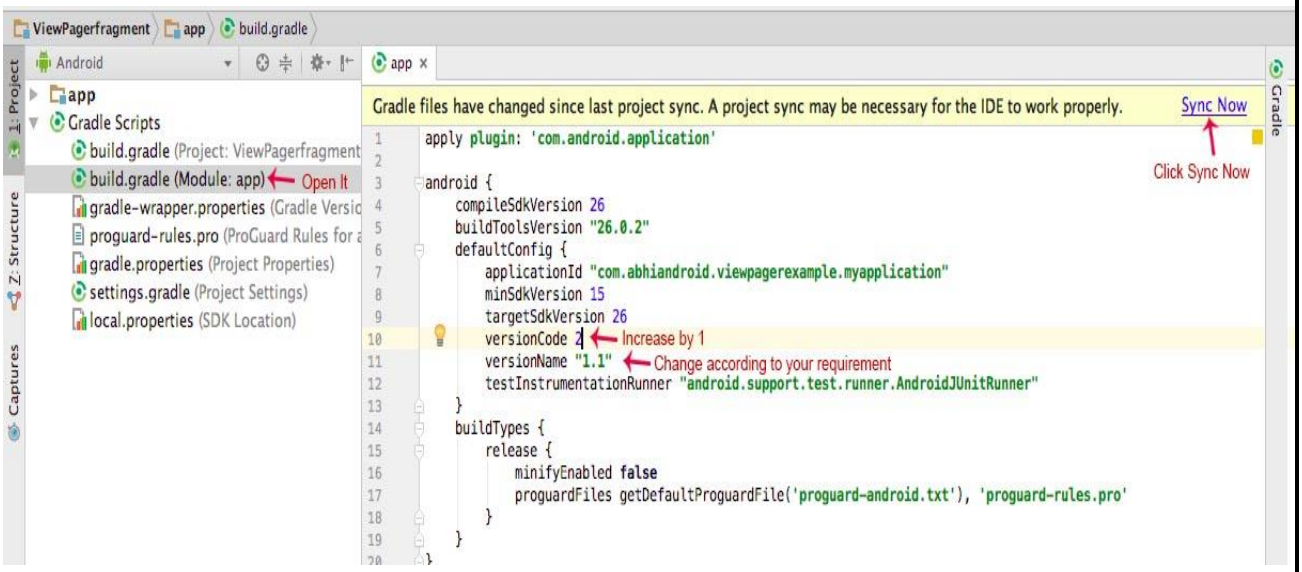
Important Note 2: Make sure to keep Keystore and key password saved somewhere with you. If you lost or forget it then also you won't be able to send new updates for your App.

How To Generate Signed apk For Updating Existing App On Playstore In Android Studio:

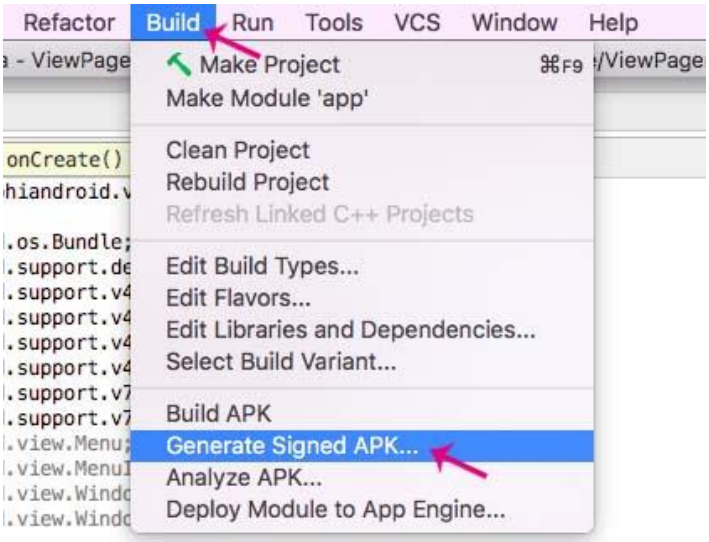
Follow the below steps to generate the signed apk for updating existing Android App on Playstore:

Step 1: Open the build.gradle (Module: app) file and increase the version code by 1 and change the version name according to your requirement

Step 2: Click Sync Now



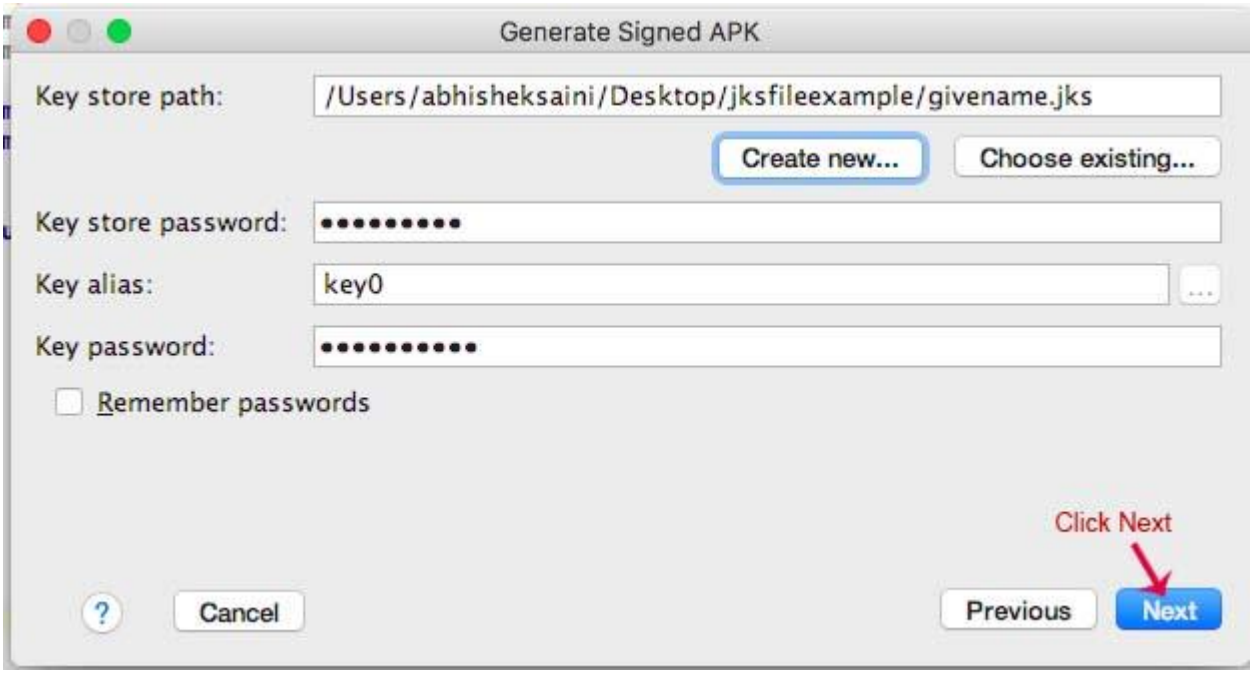
Step 3: After sync is completed. Click on build and generate signed apk…



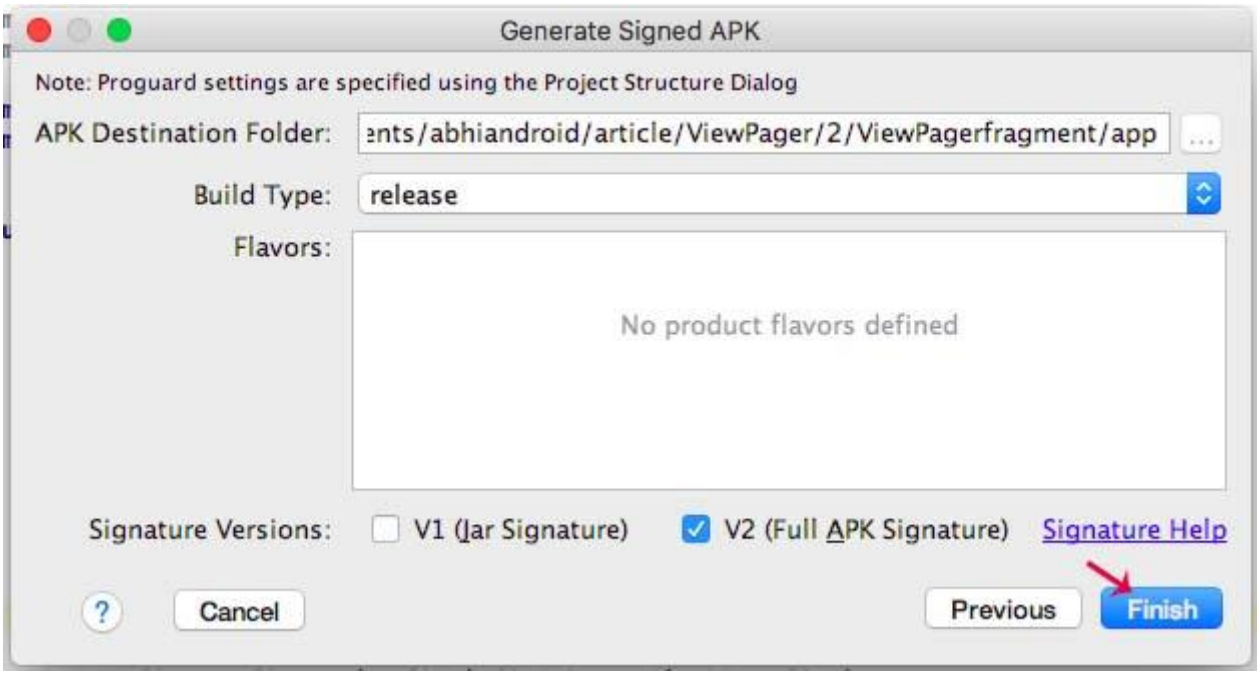
Step 4: Now click on Next

Step 5: Click on **choose existing...** and locate the path of jks of the App in your system.

Step 6: Enter the Key store password, key alias and key password that you created while creating jks file



Step 8: Now edit the destination folder of signed apk file, choose build type and select signature versions. Finally click Finish:



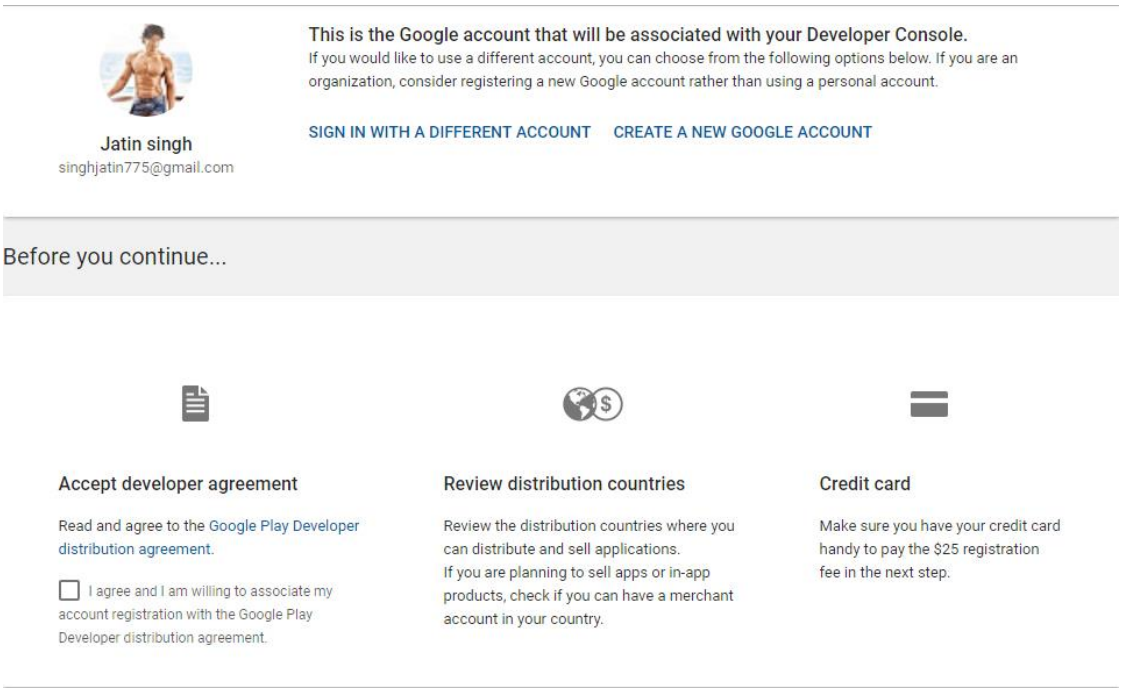
Step 9: Now you can use this signed apk for updating existing app on Playstore via your Android developer console.

How To Publish Android App On PlayStore:

Follow the below steps:

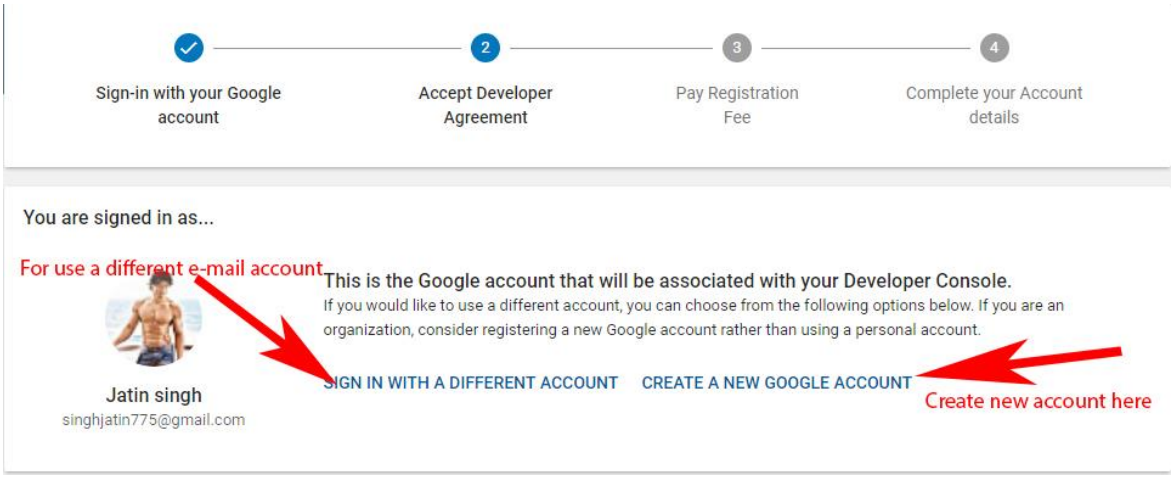
Step 1: First generate signed apk of your Android App to publish it on Play Store.

Step 2: Now you will need to sign up for Google Play Console to publish and manage your Android App.



Important Note: You can signup with this link https://play.google.com/apps/publish/

Step 3: Login with your Gmail account that you want to use for publishing App on Play Store.

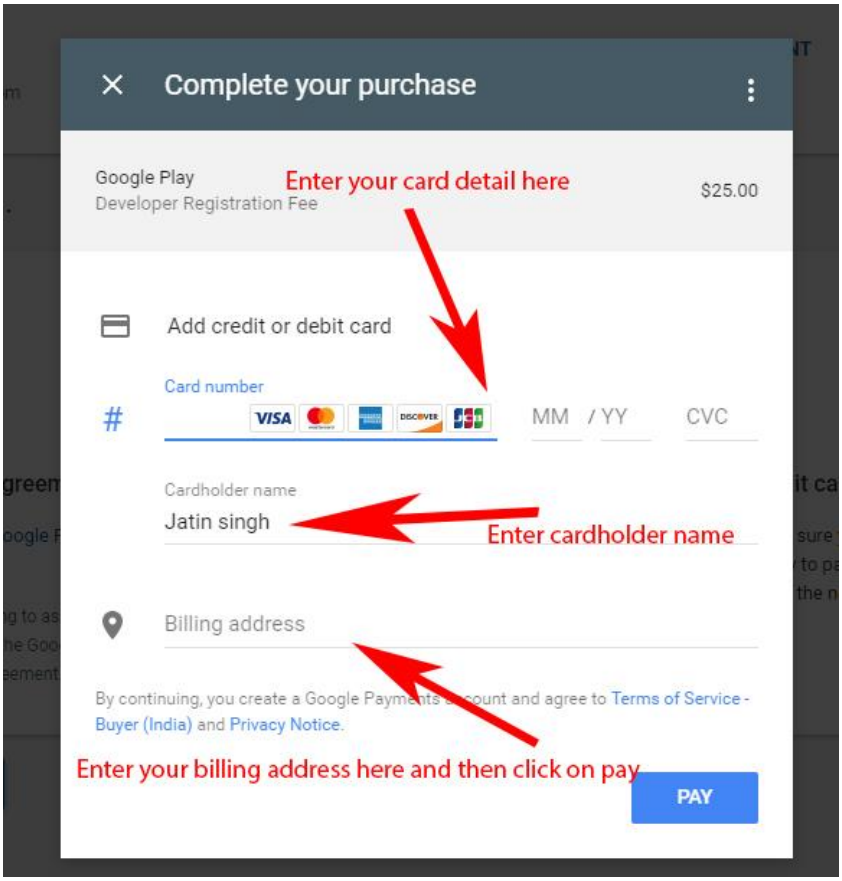


Step 4: Now there are 4 steps to complete the registration for Google play store console. You have already completed two.

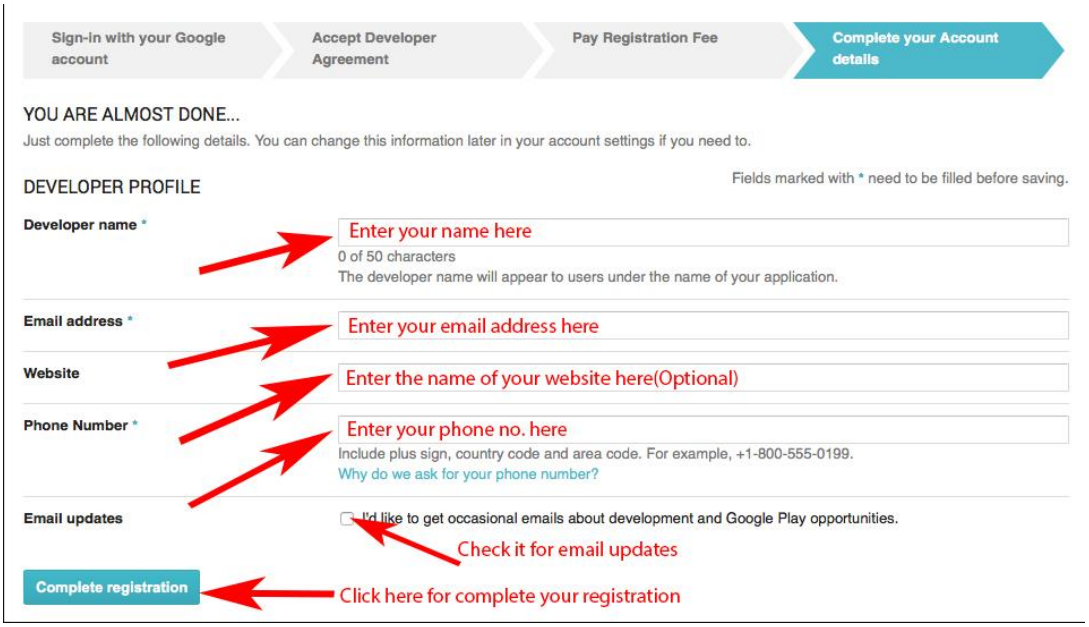
Step 5: After reading the Google play store developer distribution agreement agree to their terms by clicking on check box

Step 6: Now you will need to pay one time 'Developer Registration Fee' of $25 to Google. Please fill your credit card details to make the payment.

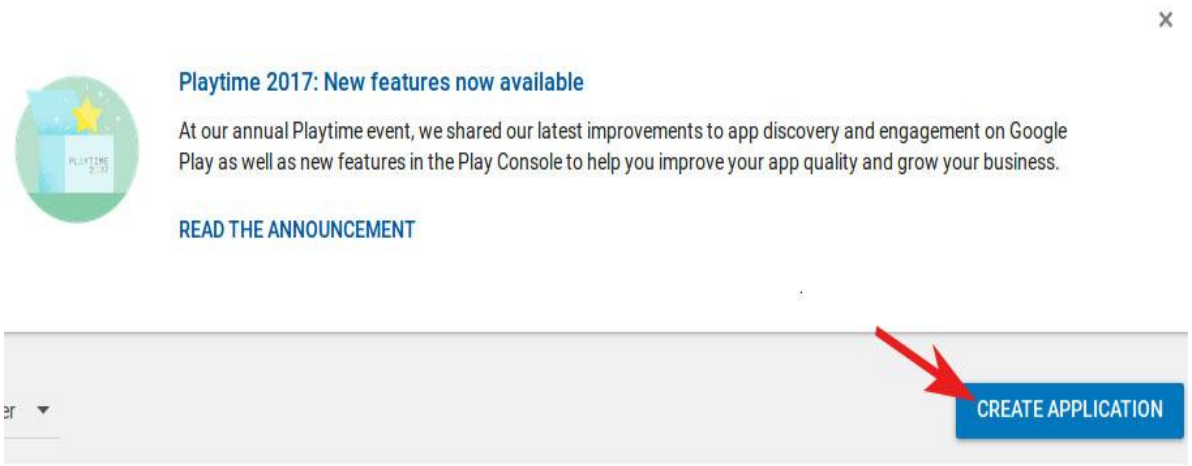
Important Note: You can upload unlimited number of Android App on Play store from single account with a limit of uploading 15 apk/day.



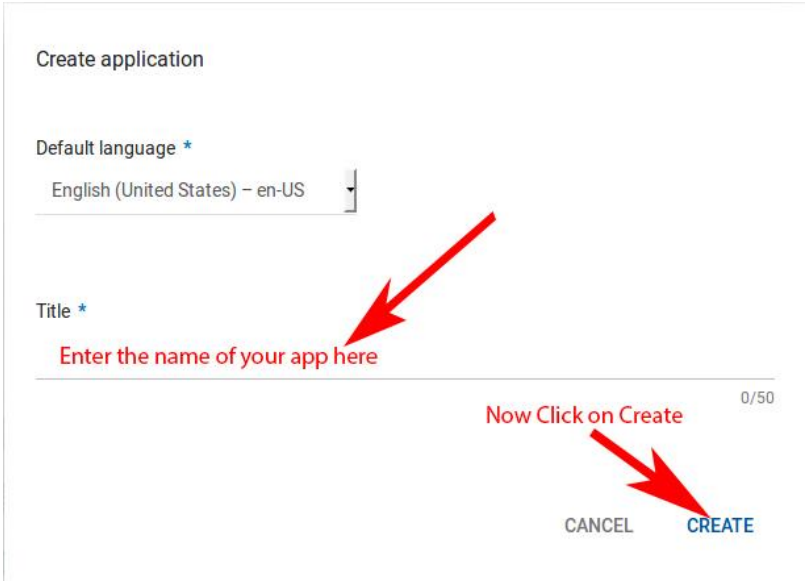
Step 7: Complete your account details for Google developer account. For example see the below image:



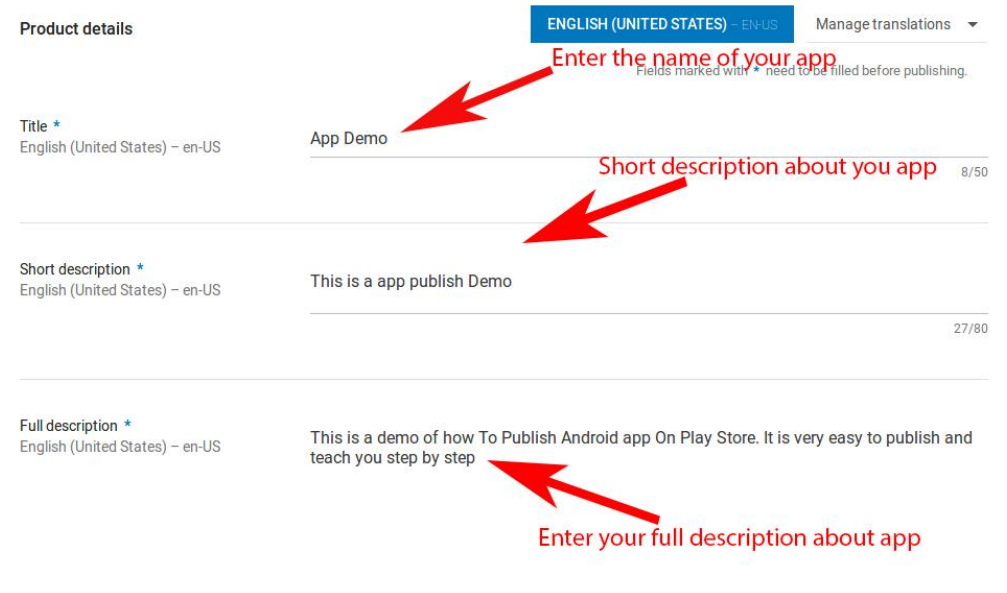
Step 8: Now click on Create Application



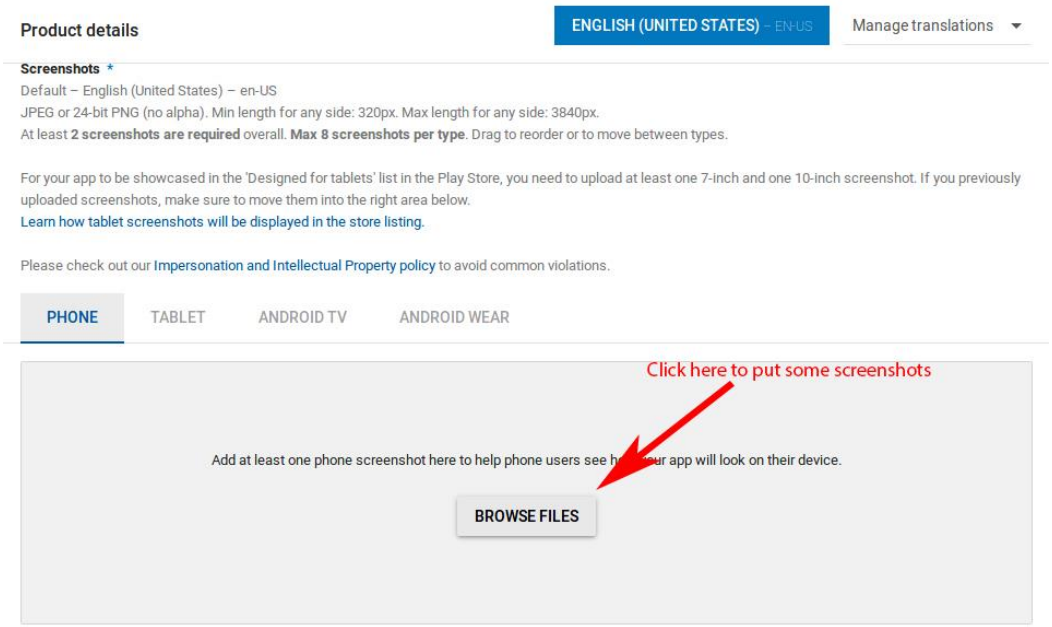
Step 9: Enter the name of your App.



Step 10: Now fill store listing details of your App which include Title, Short description, and Full description.

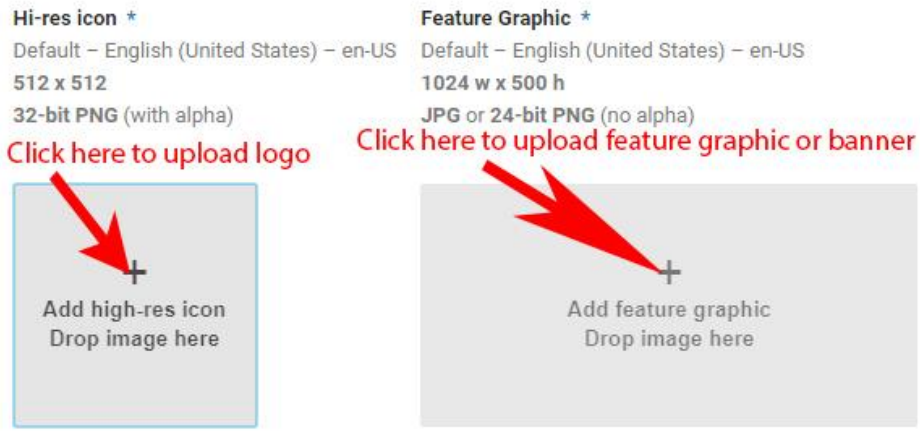


Step 11: After this you need to put some App screenshots here. The minimum required are 2 screenshots and maximum limit is 8.



Step 12: After screenshot now you need to put a high Resolution icon or logo with a size of 512 * 512 pixel. This will be displayed on Play Store.

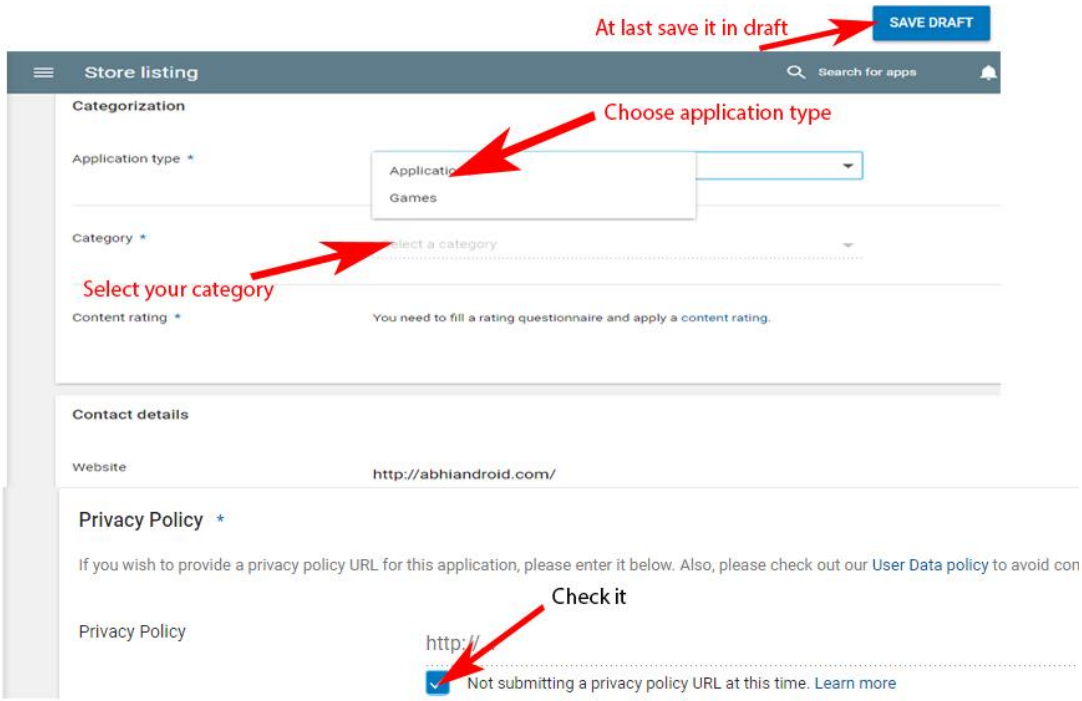
After that another mandatory thing is you need to put a feature graphic of 1024 * 500 pixel dimension. See below image for more detail.



Step 13: Now scroll down and fill other details which include application type, category, website, email and phone no.

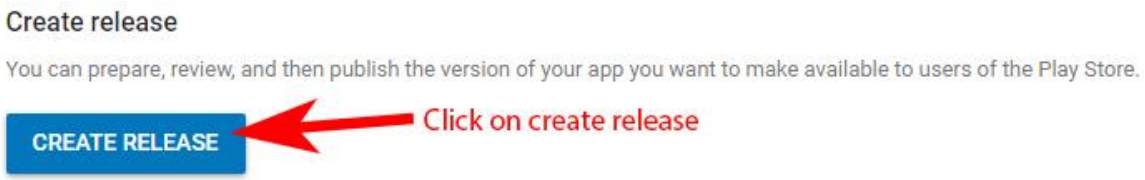
After this check privacy policy because now we are not submitting and then click on save draft. If your App require user permission then it is mandatory to put privacy url.

Click on Save Draft to save your work so far.

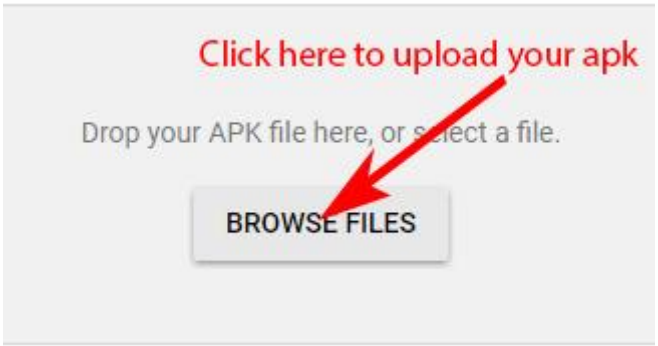


Step 14: After saving data on draft now go to **app release** and click on **manage production**.

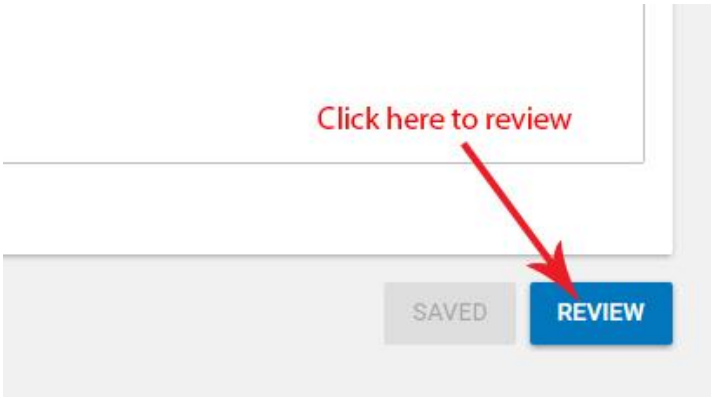
Step 15: Now you will see create release now click on it.



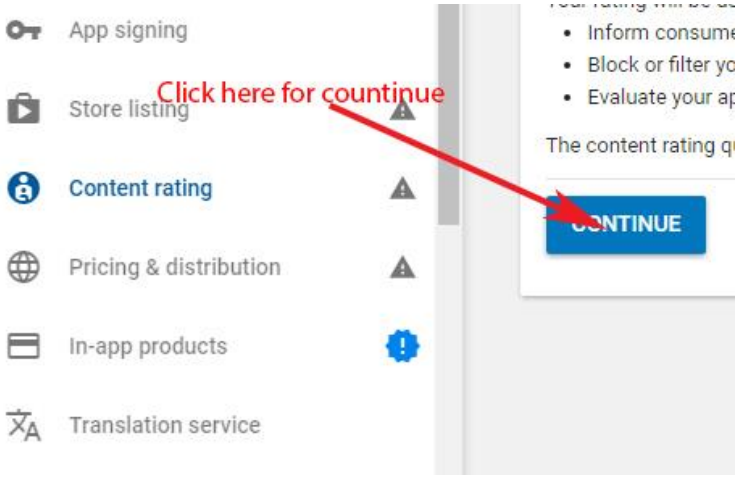
Step 16: After click on create release you will see browse files click on it and upload your signed APK.



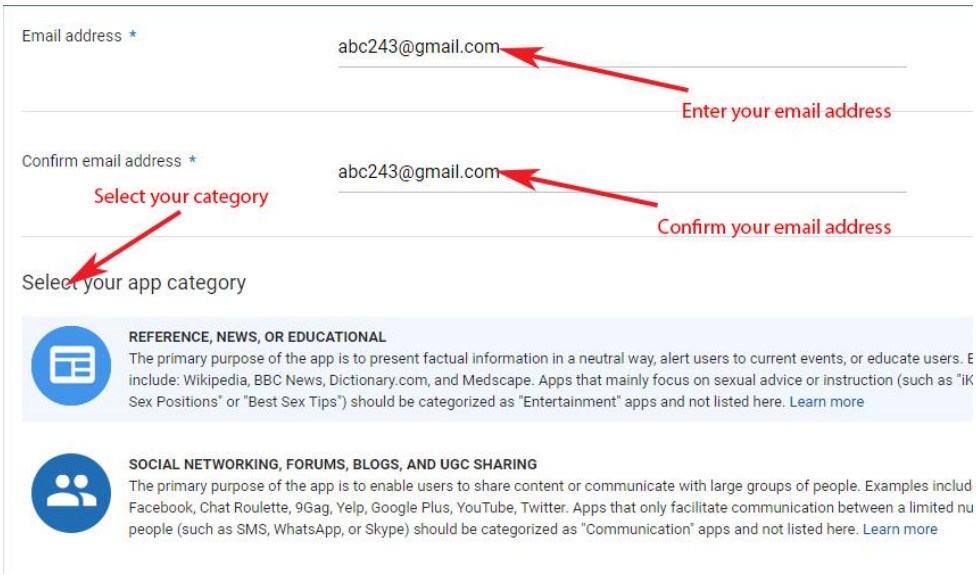
Step 17: Once the upload is successful then scroll down and click on review to check.



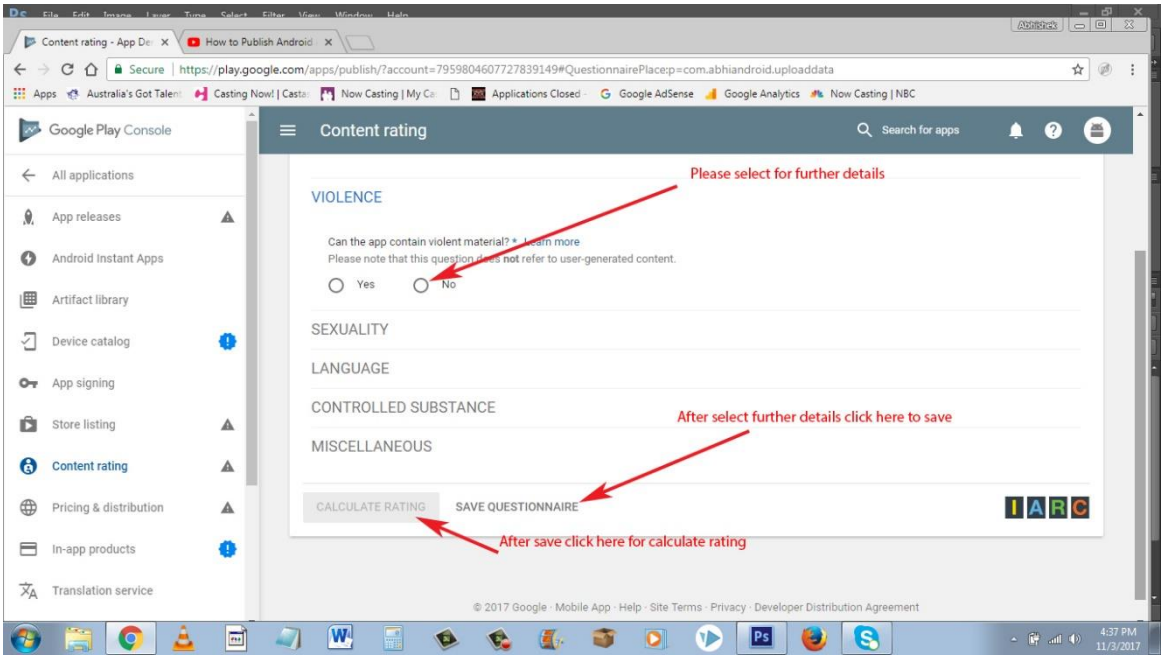
Step 18: Now go to Content Rating and click on continue.



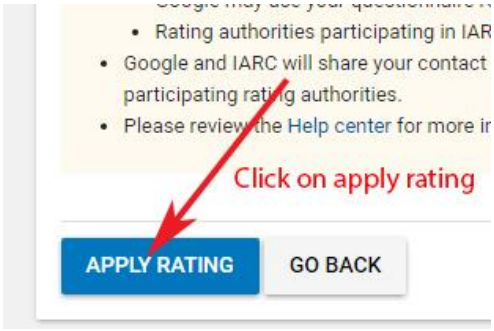
Step 19: Fill details which include email address and select your categories.



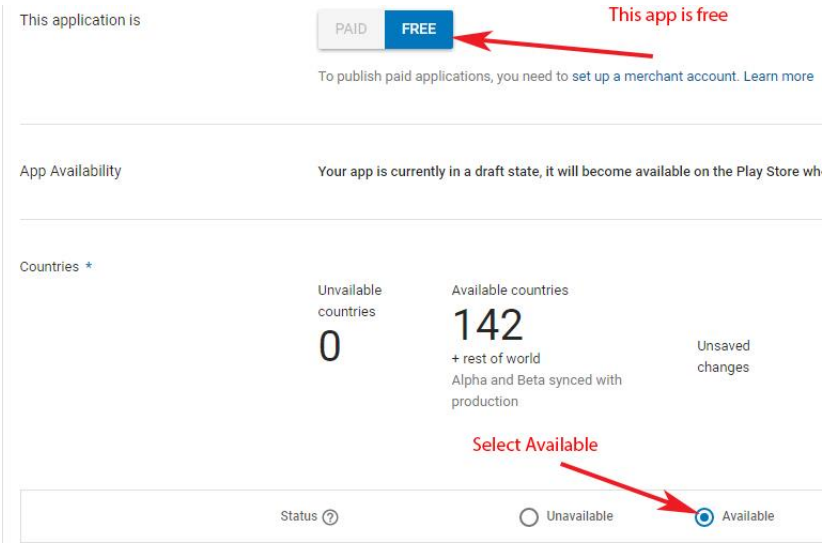
Step 20: Now select Violence, Sexuality, Language, Controlled Substance and Miscellaneous based on your App. First click on save questionnaire for save and then click on calculate rating.



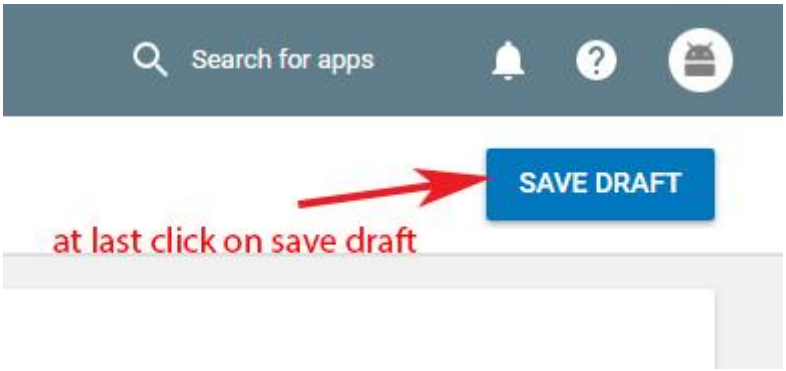
Step 21: Now click on apply rating.



Step 22: Click on pricing and distribution and select free/paid based on how you want user to access your App.

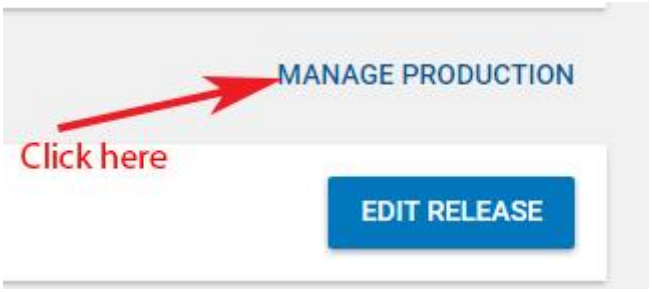


Step 23: Now scroll down and see mandatory things with * you need to select After this click on save draft .



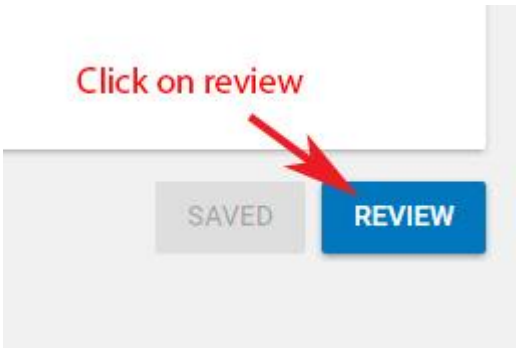
Step 24: Now Click on ready on publish along with save draft and click on Manage release.

Step 25: Click on Manage Production.

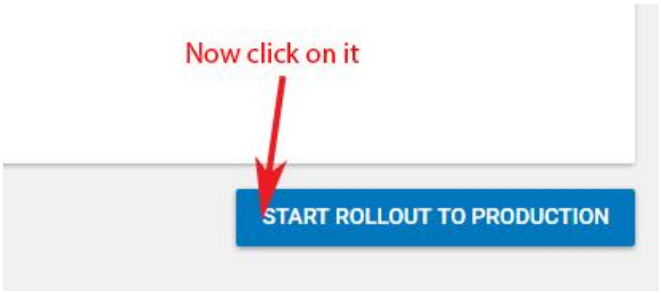


Step 26: After Manage production click on edit release.

Step 27: Now click on review.



Step 28: After review click on Start Rollout to production. Now you need to confirm. After confirm you will need to wait for one or six hour for approval.


Now click on it
START ROLLOUT TO PRODUCTION